Numerical Linear Algebra

Department of Mathematics
University of Pittsburgh

March 26, 2020

Solving Linear Systems

One of the fundamental problems in many scientific and engineering applications is to solve an algebraic **linear** system

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$$

- for the unknown vector $\boldsymbol{x}$
- when
  - the coefficient matrix $\boldsymbol{A}$, and
  - the RHS vector $\boldsymbol{b}$ are known.

Example

$$\begin{cases} 15x_1 - 2x_2 - 6x_3 = 300 \\ -2x_1 + 12x_2 - 4x_3 - x_4 = 0 \\ -6x_1 - 4x_2 + 19x_3 - 9x_4 = 0 \\ \quad - x_2 - 9x_3 + 21x_4 = 0 \end{cases}$$

with the solution $x_1 = 26.5, x_2 = 9.35, x_3 = 13.3, x_4 = 6.13.$

Such systems arise naturally in various applications such as approximating

- nonlinear equations by linear equations, or
- differential equations by algebraic equations.

The cornerstone of many numerical methods for solving a variety of practical computational problems is the **efficient** and **accurate** solution of linear systems. The system of linear algebraic equations

$$Ax = b$$

may or may not have a solution, and if it has a solution it may or may not be unique.

Solving Linear Systems

Goal: solve

$$Ax = b$$

(i) direct methods: with a finite no. of steps $\Rightarrow x$ exact solution (in exact arithmetic)

(ii) iterative methods: with large sparse systems (from discretizing PDEs)

Gaussian eliminations is the standard method for solving the linear system by using a calculator or a computer. This method is undoubtedly familiar since is the simplest way to solve a linear system by hand. When the system has no solution, other approaches are used such as linear least squares, which is discussed in Chapter 14.
In this chapter, we assume the coefficient matrix A is $n \times n$ and invertible (nonsingular).

In a pure mathematical approach, the solution to the problem of solving $Ax = b$ is simply

$$x = A^{-1}b,$$

where A^{-1} is the inverse matrix.

But in most applications, it is advisable to solve the system directly for the unknown vector x rather than explicitly computing the inverse matrix.

In applied mathematics and in many applications, it can be a daunting task for even the largest and fastest computers to solve accurately extremely large systems involving thousands or millions of unknowns. Some of the questions are the following:

1. How do we store such large systems in computer?
2. How do we know that the answers are correct?
3. What is the precision of the computed results?
4. Can the algorithm fail?
5. How long will it take to compute answers?
6. What is the asymptotic operation count of the algorithm?
7. Will the algorithm be unstable for certain systems?
8. Can instability be controlled by pivoting? (Permuting the order of the rows of the matrix is called **pivoting**.)
9. Which strategy of pivoting should be used?
10. How do we know whether the matrix is ill-conditioned and whether the answers are accurate?

Gaussian elimination transforms a linear system into an upper triangular form, which is easier to solve.
This process, in turn, is equivalent to finding the factorization

$$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U},$$

where $\boldsymbol{L}$ is a unit lower triangular matrix and $\boldsymbol{U}$ is an upper triangular matrix.
This factorization is especially useful when solving many linear systems involving the same coefficient matrix but different RHS, which occurs in various applications.

When the coefficient matrix $\boldsymbol{A}$ has a special structure such as being symmetric, positive definite, triangular, banded, block, sparse, the general approach of Gaussian elimination with partial pivoting needs to be modified or rewritten specifically for the system.

When the coefficient matrix has predominantly zero entries, the system is sparse and iterative methods can involve much less computer memory than Gaussian elimination.

Our objective in this chapter is to develop a good program for solving a system of n linear equations in n unknowns:

$$\left\{\begin{array}{lllllllllll} a_{11}x_1 & + & a_{12}x_2 & + & a_{13}x_3 & + & \cdots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & a_{23}x_3 & + & \cdots & + & a_{2n}x_n & = & b_2 \\ \vdots & & \vdots & & \vdots & & & & \vdots & & \vdots \\ a_{i1}x_1 & + & a_{i2}x_2 & + & a_{i3}x_3 & + & \cdots & + & a_{in}x_n & = & b_i \\ \vdots & & \vdots & & \vdots & & & & \vdots & & \vdots \\ a_{n1}x_1 & + & a_{n2}x_2 & + & a_{n3}x_3 & + & \cdots & + & a_{nn}x_n & = & b_n \end{array}\right. \tag{7.1}$$

In compact form, this system can be written simply as

$$\sum_{j=1}^{n} a_{ij}x_j = b_i \qquad (1 \le i \le n).$$

In these equations, a_{ij} and b_j are prescribed real numbers (data) and the unknowns x_j are to be determined. Subscripts on the letter a are separated by comma only if necessary for clarity - for example, in $a_{32,75}$, but not in a_{ij}.

Numerical example

In this section, the simplest for of Gaussian elimination is explained. The adjective **naive** applies because this form is not usually suitable for automatic computation unless essential modifications are made, as in Section 7.2. We illustrate naive Gaussian elimination with a specific example that has four equations and four unknowns:

$$\left\{\begin{array}{rcrcrcrcr} 6x_1 & - & 2x_2 & + & 2x_3 & + & 4x_4 & = & 16 \\ 12x_1 & - & 8x_2 & + & 6x_3 & + & 10x_4 & = & 26 \\ 3x_1 & - & 13x_2 & + & 9x_3 & + & 3x_4 & = & -19 \\ -6x_1 & + & 4x_2 & + & x_3 & - & 18x_4 & = & -34 \end{array}\right. \tag{7.2}$$

In the first step of the elimination procedure, certain multiples of the 1^{st} equation are subtracted from the 2^{nd}, 3^{rd}, and 4^{th} EQS so as to eliminate x_1 from these equations. Thus, we want to create 0's as the coefficients for each x_1 below the first (where 12,3, and -6 now stand). It is clear that we should 2 times the 1st EQ from the 2nd. (This multiplier is simply the quotient $\frac{12}{6}$.) Likewise, we should subtract $\frac{1}{2}$ times 1^{st} EQ from 3^{rd}. (Again, this multiplier is just $\frac{3}{6}$). Finally, we should subtract -1 times 1^{st} EQ from the 4^{th}.

When all of this has been done, the result is

$$\left\{\begin{array}{rcrcrcrcr} 6x_1 & - & 2x_2 & + & 2x_3 & + & 4x_4 & = & 16 \\ & - & 4x_2 & + & 2x_3 & + & 2x_4 & = & -6 \\ & - & 12x_2 & + & 8x_3 & + & x_4 & = & -27 \\ & & 2x_2 & + & 3x_3 & - & 14x_4 & = & -18 \end{array}\right. \tag{7.3}$$

Note that the first equation was not altered in this process, although it was used to produce the 0 coefficient in the other equations. In this context, it is called the **pivot equation**.
Notice, also, that systems (7.2) and (7.3) are *equivalent* in the following technical sense: Any solution of (7.2) is also a solution of (7.3), and vice versa. This follows at once from the fact that if equal quantities are added to equal quantities, the resulting quantities are equal. Once can get system (7.2) from (7.3) by adding 2 times the 1^{st} EQ to the 2^{nd}, and so on.

$$\left\{\begin{array}{rrrrrrrrr} 6x_1 & - & 2x_2 & + & 2x_3 & + & 4x_4 & = & 16 \\ & - & 4x_2 & + & 2x_3 & + & 2x_4 & = & -6 \\ & - & 12x_2 & + & 8x_3 & + & x_4 & = & -27 \\ & & 2x_2 & + & 3x_3 & - & 14x_4 & = & -18 \end{array}\right.$$

In the second step of the process, we mentally ignore the first equation and first column of coefficients. This leaves a system of three equations with three unknowns. The same process is now repeated using the top equation in the smaller system as the current pivot equation.
Thus we begin by subtracting 3 times the 2^{nd} EQ from 3^{rd}. (The multiplier is just the quotient $\frac{-12}{-4}$.) Then we subtract $-\frac{1}{2}$ times the 2^{nd} equation from the 4^{th}. After doing the arithmetic, we arrive at

$$\left\{\begin{array}{rrrrrrrrr} 6x_1 & - & 2x_2 & + & 2x_3 & + & 4x_4 & = & 16 \\ & - & 4x_2 & + & 2x_3 & + & 2x_4 & = & -6 \\ & & & & 2x_3 & - & 5x_4 & = & -9 \\ & & & & 4x_3 & - & 13x_4 & = & -21 \end{array}\right. \tag{7.4}$$

$$\left\{\begin{array}{rcrcrcrcr} 6x_1 & - & 2x_2 & + & 2x_3 & + & 4x_4 & = & 16 \\ & - & 4x_2 & + & 2x_3 & + & 2x_4 & = & -6 \\ & & & & 2x_3 & - & 5x_4 & = & -9 \\ & & & & 4x_3 & - & 13x_4 & = & -21 \end{array}\right.$$

The final step consists in subtracting 2 times the 3^{rd} EQ from the 4^{th}. The results is

$$\left\{\begin{array}{rcrcrcrcr} 6x_1 & - & 2x_2 & + & 2x_3 & + & 4x_4 & = & 16 \\ & - & 4x_2 & + & 2x_3 & + & 2x_4 & = & -6 \\ & & & & 2x_3 & - & 5x_4 & = & -9 \\ & & & & & - & 3x_4 & = & -3 \end{array}\right. \tag{7.5}$$

This system is said to be in **upper triangular** form. It is equivalent to system (7.2).

This completes the first phase (**forward elimination**) in the Gaussian algorithm. The second phase (**back substitution**) will solve (7.5) for the unknowns *starting at the bottom*. Thus, from the 4^{th} equation, we obtain the last unknown

$$x_4 = \frac{-3}{-3} = 1$$

Putting $x_4 = 1$ in the 3^{rd} EQ gives us

$$2x_3 - 5 = -9$$

and we find the next to last unknown

$$x_3 = \frac{-4}{2} = -2$$

and so on. The solution is

$$x_1 = 3 \qquad x_2 = 1 \qquad x_3 = -2 \qquad x_4 = 1$$

This example can be solved directly using Matlab. However, Matlab may obtain the solution by a different sequence of steps.

$$
\begin{aligned}
A = [\quad & \begin{matrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18] \end{matrix} \\
b = [\quad & \begin{matrix} 16 \\ 26 \\ -19 \\ -34] \end{matrix} \\
x = & A\backslash b
\end{aligned}
$$

Algorithm

To simplify the discussion, we write system (7.1) in matrix-vector form. The coefficient elements a_{ij} form an $n \times n$ square array, or matrix. The unknowns x_i and the RHS elements b_i form $n \times 1$ arrays, or vectors. Hence, we have

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ a_{i1} & a_{i2} & a_{i3} & \cdots & a_{in} \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_i \\ \vdots \\ b_n \end{bmatrix} \tag{7.6}$$

or

$$\boldsymbol{Ax} = \boldsymbol{b}$$

Operations between *equations* correspond to operations between *rows* in this notation. We will use these notations interchangeably.

Algorithm

Now let us organize the naive Gaussian elimination algorithm for the general system, which contains n equations and n unknowns. In this algorithm, the original data are overwritten with new computed values. In the forward elimination phase of the process, there are $n-1$ principal steps. The first of these steps uses the 1^{st} EQ to produce $n-1$ zeros as coefficients for each x_1 in all but the 1^{st} EQ. This is done by subtracting appropriate multiples of the 1^{st} EQ from the others. In this process, we refer to the 1^{st} EQ as the first **pivot equation** and to a_{11} as the first **pivot element**. For each of the remaining equations $(2 \le i \le n)$, we compute

$$\begin{cases} a_{ij} \leftarrow a_{ij} - \left(\frac{a_{i1}}{a_{11}}\right) a_{1j} \quad (1 \le j \le n) \\ b_i \leftarrow b_i - \left(\frac{a_{i1}}{a_{11}}\right) b_1 \end{cases}$$

The symbol $\leftarrow$ indicates a *replacement*. Thus, the content of the memory location allocated to a_{ij} is replaced by $a_{ij} - \left(\frac{a_{i1}}{a_{11}}\right) a_{1j}$, and so on.

Algorithm

This is acomplished by the line of pseudocode

$$a_{ij} \leftarrow a_{ij} - \left(\frac{a_{i1}}{a_{11}}\right) a_{1j}$$

Note that the quantities $\left(\frac{a_{i1}}{a_{11}}\right)$ are the **multipliers**. The new coefficient of x_1 in the i^{th} EQ will be 0 because $a_{i1} - \left(\frac{a_{i1}}{a_{11}}\right) a_{11} = 0$.
After the first step, the system will be of the form

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22} & a_{23} & \cdots & a_{2n} \\ 0 & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ 0 & a_{i2} & a_{i3} & \cdots & a_{in} \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ 0 & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_i \\ \vdots \\ b_n \end{bmatrix}$$

Algorithm

From here on, we will not alter the 1^{st} EQ, nor will alter any of the coefficients for x_1 (since a multiplier times 0 subtracted from 0 is still 0).

Thus, we can mentally ignore the first row and the first column and repeat the process on the smaller system.

With the second equation as the pivot equation, we can compute for the remaining equations $(3 \leq i \leq n)$

$$\begin{cases} a_{ij} \leftarrow a_{ij} - \left(\frac{a_{i2}}{a_{22}}\right) a_{2j} & (2 \leq j \leq n) \\ b_i \leftarrow b_i - \left(\frac{a_{i2}}{a_{22}}\right) b_2 \end{cases}$$

Just prior to the k^{th} step in the forward elimination, the system will appear as follows:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & & \cdots & & \cdots & a_{1n} \\ 0 & a_{22} & a_{23} & \cdots & & \cdots & & \cdots & a_{2n} \\ 0 & 0 & a_{33} & \cdots & & \cdots & & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & & & & & \vdots \\ 0 & 0 & 0 & \cdots & a_{kk} & \cdots & a_{kj} & \cdots & a_{kn} \\ \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & a_{ik} & \cdots & a_{ij} & \cdots & a_{in} \\ \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & a_{nk} & \cdots & a_{nj} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_k \\ \vdots \\ x_i \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_k \\ \vdots \\ b_i \\ \vdots \\ b_n \end{bmatrix}$$

Here a wedge of 0 coefficients has been created, and the first k equations have been processed and are now fixed. Using the k^{th} equation as the pivot equation, we select multipliers in order to create 0's as coefficients for each x_i below the a_{kk} coefficient.

Algorithm

Hence, we compute for each remaining equation $(k+1 \leq i \leq n)$

$$\begin{cases} a_{ij} \leftarrow a_{ij} - \left(\frac{a_{ik}}{a_{kk}}\right) a_{kj} & (k \leq j \leq n) \\ b_i \leftarrow b_i - \left(\frac{a_{ik}}{a_{kk}}\right) b_k \end{cases}$$

Obviously, we must assume that al the divisors in this algorithm are nonzero.

Pseudocode

We now consider the pseudocode for the forward elimination. The coefficient array is stored as a double-subscripted array (a_{ij}); the RHS of the system of equations is stored as a single-subscripted array (b_i); the solution is computed and stored in a single-subscripted array (x_i).
It is easy to see that the following lines of pseudocode carry out the forward elimination phase of naive Gaussian elimination:

```
**real array** $(a_{ij})_{n\times n}, (b_i)_n$
**integer** $i, j, k$
**for** $k = 1$ **to** $n-1$ **do**
    **for** $i = k+1$ **to** $n$ **do**
        **for** $j = k$ **to** $n$ **do**
            $a_{ij} \leftarrow a_{ij} - \left(\frac{a_{ik}}{a_{kk}}\right) a_{kj}$
        **end for**

    $b_i \leftarrow b_i - \left(\frac{a_{ik}}{a_{kk}}\right) b_k$
    **end for**
**end for**
```

Pseudocode

Since the multiplier $\frac{a_{ik}}{a_{kk}}$ does not depend on j, it should be moved outside the j loop.

Notice also that the new values in column k will be 0, at least theoretically, because when $j = k$, we have

$$a_{ik} \leftarrow a_{ik} - \left(\frac{a_{ik}}{a_{kk}}\right) a_{kk}$$

Since we expect this to be 0, no purpose is served in computing it.

The location where the 0 is being created is a good place to store the multiplier.

Pseudocode

If these remarks are put into practice, the pseudocode will look like this:

```
**real array** $(a_{ij})_{n\times n}, (b_i)_n$
**integer** $i, j, k$
**for** $k = 1$ **to** $n-1$ **do**
   **for** $i = k+1$ **to** $n$ **do**
      $xmult \leftarrow \frac{a_{ik}}{a_{kk}}$
      $a_{ik} \leftarrow xmult$
      **for** $j = k+1$ **to** $n$ **do**
         $a_{ij} \leftarrow a_{ij} - (xmult)a_{kj}$
      **end for**

   $b_i \leftarrow b_i - (xmult)\, b_k$
   **end for**
**end for**
```

Pseudocode

At the beginning of the back elimination phase, the linear system is of the form

$$\left\{\begin{array}{cccccccccc}
a_{11}x_1 & + & a_{12}x_2 & + & a_{13}x_3 & + & \cdots & \cdots & + & a_{1n}x_n = b_1 \\
 & & a_{22}x_2 & + & a_{23}x_3 & + & \cdots & & + & a_{2n}x_n = b_2 \\
 & & & & a_{33}x_3 & + & \cdots & & + & a_{3n}x_n = b_3 \\
 & & & & & \ddots & & & & \vdots \qquad \vdots \\
 & & & & & & a_{ii}x_i + a_{i,i+1}x_{i+1} + & \cdots & + & a_{in}x_n = b_i \\
 & & & & & & & \ddots & \vdots & \vdots \\
 & & & & & & & a_{n-1,n-1}x_{n-1} & + & a_{nn}x_n = b_{n-1} \\
 & & & & & & & & & a_{nn}x_n = b_n
\end{array}\right.$$

where the a_{ij}'s and b_i's are **not** the original ones from system (7.6) but instead are the one that have been altered by the elimination process.

The back substitution starts by solving the n^{th} EQ for x_n:

$$x_n = \frac{b_n}{a_{nn}}$$

Then, using the $(n-1)^{th}$ EQ, we solve for x_{n-1}:

$$x_{n-1} = \frac{1}{a_{n-1,n-1}} \left(b_{n-1} - a_{n-1,n}x_n\right)$$

We continue working upward, recovering each x_i by the formula

$$\boldsymbol{x_i = \frac{1}{a_{ii}} \left(b_i - \sum_{j=i+1}^{n} a_{ij}x_j\right) \qquad (i = n-1, n-2, \ldots, 1)} \tag{7.7}$$

Pseudocode

Here is the pseudocode to do this:

```
real array $(a_{ij})_{n\times n}, (x_i)_n$
integer $i, j, n$
real $sum$
$x_n \leftarrow \frac{b_n}{a_{nn}}$
for $i = n-1$ to $1$ step -1 do
    $sum \leftarrow b_i$
    for $j = i+1$ to $n$ do
        $sum \leftarrow sum - a_{ij}x_j$
    end for
    $x_i \leftarrow \frac{sum}{a_{ii}}$
end for
```

Pseudocode

```
procedure $Naive\_Gauss(n, (a_{ij}), (b_i), (x_i))$
real array $(a_{ij})_{n\times n}, (b_i)_n$
integer $i, j, k$
for $k = 1$ to $n - 1$ do
   for $i = k + 1$ to $n$ do
      $xmult \leftarrow \frac{a_{ik}}{a_{kk}}$
      $a_{ik} \leftarrow xmult$
      for $j = k + 1$ to $n$ do
         $a_{ij} \leftarrow a_{ij} - (xmult)a_{kj}$
      end for
   $b_i \leftarrow b_i - (xmult)\, b_k$
   end for
end for
$x_n \leftarrow \frac{b_n}{a_{nn}}$
for $i = n - 1$ to 1 step -1 do
   $sum \leftarrow b_i$
   for $j = i + 1$ to $n$ do
      $sum \leftarrow sum - a_{ij}x_j$
   end for
   $x_i \leftarrow \frac{sum}{a_{ii}}$
end for
end procedure $Naive\_Gauss$
```

Pseudocode

Before giving a test example, let us examine the crucial computation in our pseudocode, namely the replacement

$$a_{ij} \leftarrow a_{ij} - \left(\frac{a_{ik}}{a_{kk}}\right) a_{kj}$$

Here we must expect all quantities to be infected with roundoff error.
Such a roundoff error in a_{kj} is multiplied by the factor $\frac{a_{ik}}{a_{kk}}$.
This factor is large if the pivot element $|a_{kk}|$ is small relative to $|a_{ik}|$.

Hence we conclude, tentatively, that small pivot elements lead to large multipliers and to worse roundoff errors.

Testing pseudocode

One good way to test a procedure is to set up an artificial problem whose solution is known beforehand. Sometimes the test problem will include a parameter that can be changed to vary the difficulty.
Fixing a value of n, define the polynomial

$$p(t) = 1 + t + t^2 + \cdots + t^{n-1} = \sum_{j=1}^{n} t^{j-1}$$

The coefficients in this polynomial are all equal to 1. We shall try to recover these coefficients from n values of the polynomial. We use the values of $p(t)$ at the integers $t = i + 1$ for $i = 1, 2, \ldots, n$.
If the coefficients in the polynomial are denoted by $x_1, x_2, \ldots, x_n$, we should have

$$\sum_{j=1}^{n} (1+i)^{j-1} x_j = \frac{1}{i}\left[(1+i)^n - 1\right], \quad (1 \leq i \leq n) \qquad (7.8)$$

Testing pseudocode

Here we have used the formula for the sum of a geometric series on the RHS:

$$p(1+i) = \sum_{j=1}^{i} (1+i)^{j-1} = \frac{(1+i)^n - 1}{(1+i) - 1} = \frac{1}{i}\left[(1+i)^n - 1\right]$$

Letting $a_{ij} = (1+i)^{j-1}$ and $b_i = \frac{(1+i)^n - 1}{i}$ in equation (7.8), we have a linear system:

$$\sum_{j=1}^{n} a_{ij} x_j = b_i, \quad (1 \le i \le n) \tag{7.9}$$

Testing pseudocode

Example

Write a pseudocode that solves the system of equation (7.9) for various values of n.

Since the naive Gaussian elimination procedure $Naive\_Gauss$ can be used, all that is needed is a calling program. We use $n = 4, 5, 6, 7, 8, 9$ for the test.

```
program $Main$
integer parameter $m = 10$
real array $(a_{ij})_{m\times m}, (b_i)_m, (x_i)_m$
integer $i, j, n$
for $n = 4$ to $10$ do
   for $i = 1$ to $n$ do
      for $j = 1$ to $n$ do
         $a_{ij} \leftarrow (i+1)^{j-1}$
      end for
      $b_i \leftarrow \frac{(i+1)^n - 1}{i}$
   end for
   call $Naive\_Gauss(n, (a_{ij}), (b_i), (x_i))$
   output $n, (x_i)_n$
end for
end program $Main$
```

Testing Pseudocode

When this pseudocode was run on a machine that carries approximately seven decimal digits of accuracy, the solution was obtained with complete precision until n was changed to 9, and then the computed solution was worthless because one component exhibited a relative error of **16120%**! (Why?)

The coefficient matrix for this linear system is an example of a well-known ill-conditioned matrix called the **Vandermonde matrix**, and this accounts for the fact that the system cannot be solved accurately using naive Gaussian elimination.
What is amazing is that the trouble happens so suddenly!
When $n \geq 9$, the roundoff error present in computing x_i is propagated and magnified throughout the back substitution phase so that most of the computed values for x_i are worthless.

Condition number and Ill-Conditioning

An important quantity that has some influence in the numerical solution of a linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ is the **condition number**, which is defined as

$$\kappa(\boldsymbol{A}) = \|\boldsymbol{A}\|\|\boldsymbol{A}^{-1}\|$$

Since it involves a matrix norm and we do not want to go into the details of that here, we will just discuss some general concepts.
It turns out that it is not necessary to compute the inverse of $\boldsymbol{A}$ to obtain an estimate of the condition number.
Also, it can be shown that the condition number $\kappa(\boldsymbol{A})$ gauges the transfer of error from the matrix $\boldsymbol{A}$ and the vector $\boldsymbol{b}$ to the solution $\boldsymbol{x}$.

Condition number and Ill-Conditioning

The rule of thumb is that if

$$\kappa(\boldsymbol{A}) = 10^k,$$

then one can expect to lose at least k digits of precision in solving the system $\boldsymbol{Ax} = \boldsymbol{b}$.

If the linear system is sensitive to perturbations in the elements of $\boldsymbol{A}$, or to perturbations of the components of $\boldsymbol{b}$, then this fact is reflected in $\boldsymbol{A}$ having a large condition number. In such a case, the matrix $\boldsymbol{A}$ is said to be **ill-conditioned**.

Briefly, the larger the condition number, the more ill-conditioned the system.

Condition number and Ill-Conditioning

As an example of an ill-conditioned matrix consider the Hilbert matrix:

$$H_3 = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix}$$

We can use the Matlab commands

```
>> H3 = hilb(3)
  H3 =
  1.0000     0.5000     0.3333
  0.5000     0.3333     0.2500
  0.3333     0.2500     0.2000
>> cond(H3)
ans =
524.0568
>> det(H3)
ans =

4.6296e-04
```

to generate the matrix and then to compute both the condition number using the 2-norm and the determinant of the matrix.

Condition number and Ill-Conditioning

We find

- the condition number to be 524.0568 and
- the determinant to be 4.6296×10^{-4}.

When solving linear systems, **the condition number of the coefficient matrix measures the sensitivity of the system to errors in the data**.

When the condition number is large, the computed solution of the system may be dangerously in error!

Further checks should be made before accepting the solution as being accurate.

- Values of the **condition number near 1 indicate a well-conditioned matrix**, whereas
- **large values indicate an ill-conditioned matrix**.

Using the determinant to check for singularity is only appropriate for matrices of modest size. One can use the condition number to check for singular or near-singular matrices.

Condition number and Ill-Conditioning

A goal in the study of numerical methods is to acquire an awareness of whether a numerical result can be trusted or whether it may be suspect (and therefore in need of further analysis).

The condition number provides some evidence regarding this question when one is solving linear systems.

With the advent of sophisticated mathematical a software systems such as Matlab and others, *an estimate of the condition number is often available along with an approximate solution*, so that one can judge the trustworthiness of the results.

In fact, some solution procedures involve advanced features that depend on an estimated condition number and may switch of the solution technique from a variant of Gaussian elimination to a least squares solution for an ill-conditioned system.

Unsuspecting users may not realize that this has happened unless they look at all of the results, including the estimate of the condition number.

(Condition numbers can also be associated with other numerical problems such as locating roots of equations.)

Residual and Error Vectors

For a linear system $\boldsymbol{Ax} = \boldsymbol{b}$ having the true solution $\boldsymbol{x}$ and the computed solution $\widetilde{\boldsymbol{x}}$, we define

$\boldsymbol{e} = \widetilde{\boldsymbol{x}} - \boldsymbol{x}$	**error vector**
$\boldsymbol{r} = \boldsymbol{A}\widetilde{\boldsymbol{x}} - \boldsymbol{b}$	**residual vector**

An important relationship between the error vector and the residual vector is

$$\boldsymbol{Ae} = \boldsymbol{r}$$

Suppose that two students using different computer systems, solve the same linear system $\boldsymbol{Ax} = \boldsymbol{b}$. What algorithm and what precision each used are not known. Each vehemently claims to have the correct answer, but the two computer solutions $\widetilde{\boldsymbol{x}}$ and $\widehat{\boldsymbol{x}}$ are totally different!!

How do we determine which, if either, computed solution is correct?

Residual and Error Vectors

We can *check* them by substituting them into the original system, which is the same as computing the **residual vectors**

$$\widetilde{\boldsymbol{r}} = \boldsymbol{A}\widetilde{\boldsymbol{x}} - \boldsymbol{b} \text{ and } \widehat{\boldsymbol{r}} = \boldsymbol{A}\widetilde{\boldsymbol{x}} - \boldsymbol{b}.$$

- Of course, the computed solutions are not exact, because they each must contain some roundoff errors.
  So we would want to accept the solution with the smaller residual vector.
- However, if we knew the exact solution $\boldsymbol{x}$, then we would just compare the computed solutions with the exact solution, which is the same as computing the **error vectors** $\widetilde{e} = \widetilde{\boldsymbol{x}} - \boldsymbol{x}$ and $\widehat{e} = \widehat{\boldsymbol{x}} - \boldsymbol{x}$.
  Now the computed solution that produces the smaller error vector would most assuredly be the better answer.

Residual and Error Vectors

Since the exact solution is usually not known in applications, one would tend to accept the computed solution with the smaller residual vector.

But this may not be the best computed solution if the original problem is sensitive to roundoff errors - that is, ill-conditioned.

In fact, the question of whether a computed solution to a linear system is a good solution is **extremely difficult** and beyond the scope of this course.

Residual and Error Vectors

Example

$$A = \begin{bmatrix} 0.780 & 0.563 \\ 0.913 & 0.659 \end{bmatrix} \quad b = \begin{bmatrix} 0.217 \\ 0.254 \end{bmatrix}$$

$$\widetilde{x} = \begin{bmatrix} 0.999 \\ -1.001 \end{bmatrix} \quad \widehat{x} = \begin{bmatrix} 0.341 \\ -0.087 \end{bmatrix}$$

Computing with Matlab, we get that the residual vectors are

$$\widetilde{r} = \boldsymbol{A}\widetilde{\boldsymbol{x}} - \boldsymbol{b} = \begin{bmatrix} -0.0013 \\ -0.0016 \end{bmatrix},$$

$$\widehat{r} = \boldsymbol{A}\widehat{\boldsymbol{x}} - \boldsymbol{b} = 1.0e-06 * \begin{bmatrix} -1.0000 \\ 0 \end{bmatrix}$$

while the error vectors are (the exact solution is $\boldsymbol{x} = [\mathbf{1}, -\mathbf{1}]^{\boldsymbol{T}}$):

$$\widetilde{\boldsymbol{e}} = \widetilde{\boldsymbol{x}} - \boldsymbol{x} = \begin{bmatrix} -0.0010 \\ -0.0010 \end{bmatrix}, \quad \widehat{\boldsymbol{e}} = \widehat{\boldsymbol{x}} - \boldsymbol{x} = \begin{bmatrix} -0.6590 \\ 0.9130 \end{bmatrix}.$$

Summary

1. - The basic **forward elimination** procedure using equation k to operate on equations $k+1, k+2, \dots, n$ is

$$\begin{cases} a_{ij} \leftarrow a_{ij} - \frac{a_{ik}}{a_{kk}} a_{kj} & (k \le j \le n, k < i \le n) \\ b_i \leftarrow b_i - \frac{a_{ik}}{a_{kk}} b_k \end{cases}$$

   Here we assume that $a_{kk} \neq 0$.

   - The basic **back substitution** procedure is

$$x_i = \frac{1}{a_{ii}} \left(b_i - \sum_{j=i+1}^{n} a_{ij} x_j \right) \qquad (i = n-1, n-2, \dots, 1)$$

2. When solving the linear system $\boldsymbol{Ax} = \boldsymbol{b}$, if the true or exact solution is $\boldsymbol{x}$ and the approximate or computed solution is $\widetilde{\boldsymbol{x}}$, then important quantities are

error vectors	$\boldsymbol{e} = \widetilde{\boldsymbol{x}} - \boldsymbol{x}$
residual vectors	$\boldsymbol{r} = \boldsymbol{A}\widetilde{\boldsymbol{x}} - \boldsymbol{b}$
condition number	$\kappa(\boldsymbol{A}) = \Vert\boldsymbol{A}\Vert \Vert\boldsymbol{A}^{-1}\Vert$

Examples Where Naive Gaussian Elimination Fails

To see why the naive Gaussian elimination algorithm is unsatisfactory, consider the system

$$\begin{cases} 0x_1 & +x_2 = 1 \\ x_1 & +x_2 = 2 \end{cases} \tag{7.10}$$

The pseudocode in Section 7.1 would attempt to subtract some multiple of the first equation from the second in order to produce a 0 as the coefficient for x_1 in the second equation.

This, of course, is impossible, so the algorithm fails if $a_{11} = 0$.

Examples Where Naive Gaussian Elimination Fails

If a numerical procedure actually fails for some values of the data, then the procedure is probably untrustworthy for values of the data *near* the failing values. To test this dictum, consider the system

$$\begin{cases} \varepsilon x_1 + x_2 = 1 \\ x_1 + x_2 = 2 \end{cases} \tag{7.11}$$

in which ε is a small number different from 0. Now the naive algorithm of Section 7.1 works, and after forward elimination produces the system

$$\begin{cases} \varepsilon x_1 + \quad x_2 = 1 \\ \left(1 - \frac{1}{\varepsilon}\right) x_2 = 2 - \frac{1}{\varepsilon} \end{cases} \tag{7.12}$$

In the back substitution, the arithmetic is as follows:

$$x_2 = \frac{2 - \frac{1}{\varepsilon}}{1 - \frac{1}{\varepsilon}}, \quad x_1 = \frac{1 - x_2}{\varepsilon}$$

Examples Where Naive Gaussian Elimination Fails

$$x_2 = \frac{2 - \frac{1}{\varepsilon}}{1 - \frac{1}{\varepsilon}}, \quad x_1 = \frac{1 - x_2}{\varepsilon}.$$

Now $\frac{1}{\varepsilon}$ will be large, and so if this calculation is performed by a computer tha has a fixed word length, then for small values of ε, *both* $2 - \frac{1}{\varepsilon}$ *and* $1 - \frac{1}{\varepsilon}$ *would be computed as* $-\frac{1}{\varepsilon}$*.*

For example, in a 8-digit decimal machine with a 16-digit accumulator, when $\varepsilon = 10^{-9}$, it follows that $\frac{1}{\varepsilon} = 10^9$. In order to subtract, the computer must interpret the numbers as

$$\begin{aligned} \frac{1}{\varepsilon} = 10^9 \quad &= 0.10000\,000 \times 10^{10} = \quad 0.\mathbf{1}0000\,00000\,00000\,0 \times 10^{10} \\ 2 \quad &= 0.20000\,000 \times 10^{1} = \quad 0.00000\,0000\mathbf{2}\,00000\,0 \times 10^{10} \end{aligned}$$

Thus, $\frac{1}{\varepsilon} - 2$ is computed initially as $0.09999\,99998\,00000\,0 \times 10^{10}$ and then rounded to $0.10000\,000 \times 10^{10} = \frac{1}{\varepsilon}$.

Examples Where Naive Gaussian Elimination Fails

We conclude that for values of ε sufficiently close to 0, the computer calculates

$$x_2 \text{ as } 1 \text{ and then } x_1 \text{ as } \mathbf{0}.$$

Since the *correct* solution is

$$x_1 = \frac{1}{1-\varepsilon} \approx \mathbf{1}, \quad x_2 = \frac{1-2\varepsilon}{1-\varepsilon} \approx 1,$$

the relative error in the computed solution for x_1 is extremely large: 100%.

Examples Where Naive Gaussian Elimination Fails

Actually, the naive Gaussian elimination algorithm works **well** on systems (7.10) and (7.11) if the equations are first permuted:

$$\begin{cases} x_1 & + x_2 = 2 \\ 0x_1 & + x_2 = 1 \end{cases}$$

and

$$\begin{cases} x_1 & + x_2 = 2 \\ \varepsilon x_1 & + x_2 = 1 \end{cases}$$

The first system is easily solved obtaining $x_2 = 1$ and $x_1 = 2 - x_2 = 1$.
Moreover, the second of these systems becomes

$$\begin{cases} x_1 + & x_2 & = 2 \\ & (1-\varepsilon)x_2 & = 1 - 2\varepsilon \end{cases}$$

after the forward elimination.
Then from the back substitution, the solution is computed as

$$x_2 = \frac{1-2\varepsilon}{1-\varepsilon} \approx 1, \qquad x_1 = 2 - x_1 \approx 1.$$

Examples Where Naive Gaussian Elimination Fails

Notice that we do not have to rearrange the equations in the system: it is necessary only to select a different ***pivot row****.*

The difficulty in system (7.11) is not due simply to ε being small but rather to its being small relative to other coefficients in the same row. To verify this, consider

$$\begin{cases} x_1 & +\frac{1}{\varepsilon}x_2 = \frac{1}{\varepsilon} \\ x_1 & +\ x_2 = 2 \end{cases} \tag{7.13}$$

System (7.13) is mathematically equivalent to (7.11). The naive Gaussian elimination algorithm fails here. It produces the triangular system

$$\begin{cases} x_1 & +\frac{1}{\varepsilon}x_2 = \frac{1}{\varepsilon} \\ & (1-\frac{1}{\varepsilon})x_2 = 2-\frac{1}{\varepsilon} \end{cases}$$

and then, in the back substitution, it produces the erroneous result

$$x_2 = \frac{2-\frac{1}{\varepsilon}}{1-\frac{1}{\varepsilon}} \approx 1, \qquad x_1 = \frac{1}{\varepsilon} - \frac{1}{\varepsilon}x_2 \approx \mathbf{0}.$$

Examples Where Naive Gaussian Elimination Fails

The situation can be resolved by interchanging the two equations in (7.13):

$$\begin{cases} x_1 & + x_2 = 2 \\ x_1 & + \frac{1}{\varepsilon} x_2 = \frac{1}{\varepsilon} \end{cases}$$

Now the naive Gaussian elimination algorithm can be applied, resulting in the system

$$\begin{cases} x_1 & + x_2 & = 2 \\ & (\frac{1}{\varepsilon} - 1) x_2 & = \frac{1}{\varepsilon} - 2. \end{cases}$$

The solution is

$$x_2 = \frac{\frac{1}{\varepsilon} - 2}{\frac{1}{\varepsilon} - 1} \approx 1, \qquad x_1 = 2 - x_2 \approx 1$$

which is the correct solution.

Partial Pivoting and Complete Pivoting

Gaussian elimination with **partial pivoting** selects the pivot row to be the one with the maximum pivot entry in absolute value from those in the leading column of the reduced submatrix.

Two rows are interchanged to move the designated row into the pivot row position.

Gaussian elimination with **complete pivoting** selects the pivot entry as the maximum pivot entry from all entries in the submatrix.

(This complicates things because some of the unknowns are rearranged.)
Two rows and columns are interchanged to accomplish this.
In practice, partial pivoting is almost as good as full pivoting and involves significantly less work.
Simply picking the largest number in magnitude as is done in partial pivoting may work well, but here row scaling does not play a role - the relative sizes of entries in a row are not considered. Systems with equations having coefficients of disparate sizes may cause difficulties and should be viewed with suspicion. Sometimes a scaling strategy may ameliorate these problems.

We present Gaussian elimination with scaled partial pivoting, and the pseudocode contains an implicit pivoting scheme.

Partial Pivoting and Complete Pivoting

In certain situations, Gaussian elimination with the **simple** *partial pivoting strategy may lead to an incorrect solution.*

Consider the augmented matrix

$$\left[\begin{array}{cc|c} 2 & 2c & 2c \\ 1 & 1 & 2 \end{array}\right]$$

where c is a parameter that can take on very large numerical values and the variables are x and y.
The first row is selected as the pivot row by choosing the larger number in the first column. Since the multiplier is $\frac{1}{2}$, one step in the row reduction process brings us to

$$\left[\begin{array}{cc|c} 2 & 2c & 2c \\ 0 & 1-c & 2-c \end{array}\right]$$

Partial Pivoting and Complete Pivoting

Now suppose we are working with a computer of limited word length. So in this computer, we obtain $1 - c \approx -c$ and $2 - c \approx -c$. Consequently, the computer contains these numbers:

$$\left[\begin{array}{cc|c} 2 & 2c & 2c \\ 0 & -c & -c \end{array}\right]$$

Thus, as the solution, we obtain $y = 1$ and $x = 0$, whereas the correct solution is $x = y = 1$.

Partial Pivoting and Complete Pivoting

On the other hand, Gaussian elimination with **scaled** *partial pivoting selects the second row as the pivot row.*

$$\left[\begin{array}{cc|c} 2 & 2c & 2c \\ 1 & 1 & 2 \end{array}\right]$$

The scaling constants are $(2c, 1)$, and the larger of the two ratios for selecting the pivot row from $\{\frac{2}{2c}, 1\}$ is the second one.
Now the multiplier is 2, and one step in the row reduction process brings us to

$$\left[\begin{array}{cc|c} 0 & 2c-2 & 2c-4 \\ 1 & 1 & 2 \end{array}\right]$$

On our computer of limited word length, we find $2c - 2 \approx 2c$ and $2c - 4 \approx 2c$. Consequently, the computer contains these numbers:

$$\left[\begin{array}{cc|c} 0 & 2c & 2c \\ 1 & 1 & 2 \end{array}\right]$$

Now we obtain the correct solution, $y = 1$ and $x = 1$.

Gaussian Elimination with Scaled Partial Pivoting

These simple examples should make it clear that the **order** *in which we treat the equations significantly affects the accuracy of the elimination algorithm in the computer.*

In the naive Gaussian elimination algorithm, we use the first equation to eliminate x_1 from the equations that follow it.
Then we use the second equation to eliminate x_2 from the equations that follow it, and so on.

The order in which the equations are used as pivot equations is the **natural** order $\{1, 2, \ldots, n\}$.
Note that the last equation (equation number n) is **not** used as an operating equation in the natural ordering: At no time are multiples of it subtracted from other equations in the naive algorithm.

Gaussian Elimination with Scaled Partial Pivoting

From the previous examples it is clear that a strategy is needed for selecting new pivots at each stage in Guassian elimination.
Perhaps the best approach is

- **complete pivoting**, which involves searches over all entries in the submatrices for the largest entry in absolute value and then interchanges rows and columns to move it into the pivot position.

This would be quite expensive, since it involves a great amount of searching and data movement. However,

- searching just the first column in the submatrix at each stage accomplishes most of what is needed (avoiding all small or zeros pivots). This is **partial pivoting**, and it is the most common approach. It does not involve an examination of the elements in the rows, since it looks only at columns entries.

Gaussian Elimination with Scaled Partial Pivoting

We advocate a strategy that *simulates a scaling of the row vectors and then selects as a pivot element the relatively largest entry in the column.*

Also, rather than interchanging rows to move the desired element into the pivot position, we use an indexing array to avoid the data movement.
This procedure is not as expensive as complete pivoting, and it goes beyong partial pivoting to include an examination of all elements in the original matrix.

Of course, other strategies for selecting pivot elements could be used.

Gaussian Elimination with Scaled Partial Pivoting

The Gaussian elimination algorithm now to be described uses the equations in an order that is determined by the actual system being solved.
For instance, if the algorithm were asked to solve systems (7.10) and (7.11), the order in which the equations would be used as pivot equations would not be the natural order $\{1, 2\}$ but $\{2, 1\}$.
This order is automatically determined by the computer program.

The order in which the equations are employed is denoted by the row vector $[\ell_1, \ell_2, \ldots, \ell_n]$, where ℓ_n is not actually being used in the forward elimination phase.
Here, the ℓ_i are integers from 1 to n in a possibly different order.
We call $\boldsymbol{\ell}= [\ell_1, \ell_2, \ldots, \ell_n]$ the **index vector**.
The strategy to be described now for determining the index vector is termed **scaled partial pivoting**.

Gaussian Elimination with Scaled Partial Pivoting

At the beginning, a **scale factor** must be computed for each equation in the system. Reffering to the notation in Section 7.1, we define

$$s_i = \max_{1 \le i \le n} |a_{ij}| \qquad (1 \le i \le n)$$

These n numbers are recorded in the **scale vector** $\boldsymbol{s} = [s_1, s_2, \dots, s_n]$.

In starting the forward elimination process, we do not arbitrarily use the first equation as the pivot equation.

Instead, we use the equation for which the ratio $\frac{|a_{i1}|}{s_i}$ is greatest.

Let ℓ_1 be the first index for which this ratio is greatest.
Now appropriate multiples of equation ℓ_1 are subtracted from the other equation to create 0's as coefficients for each x_1 except in the pivot equation.

Gaussian Elimination with Scaled Partial Pivoting

The best way to keep track of the indeces is as follows: At the beginning, define the index vector $\boldsymbol{\ell}$ to be

$$\boldsymbol{\ell} = [\ell_1, \ell_2, \dots, \ell_n] = [1, 2, \dots, n].$$

(1) Select j to be the first index associated with the largest ratio in the set:

$$\left\{ \frac{|a_{\ell_i 1}|}{s_{\ell_i}} : \mathbf{1} \le i \le n \right\}$$

Now interchange ℓ_j with ℓ_1 in the index vector $\boldsymbol{\ell}$. Next, use multipliers

$$\frac{a_{\ell_i 1}}{a_{\ell_1 1}}$$

times row ℓ_1, and subtract from equations ℓ_i for $\mathbf{2} \le i \le n$.

It is important to note that only entries in $\boldsymbol{\ell}$ are being interchanged and *not* the equations. This eliminates the time-consuming and unnecessary process of moving the coefficients of equations around in the computer memory!

Gaussian Elimination with Scaled Partial Pivoting

(2) In the second step, the ratios

$$\left\{ \frac{|a_{\ell_i 2}|}{s_{\ell_i}} : \mathbf{2} \le i \le n \right\}$$

are scanned.
If j is the first index for the largest ration, interchange ℓ_j with ℓ_2 in $\boldsymbol{\ell}$.
Then multipliers

$$\frac{a_{\ell_i 2}}{a_{\ell_2 2}}$$

times equation ℓ_2 are subtracted from equations ℓ_i for $\mathbf{3} \le i \le n$.

Gaussian Elimination with Scaled Partial Pivoting

(k) At step k, select j to be the first index corresponding to the largest of the ratios,

$$\left\{ \frac{|a_{\ell_i k}|}{s_{\ell_i}} : \boldsymbol{k} \le i \le n \right\}$$

and interchange ℓ_j with ℓ_k in the index vector $\boldsymbol{\ell}$.
Then multipliers

$$\frac{a_{\ell_i k}}{a_{\ell_k k}}$$

times pivot equation ℓ_k are subtracted from equations ℓ_i for $\boldsymbol{k+1} \le i \le n$.

Gaussian Elimination with Scaled Partial Pivoting

Notice that the scale factors are not changed after each pivot step.
Intuitively, *one might think that after each step in Gaussian algorithm, the remaining (modified) coefficients should be used to recompute the scale factors instead of using the original scale vector.*
Of course, this could be done, but it is generally believed that the extra computations involved in this procedure are not worthwhile in the majority of linear systems.

Gaussian Elimination with Scaled Partial Pivoting

Example

Solve this system of linear equations:

$$\begin{cases} 0.0001x & + \quad y = 1 \\ x & + \quad y = 2 \end{cases}$$

using no pivoting, partial pivoting and scaled partial pivoting. Carry at most five significant digits of precision (rounding to see how finite precision computations and roundoff errors can affect the calculations.)

By direct substitution, it is easy to verify that the true solution is

$$x = 1.0001 \text{ and } y = 0.99990$$

to five significant digits.

Gaussian Elimination with Scaled Partial Pivoting

1. For **no pivoting**, the first equation in the original system is the pivot equation, and the multiplier is $xmult = \frac{1}{0.0001} = 10000$.
Multiplying the first equation by this multiplier and subtracting the result from the second equation, we get

$$\begin{cases} 0.0001x & + & y & = 1 \\ & & 9999y & = 9998 \end{cases}$$

From the second equation we obtain $y = \frac{9998}{9999} \approx 0.99990$.
Using this result and the first equation, we find

$$0.0001x = 1 - y = 1 - 0.999900 = 0.0001$$

and

$$x = \frac{0.0001}{0.0001} = 1.$$

Notice that we have *lost the last significant digit* in the correct value of x.

$$\kappa(A) = 2.6184$$

Gaussian Elimination with Scaled Partial Pivoting

2. We repeat the solution using **partial pivoting** in the original system. Examining first column of x coefficients (0.0001,1), we see that the second is larger, so the second equation is used is the pivot equation.
We can *interchange equations*, obtaining

$$\begin{cases} x + y = 2 \\ 0.0001x + y = 1 \end{cases}$$

The multiplier is $xmult = \frac{0.0001}{1} = 0.0001$. This multiple of the first equation is subtracted from the second equation, yielding

$$\begin{cases} x + y = 2 \\ 0.99990y = 0.99980 \end{cases}$$

We obtain

$$y = \frac{0.99980}{0.99990} = 0.9990$$

Now using the second equation and this values we find

$$x = 2 - y = 2 - 0.99990 = 1.0001.$$

Both computed values of x and y are correct to five significant digits.

Gaussian Elimination with Scaled Partial Pivoting

3. We repeat the solution using **scaled partial pivoting** on the original system.
Since the scaling constants are $s = (1, 1)$ and the ratios for determining the pivot equations are $(\frac{0.0001}{1}, \frac{1}{1})$, **the second equation is now the pivot equation**.
We do not actually interchange the equations, but can work with an index array $\ell = (\mathbf{2}, 1)$ that tell us to use the **second equation** as the first pivot equation.
The rest of the calculations are as above for partial pivoting.
The computed values of x and y are correct to five significant digits.

Gaussian Elimination with Scaled Partial Pivoting

We cannot promise that scaled partial pivoting will be better than partial pivoting, but it clearly has some advantages.

For example, suppose that someone wants to force the first equation in the original system to be the pivot equation and multiply it by a large number such as 20,000, obtaining

$$\begin{cases} 2x + 20000y = 20000 \\ x + y = 2 \end{cases}$$

- Partial pivoting *ignores the fact that the coefficients in the first equation differ by orders of magnitude* and selects the first equation as the pivot equation.
- However, scaled partial pivoting uses the scaling (20000,1), and the ratios for determining the pivot equations are $(\frac{2}{20000}, \frac{1}{1})$. **Scaled partial pivoting continues to select the second equation as the pivot equation!**

A Larger Numerical Example

We are not quite ready to write pseudocode, but let us consider what has been described in a concrete example. Consider

$$\begin{bmatrix} 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \\ 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -19 \\ -34 \\ 16 \\ 26 \end{bmatrix} \tag{7.14}$$

The index vector is $\boldsymbol{\ell} = [1, 2, 3, 4]$ at the beginning.
The scale vector does not change throughout the procedure and is $\boldsymbol{s} = [13, 18, 6, 12]$. To determine the first pivot row, we look at four ratios:

$$\left\{ \frac{|a_{\ell_i,1}|}{s_{\ell i}} : i = 1, 2, 3, 4 \right\} = \left\{ \frac{3}{13}, \frac{6}{18}, \frac{6}{6}, \frac{12}{12} \right\} \approx \{0.23, 0.33, 1.0, 1.0\}$$

We select the index j as the *first* occurrence of the largest value of these ratios. In this example, the largest of these occurs for the index $j = 3$.

A Larger Numerical Example

1. So row three is to be the pivot equation in step 1 ($k = 1$) of the elimination process. In the index vector $\boldsymbol{\ell}$, entries ℓ_k and ℓ_j are interchanged so that the new index vector is $\boldsymbol{\ell} = [3, 2, 1, 4]$.

Thus, the pivot equation is ℓ_k, which is $\ell_1 = 3$.

Now appropriate multiples of the third equation are subtracted from the other equations so as to create 0's as the coefficients for x_1 in each of those equations.

Explicitly, $\frac{1}{2}$ times row three is subtracted from row one, -1 times row three is subtracted from row two, and 2 times row three is subtracted from row four:

$$\begin{bmatrix} 0 & -12 & 8 & 1 \\ 0 & 2 & 3 & -14 \\ 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -27 \\ -18 \\ 16 \\ -6 \end{bmatrix}$$

A Larger Numerical Example

$$\begin{bmatrix} 0 & -12 & 8 & 1 \\ 0 & 2 & 3 & -14 \\ 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \end{bmatrix}, \qquad \boldsymbol{s} = \begin{bmatrix} 13 \\ 18 \\ 6 \\ 12 \end{bmatrix}.$$

2. In the next step ($k = 2$), we use the index vector $\boldsymbol{\ell} = [3, 2, \mathbf{1}, 4]$ and scan the ratios corresponding to rows two, one and four

$$\left\{ \frac{|a_{\ell_i,2}|}{s_{\ell i}} : i = 2, \mathbf{3}, 4 \right\} = \left\{ \frac{2}{18}, \frac{\mathbf{12}}{\mathbf{13}}, \frac{4}{12} \right\} \approx \{0.11, \mathbf{0.92}, 0.33\}$$

looking for the largest value.

We find that the largest is the **second** ratio, and we therefore set $j = 3$ and interchange ℓ_k with ℓ_j in the index vector.

Thus, the index vector becomes $\boldsymbol{\ell} = [3, \mathbf{1}, 2, 4]$.

A Larger Numerical Example

The pivot equation for step 2 in the elimination is now row one, and $\ell_2 = \mathbf{1}$. Next, multiples of the **first** equation are subtracted from the second equation and fourth equation. The appropriate multiples are $-\frac{1}{6}$ and $\frac{1}{3}$, respectively. the result is

$$\begin{bmatrix} 0 & -12 & 8 & 1 \\ 0 & 0 & \frac{13}{3} & -\frac{83}{6} \\ 6 & -2 & 2 & 4 \\ 0 & 0 & -\frac{2}{3} & \frac{5}{3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -27 \\ -\frac{45}{2} \\ 16 \\ 3 \end{bmatrix}$$

A Larger Numerical Example

$$\begin{bmatrix} 0 & -12 & 8 & 1 \\ 0 & 0 & \frac{13}{3} & -\frac{83}{6} \\ 6 & -2 & 2 & 4 \\ 0 & 0 & -\frac{2}{3} & \frac{5}{3} \end{bmatrix}, \qquad \boldsymbol{s} = \begin{bmatrix} 13 \\ 18 \\ 6 \\ 12 \end{bmatrix}.$$

3. The third and final step $(k=3)$ is to examine the ratios corresponding to rows two and four:

$$\left\{ \frac{|a_{\ell_i,3}|}{s_{\ell i}} : i = \mathbf{3}, 4 \right\} = \left\{ \frac{\frac{13}{3}}{18}, \frac{\frac{2}{3}}{12} \right\} \approx \{\mathbf{0.24}, 0.06\}$$

with the index vector $\boldsymbol{\ell} = [\mathbf{3}, \mathbf{1}, \mathbf{2}, 4]$.
The larger value is the first, so we set $j = 3$.

A Larger Numerical Example

Since this is step $k = 3$, interchanging ℓ_k with ℓ_j leaves index vector unchanged, $\boldsymbol{\ell} = [3, 2, 1, 4]$. The pivot equation is row **two** and $\ell_3 = 2$, and we subtract $-\frac{2}{13}$ times the **second** equation from the fourth equation.

So forward elimination phase ends with the final system

$$\begin{bmatrix} 0 & -12 & 8 & 1 \\ 0 & 0 & \mathbf{\frac{13}{3}} & -\mathbf{\frac{83}{6}} \\ 6 & -2 & 2 & 4 \\ 0 & 0 & 0 & -\frac{6}{13} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -27 \\ -\mathbf{\frac{45}{2}} \\ 16 \\ -\frac{6}{13} \end{bmatrix}$$

The order in which the pivot equations were selected is displayed in the final index vector $\boldsymbol{\ell} = [\mathbf{3}, \mathbf{1}, \mathbf{2}, 4]$.

A Larger Numerical Example

$$\boldsymbol{\ell} = [\mathbf{3}, \mathbf{1}, \mathbf{2}, 4].$$

Now reading the entries in the index vector from the last to the first, we have the order in which the back substitution is to be performed. The solution is obtained by using equation $\ell_4 = 4$ to determine x_4, and then equation $\ell_3 = 2$ to find x_3, and so on.

Carrying out the calculations, we have

$$\begin{aligned} x_4 &= \frac{1}{-\frac{6}{13}} \frac{-6}{13} = 1 \\ x_3 &= \frac{1}{\frac{13}{3}} \left[-\frac{45}{2} + \frac{83}{6} 1 \right] = -2 \\ x_2 &= \frac{1}{-12} [-27 - 8(-2) - 1(1)] = 1 \\ x_1 &= \frac{1}{6} [16 + 2(1) - 2(-2) - 4(1)] = 3 \end{aligned}$$

Hence, the solution is

$$\boldsymbol{x} = [3 \;\; 1 \;\; -2 \;\; 1]^T$$

Pseudocode

The algorithm as programmed caries out

1. the forward elimination phase on the coefficient array (a_{ij}) only.
2. The RHS array (b_i) is treated in the next phase.

This method is adopted because it is more efficient if several systems must be solved with the same array (a_{ij}) but differing arrays (b_i).

Because we wish to treat (b_i) later, it is necessary to store not only the index array, but also the various multipliers that are used.
These multipliers are conveniently stored in array (a_{ij}) in the positions where the 0 entries would have been created.

These multipliers are useful in constructing the $\boldsymbol{LU}$ factorization of the matrix $\boldsymbol{A}$, as we explain in Section 8.1.

Pseudocode

We are now ready to write a procedure for forward elimination with scaled partial pivoting.
Our approach is to modify procedure $Naive\_Gauss$ of Section 7.1 by introducing scaling and indexing arrays.

The procedure carrying out Gaussian elimination with scaled partial pivoting on the square array (a_{ij}) is called $Gauss$.

Its calling sequence is $(n, (a_{ij}), (\ell_i))$, where (a_{ij}) is the $n \times n$ coefficient array and (ℓ_i) is the index array $\boldsymbol{\ell}$.
In the pseudocode, (s_i) is the scale array, $\boldsymbol{s}$.

```
procedure $Gauss(n, (a_{ij}), (\ell_i))$
integer $i, j, k, n$;    real $r, rmax, smax, xmult$
real array $(a_{ij})_{1:n \times 1:n}, (\ell_i)_{1:n}$;    real array allocate $(s_i)_{1:n}$
for $i = 1$ to $n$ do
  $\ell_i \leftarrow i$
  $smax \leftarrow 0$
  for $j = 1$ to $n$ do
    $smax \leftarrow max(smax, |a_{ij}|)$
  end for
  $s_i \leftarrow smax$
end for
for $k = 1$ to $n - 1$ do
  $rmax \leftarrow 0$
  for $i = k$ to $n$ do
    $r \leftarrow \left|\frac{a_{\ell_i,k}}{s_{\ell_i}}\right|$
    if $(r > rmax)$ then
      $rmax \leftarrow r$
      $j \leftarrow i$
    end if
  end for
  $\ell_j \leftrightarrow \ell_k$
  for $i = k + 1$ to $n$ do
    $xmult \leftarrow \frac{a_{\ell_i,k}}{a_{\ell_k,k}}$
    $a_{\ell_i,k} \leftarrow xmult$
    for $j = k + 1$ to $n$ do
      $a_{\ell_i,j} \leftarrow a_{\ell_i,j} - (xmult)a_{\ell_k,j}$
    end for
  end for
end for
deallocate array $(s_i)$
end procedure $Gauss$
```

Pseudocode

A detailed explanation of the above procedure is now presented. In the first loop, the initial form of the index array is being established, namely $\ell_i = i$.

```
for $i = 1$ to $n$ do
    $\ell_i \leftarrow i$
    $smax \leftarrow 0$
    for $j = 1$ to $n$ do
        $smax \leftarrow max(smax, |a_{ij}|)$
    end for
    $s_i \leftarrow smax$
end for
```

The statement **for** $i = 1$ **to** n **do** initiates the principal outer loop.
The index k is the subscript of the variable whose coefficients will be made 0 in the array (a_{ij}); that is k is the index of the column in which new 0's are to be created.
Remember the the 0's in the array (a_{ij}) do not actually appear because those storage locations are used for the multipliers.

This fact can be seen in the line of procedure where $xmult$ is stored in the array (a_{ij}). (We will see this again in

Section 8.1 on the $\boldsymbol{LU}$ factorization of $\boldsymbol{A}$ for why this is done.)

Pseudocode

Once k has been set, the first task is to select the correct pivot row, which is done by computing $|\frac{a_{\ell_i,k}}{s_{\ell_i}}|$ for $i = k, k+1, \ldots, n$.
The next set of lines in the pseudocode is calculating this greatest ratio, called $rmax$ in the outline, and the index j where it occurs. Next ℓ_k and ℓ_j are interchanged in the array (ℓ_i).

The arithmetic modifications in the array (a_{ij}) due to subtracting multiples of row ℓ_k from rows $\ell_{k+1}, \ell_{k+2}, \ldots, \ell_n$ all occur in the final lines. First the multiplier is computed and stored: then the subtraction occurs in a loop.

Caution: Values in array (a_{ij}) that results as *output* from procedure $Gauss$ are not the same as those in array (a_{ij}) at *input*. If the original array must be retained, one should store a duplicate of it in another array.

Pseudocode

In the procedure $Naive\_Gauss$ for naive Gaussian elimination from Section 7.1, the RHS $\boldsymbol{b}$ was modified during the forward elimination phase; however, this was not done in the procedure $Gauss$.
Therefore, we need to update $\boldsymbol{b}$ before considering the back subsitution phase.
For simplicity, we discuss updating $\boldsymbol{b}$ for the naive for the naive forward elimination first. Stripping out the pseudocode from $Naive\_Gauss$ that involves the (b_i) array in the forward elimination phase, we obtain

```
for $1$ to $n-1$ do
    for $i = k+1$ to $n$ do
        $b_i = b_i - a_{ik}b_k$
    end for
end for
```

Pseudocode

This updates the (b_i) array based on the stored multipliers from the (a_{ij}) array.
When scaled partial pivoting is done in the forward elimination phase, such as in procedure $Gauss$, the multipliers for each step are not one below another in the (a_{ij}) array but are jumbled around.
To unravel this situation, all we have to do is introduce the index array (ℓ_i) into the above pseudocode:

```
for $1$ to $n-1$ do
    for $i = k+1$ to $n$ do
        $b_{\ell_i} = b_{\ell_i} - a_{\ell_i,k} b_{\ell_k}$
    end for
end for
```

Pseudocode

After the array b has been processed in the forward elimination, the back substitution process is carried out. It begins by solving the equation

$$a_{\ell_n,n}x_n = b_{\ell_n} \tag{7.15}$$

whence

$$x_n = \frac{b_{\ell_n}}{a_{\ell_n,n}}$$

Then the equation

$$a_{\ell_{n-1},n-1}x_{n-1} + a_{\ell_{n-1},n}x_n = b_{\ell_{n-1}}$$

is solved for x_{n-1}:

$$x_{n-1} = \frac{1}{a_{\ell_{n-1},n-1}}\left(b_{\ell_{n-1}} - a_{\ell_{n-1},n}x_n\right)$$

Pseudocode

After $x_n, x_{n-1}, \ldots, x_{i+1}$ have been determined, x_i is found from the equation

$$a_{\ell_i,i}x_i + a_{\ell_i,i+1}x_{i+1} + \ldots + a_{\ell_i,n}x_n = b_{\ell_i}$$

whose solution is

$$x_i = \frac{1}{a_{\ell_i,i}}\left(b_{\ell_i} - \sum_{j=i+1}^{n} a_{\ell_i,j}x_j\right) \tag{7.16}$$

Except for the presence of the index array ℓ_i, this is similar to the back substitution formula (7.7) in Section 7.1. obtained for the naive Gaussian elimination.

Pseudocode

The pseudocode for processing the array $\boldsymbol{b}$ and performing the back substitution phase

```
procedure $Solve(n, (a_{ij}), (b_i), (x_i))$
integer $i, k, n$;   real $sum$
real array $(a_{ij})_{1:n \times 1:n}, (\ell_i)_{1:n}, (b_i)_{1:n}, (x_i)_{1:n}$
for $k = 1$ to $n - 1$ do
    for $i = k + 1$ to $n$ do
        $b_{\ell_i} \leftarrow b_{\ell_i} - a_{\ell_i,k} b_{\ell_k}$
    end for
end for
$x_n \leftarrow \frac{b_{\ell_n}}{a_{\ell_n,n}}$
for $i = n - 1$ to $1$ step $-1$ do
    $sum \leftarrow b_{\ell_i}$
    for $j = i + 1$ to $n$ do
        $sum \leftarrow sum - a_{\ell_i,j} x_j$
    end for
    $x_i \leftarrow \frac{sum}{a_{\ell_i,i}}$
end for
end procedure $Solve$
```

Pseudocode

Here, the first loop carries out the forward elimination process on array (b_i), using arrays a_{ij} and (ℓ_i) that results from procedure $Gauss$.
The next line carries out the solution of equation (7.15).
The final part carries out equation (7.16).
The variable sum is a temporary variable for accumulating the terms in parentheses.

Long Operation Count

Solving large systems of linear equations can be expensive in computer time. To understand why, let us perform an operation count on two algorithms whose codes have been given.

We count only multiplications and divisions (long operations) because they are more time consuming than addition.
Furthermore, we lump multiplications and divisions together even though division is slower than multiplication.
In modern computers, all floating-point operations are done in hardware, so long operations may be as significant, but still gives an indication of the operational cost of Gaussian elimination.

Long Operation Count

Consider first procedure $Gauss$.

In step 1, the choice of a pivot element requires the calculation of n ratios - that is n divisions.

Then for rows $\ell_2, \ell_3, \dots, \ell_n$, we first compute a multiplier and then subtract from row ℓ_i that multiplier times row ℓ_1. The zero that is being created in this process in **not** computed.
So the elimination requires $n-1$ multiplications per row.
If we include the calculation of the multiplier, there are n long operations (divisions or multiplications) per row.

There are $n-1$ rows to be processed for a total of $n(n-1)$ operations.

If we add the cost of computing the ratios, a total of $\boldsymbol{n^2}$ operations is needed for step 1.

Long Operation Count

The next step is like step 1 except that row ℓ_1 is not affected, nor is the column of multipliers created and stored in step 1.

So step 2 will require $(n-1)^2$ multiplications or divisions because it operates on a system without row ℓ_1 and without column 1.

Continuing with this reasoning, we conclude that the total number of long operations for procedure $Gauss$ is

$$n^2+(n-1)^2+(n-2)^2+\cdots+4^2+3^2+2^2=\frac{n}{6}(n+1)(2n+1)-1\approx\boldsymbol{\frac{n^3}{3}}$$

Note that the number of long operations in this procedure grows like $\boldsymbol{\frac{n^3}{3}}$, the dominant term.

Long Operation Count

Now consider procedure $Solve$

$*$ The forward processing of the array (b_i) involves $n-1$ steps.
The first step contains $n-1$ multiplications, the second contains $n-2$ multiplications, and so on.
The total of the forward processing of array (b_i) is thus

$$(n-1)+(n-2)+\cdots+3+2+1=\frac{n}{2}(n-1)$$

$*$ In the back substitution procedure, one long operation is involved in the first step, two in the second step, and so on.
The total in the back substitution is

$$1+2+3+\cdots+n=\frac{n}{2}(n+1)$$

Thus, procedure $Solve$ involves altogether $\boldsymbol{n^2}$ long operations.

Long Operation Count

Theorem (on Long Operations)

- *The forward elimination phase of the Gaussian elimination algorithm with scaled partial pivoting, if applied only to the $n \times n$ coefficient array, involves approximately $\frac{n^3}{3}$ long operations (multiplications and divisions).*
- *Solving for x requires an additional n^2 long operations.*

An intuitive way to think this result is that the Gaussian elimination algorithm involves a triply nested for-loop.
So an $\mathcal{O}(n^3)$ algorithmic structure is driving the elimination process, and the work is heavily influenced by the cube of the number of equations and unknowns.

Numerical Stability

The **numerical stability** of a numerical algorithm is related to the accuracy of the procedure. An algorithm can have different levels of numerical stability because many computations can be achieved in various ways that are algebraically equivalent but may produce different results. A robust numerical algorithm with a high level of numerical stability is desirable.

Gaussian elimination is numerically stable for **strictly diagonally dominant** matrices or **symmetric positive definite** matrices.

(These are properties we will present in Sections 7.3 and 8.1, respectively.)

For matrices with a general dense structures, Gaussian elimination with partial pivoting is usually numerically stable in practice.

Nevertheless, there exist unstable **pathological** examples in which will fail.

An early version of Gaussian elimination can be found in a Chinese mathematics text dating from 150 B.C. ["Jiuzhang suanshu" or "The Nine Chapters on the Mathematical Art".]

Scaling

We should not confuse

- **scaling** in Gaussian elimination (which is **not** recommended) with our discussion of
- **scaled** partial pivoting in Gaussian elimination.

The word **scaling** has more than one meaning.

1. It could mean actually dividing each row by its maximum element in absolute value. We certainly do not advocate that. In other words, we do not recommend scaling of the matrix at all.
2. However, we do compute a scale array and use it in selecting the pivot element in Gaussian elimination with scaled partial pivoting.
   We **do not actually scale the rows**; we do just keep a vector of the "row infinity norms", that is the maximum element in absolute value for each row.
   This and the need for a vector of indices to keep track of the pivot rows make the algorithm somewhat complicated, but this is the price to pay for some degree of robustness in the procedure.

The simple 2×2 example in equation (7.13) shows that scaling does not help in choosing a good pivot row.

Summary

(1) In performing Gaussian elimination, *partial pivoting is highly recommended to avoid zero pivots and small pivots*.
In Gaussian elimination with scaled partial pivoting, we use

- a **scale vector** $\boldsymbol{s} = [s_1, s_2, \ldots, s_n]^T$ in which

$$s_i = \max_{1 \le j \le n} |a_{ij}| \qquad (1 \le i \le n)$$

- and **index vector** $\boldsymbol{\ell} = [\ell_1, \ell_2, \ldots, \ell_n]^T$, initially set as $\boldsymbol{\ell} = [1, 2, \ldots, n]^T$.

The scale vector or array is set once at the beginning of the algorithm.
The elements in the index vector or array are interchanged rather than the rows of the matrix $\boldsymbol{A}$, which reduces the amount of data movement considerably.

Summary

The key step in the pivoting procedure is to select j to be the first index associated with the largest ratio in the set

$$\left\{ \frac{|a_{\ell_i,k}|}{s_{\ell_i}} : k \leq i \leq n \right\}$$

and interchange ℓ_i with ℓ_k in the index array $\boldsymbol{\ell}$. Then use multipliers

$$\frac{a_{\ell_i,k}}{a_{\ell_k,k}}$$

times row ℓ_k and subtract from equations ℓ_i for $k+1 \leq i \leq n$.
The **forward elimination** from equation ℓ_i for $\ell_{k+1} \leq \ell_i \leq \ell_n$ is

$$\begin{cases} a_{\ell_i,j} \leftarrow a_{\ell_i,j} - \frac{a_{\ell_i,k}}{a_{\ell_k,k}} a_{\ell_k,j} & \qquad (\ell_k \leq \ell_j \leq \ell_n) \\ b_{\ell_i} \leftarrow b_{\ell_i} - \frac{a_{\ell_i,k}}{a_{\ell_k,k}} b_{\ell_k} & \end{cases}$$

Summary

The steps involving the vector $\boldsymbol{b}$ are usually done separately just before the back substitution phase, which we call *updating* the RHS.
The **back substitution** is

$$x_i = \frac{1}{a_{\ell_i,i}}\left(b_{\ell_i} - \sum_{j=i+1}^{n} a_{\ell_i,j}x_j\right) \qquad (i = n, n-1, n-2, \ldots, 1)$$

(2) For an $n \times n$ system of linear equations $\boldsymbol{Ax} = \boldsymbol{b}$, the forward elimination phase of the Gaussian elimination with scaled partial pivoting involves approximately $\frac{n^3}{3}$ long operations (multiplications and divisions), whereas the back substitution requires only n^2 long operations.

In many applications including several that are considered later on, extremely large linear systems that have a **banded** structure are encountered.
Banded matrices often occur in solving ordinary differential equations and partial differential equations.
It is advantageous to develop computer codes specifically designed to such linear systems, since they reduce the amount of storage used.

Of practical importance is the **tridiagonal** system. Here, all nonzero elements in the coefficient matrix must be on the main diagonal or on the two diagonals just above and below the main diagonal (usually called **superdiagonal** and **subdiagonal**, respectively).

$$\begin{bmatrix} d_1 & c_1 & & & & & & \\ a_1 & d_2 & c_2 & & & & & \\ & a_2 & d_3 & c_3 & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & a_{i-1} & d_i & c_i & & \\ & & & & \ddots & \ddots & \ddots & \\ & & & & & a_{n-2} & d_{n-1} & c_{n-1} \\ & & & & & & a_{n-1} & d_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_i \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_i \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix} \tag{7.17}$$

(All elements not in the displayed diagonals are 0's.)

Tridiagonal Systems

- A tridiagonal matrix is characterized by the condition $a_{ij} = 0$ if $|i - j| \geq 2$.
- In general, a matrix is said to have a **banded structure** is there is an integer k (less than n) such that $a_{ij} = 0$ whenever $|i - j| \geq k$.

The storage requirements for a banded matrix are less than those for a general matrix of the same size.
Thus, a $n \times n$ diagonal matrix requires only n memory location in the computer, and a tridiagonal matrix requires only $3n - 2$. This fact is important if banded matrices of very large order are being used.

For banded matrices, the Gaussian elimination algorithm can be made very efficient if it is known beforehand that pivoting is unnecessary.
This situation occurs often enough to justify special procedures.
Here, we develop a code for tridiagonal system and give a listing for the *pentadiagonal* system (in which $a_{ij} = 0$ if $|i - j| \leq 3$).

Tridiagonal Systems

The routine to be described now is called procedure Tri. It is designed to solve a system of n equations in n unknowns, as shown in equation (7.19). Both the forward elimination phase and the back substitution phase are incorporated in the procedure, and **no** pivoting is used; that is, the pivot equations are those given by the natural ordering $\{1, 2, \ldots, n\}$.
Thus, naive Gaussian elimination is used.
In step 1, we subtract $\frac{a_1}{d_1}$ times row 1 from row 2, thus creating a 0 in the a_1 position. Only the entries d_2 and b_2 are altered. Observe that c_2 is **not** altered.
In step 2, the process is repeated, using the new row 2 as the pivot row.
Here is how the d_i's and b_i's are altered in each step:

$$\begin{cases} d_2 \leftarrow d_2 - \frac{a_1}{d_1} c_1 \\ b_2 \leftarrow b_2 - \frac{a_1}{d_1} b_1 \end{cases}$$

In general, we obtain

$$\begin{cases} d_i \leftarrow d_i - \frac{a_{i-1}}{d_{i-1}} c_{i-1} \\ b_i \leftarrow b_i - \frac{a_{i-1}}{d_{i-1}} b_{i-1} \qquad (2 \le i \le n) \end{cases}$$

Tridiagonal Systems

At the end of the forward elimination phase, the form of the system is as follows:

$$\begin{bmatrix} d_1 & c_1 & & & & & & \\ & d_2 & c_2 & & & & & \\ & & d_3 & c_3 & & & & \\ & & & \ddots & \ddots & & & \\ & & & & d_i & c_i & & \\ & & & & & \ddots & \ddots & \\ & & & & & & d_{n-1} & c_{n-1} \\ & & & & & & & d_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_i \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_i \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix} \qquad (7.18)$$

Of course, the b_i's and d_i's are not as they were at the beginning of this process, but the c_i's are.

Tridiagonal Systems

The back substitution phase solves for $x_n, x_{n-1}, \ldots, x_1$ as follows:

$$x_n \leftarrow \frac{b_n}{d_n}$$
$$x_{n-1} \leftarrow \frac{1}{d_{n-1}}(b_{n-1} - c_{n-1}x_n)$$

Finally, we obtain

$$x_i \leftarrow \frac{1}{d_i}(b_i - c_i x_{i+1}) \qquad (i = n-1, n-2, \ldots, 1)$$

In procedure Tri for a tridiagonal system, we use single-dimensioned arrays $(a_i), (d_i)$ and (c_i) for the diagonals in the coefficient matrix and array (b_i) for the RHS, and store the solution in array x_i.

Tridiagonal Systems

```
**procedure** $Tri(n, (a_i), (d_i), (c_i), (b_i), (x_i))$
**integer** $i, n$; **real** $xmult$
**real array** $(a_i)_{1:n}, (d_i)_{1:n}, (c_i)_{1:n}, (b_i)_{1:n}, (x_i)_{1:n}$
**for** $i = 2$ **to** $n$ **do**
    $xmult \leftarrow \frac{a_{i-1}}{d_{i-1}}$
    $d_i \leftarrow d_i - (xmult)c_{i-1}$
    $b_i \leftarrow b_i - (xmult)b_{i-1}$
**end for**
$x_n \leftarrow \frac{b_n}{d_n}$
**for** $i = n-1$ **to** $1$ **step** $-1$ **do**
    $x_i \leftarrow \frac{b_i - c_i x_{i+1}}{d_i}$
**end for**
**end procedure** $Tri$
```

Notice that the original data in arrays (d_i) and (b_i) have been changed.

Tridiagonal Systems

A symmetric tridiagonal system arises in the cubic spline development of Chapter 9 and elsewhere. A general symmetric tridiagonal matrix has the form

$$\begin{bmatrix} d_1 & c_1 & & & & & & \\ c_1 & d_2 & c_2 & & & & & \\ & c_2 & d_3 & c_3 & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & c_{i-1} & d_i & c_i & & \\ & & & & \ddots & \ddots & \ddots & \\ & & & & & c_{n-2} & d_{n-1} & c_{n-1} \\ & & & & & & c_{n-1} & d_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_i \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_i \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix} \tag{7.19}$$

One could overwrite the RHS vector $\boldsymbol{b}$ with the solution vector $\boldsymbol{x}$ as well. Thus, as symmetric linear system can be solved with a procedure call of the form

$$\textbf{call } Tri(n, (c_i), (d_i), (c_i), (b_i), (b_i))$$

which reduces the number of linear arrays from five to three.

Strictly diagonal dominance

Since procedure Tri does not involve pivoting, it is natural to ask whether it is likely to fail. Since examples can be given to illustrate failure because of of attempted division by zero even though the coefficient matrix in equation (7.19) is nonsingular.
On the other hand, it is not easy to give the weakest possible conditions on this matrix to guarantee the success of the algorithm. We content ourselves with one property that is easily checked and commonly encountered.
If the tridiagonal coefficient matrix is diagonally dominant, the procedure Tri will not encounter zero divisors.

Definition (Strictly diagonal dominance)

A general matrix $\boldsymbol{A} = (a_{ij})_{n\times n}$ is **stricIty diagonally dominant if**

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}| \qquad (1 \leq i \leq n)$$

Strictly diagonal dominance. Example.

Exercise. Determine whether the matrices

$$A = \begin{bmatrix} 3 & 1 & -1 \\ 2 & -4 & 2 \\ 1 & 6 & 8 \end{bmatrix}; \qquad B = \begin{bmatrix} 3 & 2 & 6 \\ 1 & 8 & 1 \\ 9 & 2 & -2 \end{bmatrix}.$$

are strictly diagonally dominant.

Strictly diagonal dominance. Example.

$$A = \begin{bmatrix} 4 & -1 & 0 & 0 \\ 1 & 4 & -1 & 0 \\ 0 & 1 & 4 & -1 \\ 0 & 0 & 1 & 4 \end{bmatrix}; \quad A^{(1)} = \begin{bmatrix} 4 & -1 & 0 & 0 \\ 0 & \frac{17}{4} & -1 & 0 \\ 0 & 1 & 4 & -1 \\ 0 & 0 & 1 & 4 \end{bmatrix}$$

$$A^{(2)} = \begin{bmatrix} 4 & -1 & 0 & 0 \\ 0 & \frac{17}{4} & -1 & 0 \\ 0 & 0 & \frac{52}{17} & -1 \\ 0 & 0 & 1 & 4 \end{bmatrix}; \quad A^{(3)} = \begin{bmatrix} 4 & -1 & 0 & 0 \\ 0 & \frac{17}{4} & -1 & 0 \\ 0 & 0 & \frac{52}{17} & -1 \\ 0 & 0 & 0 & \frac{191}{52} \end{bmatrix}$$

$$B = \begin{bmatrix} 4 & -3.8 & 0 & 0 \\ .1 & 4 & -3.8 & 0 \\ 0 & .1 & 4 & -3.8 \\ 0 & 0 & .1 & 3.8 \end{bmatrix}; \quad B^{(1)} = \begin{bmatrix} 4 & -3.8 & 0 & 0 \\ 0 & 4.095 & -3.8 & 0 \\ 0 & .1 & 4 & -3.8 \\ 0 & 0 & .1 & 4 \end{bmatrix}$$

$$B^{(2)} = \begin{bmatrix} 4 & -1 & 0 & 0 \\ 0 & \frac{17}{4} & -1 & 0 \\ 0 & 0 & \frac{52}{17} & -1 \\ 0 & 0 & 1 & 4 \end{bmatrix}; \quad B^{(3)} = \begin{bmatrix} 4 & -1 & 0 & 0 \\ 0 & \frac{17}{4} & -1 & 0 \\ 0 & 0 & \frac{52}{17} & -1 \\ 0 & 0 & 0 & \frac{191}{52} \end{bmatrix}$$

Strictly diagonal dominance

In the case of the tridiagonal system (7.19), strict diagonal dominance means simply that (with $a_0 = a_n = 0$)

$$|d_i| > |a_{i-1}| + |c_i| \qquad (1 \le i \le n)$$

Let us verify that the forward elimination phase in procedure Tri preserves strictly diagonal dominance. The new coefficient matrix produced by Gaussian elimination has 0 elements where the a_i's originally stood, and new diagonal elements are determined recursively by

$$\begin{cases} \widehat{d}_1 = d_1 \\ \widehat{d}_i = d_i - \left(\frac{a_{i-1}}{\widehat{d}_{i-1}}\right) c_{i-1} \qquad (2 \le i \le n) \end{cases}$$

where $\widehat{d}_i$ denotes a new diagonal element. The c_i elements are unaltered. Now we assume that $|d_i| > |a_{i-1}| + |c_i|$, and we want to be sure that $|\widehat{d}_i| > |c_i|$.

Strictly diagonal dominance

Obviously, this is true for $i = 1$ because $\widehat{d_1} = d_1$. If it is true for index $i-1$ (that is, $|\widehat{d}_{i-1}| > |c_{i-1}|$), then it is true for index i because

$$\begin{aligned}
|\widehat{d}_i| &= \left| d_i - \left(\frac{a_{i-1}}{\widehat{d}_{i-1}} \right) c_{i-1} \right| \\
&\geq |d_i| - |a_{i-1}| \frac{|c_{i-1}|}{|\widehat{d}_{i-1}|} \\
&> |a_{i-1}| + |c_i| - |a_{i-1}| = |c_i|
\end{aligned}$$

* While the number of long operations in Gaussian elimination on full matrices is $\mathcal{O}(n^3)$, it is only $\mathcal{O}(n)$ for tridiagonal matrices.
** Also, the scaled pivoting is not needed on strictly dominant tridiagonal systems.

Pentadiagonal Systems

The principles illustrated by procedure Tri can be applied to matrices that have wider bands of nonzero elements.

A procedure called $Penta$ is given here to solve the five diagonal system

$$\begin{bmatrix} d_1 & c_1 & f_1 & & & & & & & & \\ a_1 & d_2 & c_2 & f_2 & & & & & & & \\ e_1 & a_2 & d_3 & c_3 & f_3 & & & & & & \\ & e_2 & a_3 & d_4 & c_4 & f_4 & & & & & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & & & & \\ & & & & e_{i-2} & a_{i-1} & d_i & c_i & f_i & & \\ & & & & & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & & & & & e_{n-4} & a_{n-3} & d_{n-2} & c_{n-2} & f_{n-2} \\ & & & & & & & e_{n-3} & a_{n-2} & d_{n-1} & c_{n-1} \\ & & & & & & & & e_{n-2} & a_{n-1} & d_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_i \\ \vdots \\ x_{n-2} \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ \vdots \\ b_i \\ \vdots \\ b_{n-2} \\ b_{n-1} \\ b_n \end{bmatrix}$$

In the pseudocode, the solution is placed in array (x_i).
Also, one should not use this routine if $n \leq 4$. (Why?)

Pentadiagonal Systems

```
procedure $Penta(n, (e_i), (a_i), (d_i), (c_i), (f_i), (b_i), (x_i))$
integer $i, n$;     real $r, s, xmult$
real array $(e_i)_{1:n}, (a_i)_{1:n}, (d_i)_{1:n}, (c_i)_{1:n}, (f_i)_{1:n}, (b_i)_{1:n}, (x_i)_{1:n}$
for $i = 2$ to $n - 1$ do
   $xmult \leftarrow \frac{r}{d_{i-1}}$
   $d_i \leftarrow d_i - (xmult)c_{i-1}$
   $c_i \leftarrow c_i - (xmult)f_{i-1}$
   $b_i \leftarrow b_i - (xmult)b_{i-1}$
   $xmult \leftarrow \frac{t}{d_{i-1}}$
   $r \leftarrow s - (xmult)c_{i-1}$
   $d_{i+1} \leftarrow d_{i+1} - (xmult)f_{i-1}$
   $b_{i+1} \leftarrow b_{i+1} - (xmult)b_{i-1}$
   $s \leftarrow a_{i+1}$
   $t \leftarrow e_i$
end for
$xmult \leftarrow \frac{r}{d_{n-1}}$
   $d_n \leftarrow d_n - (xmult)c_{n-1}$
   $x_n \leftarrow \frac{b_n - (xmult)b_{n-1}}{d_n}$
   $x_{n-1} \leftarrow \frac{b_{n-1} - c_{n-1}x_n}{d_{n-1}}$
for $i = n - 2$ to 1 step $-1$ do
   $x_i \leftarrow \frac{b_i - f_i x_{i+2} - c_i x_{i+1}}{d_i}$
end for

end procedure $Penta$
```

Pentadiagonal Systems

To be able to solve symmetric pentadiagonal systems with the same code and with a minimum of storage, we have used variables r, s and t to store temporarily some information rather than overwriting into arrays.
This allows us to solve a symmetric pentadiagonal system with a procedure call of the form

$$\textbf{call } Penta(n, (f_i), (c_i), (d_i), (c_i), (f_i), (b_i), (b_i))$$

which reduces the number of linear arrays from seven to four.
Of course, the original data in some of these arrays will be corrupted.
The computed solution will be stored in the (b_i) array.
Here, we assume that all linear arrays are padded with zeros to length n in order not to exceed the array dimensions in the pseudocode.

Block Pentadiagonal Systems

Many mathematical problems involve matrices with block structures. In many cases, there are advantages in exploiting the block structure in the numerical solution. This is particularly true in solving PDEs numerically.

$$\begin{bmatrix} \mathbf{D}_1 & \mathbf{C}_1 & & & & & & & \\ \mathbf{A}_1 & \mathbf{D}_2 & \mathbf{C}_2 & & & & & & \\ & \mathbf{A}_2 & \mathbf{D}_3 & \mathbf{C}_3 & & & & & \\ & & \ddots & \ddots & \ddots & & & & \\ & & & \mathbf{A}_{i-1} & \mathbf{D}_i & \mathbf{C}_i & & & \\ & & & & \ddots & \ddots & \ddots & & \\ & & & & & \mathbf{A}_{n-2} & \mathbf{D}_{n-1} & \mathbf{C}_{n-1} \\ & & & & & & \mathbf{A}_{n-1} & \mathbf{D}_n \end{bmatrix} \begin{bmatrix} \mathbf{X}_1 \\ \mathbf{X}_2 \\ \mathbf{X}_3 \\ \vdots \\ \mathbf{X}_i \\ \vdots \\ \mathbf{X}_{n-1} \\ \mathbf{X}_n \end{bmatrix} = \begin{bmatrix} \mathbf{B}_1 \\ \mathbf{B}_2 \\ \mathbf{B}_3 \\ \vdots \\ \mathbf{B}_i \\ \vdots \\ \mathbf{B}_{n-1} \\ \mathbf{B}_n \end{bmatrix}$$

where

$$\mathbf{D}_i = \begin{bmatrix} d_{2i-1} & c_{2i-1} \\ a_{2i-1} & d_{2i} \end{bmatrix}, \quad \mathbf{A}_i = \begin{bmatrix} e_{2i-1} & c_{2i-1} \\ 0 & e_{2i} \end{bmatrix} \quad \mathbf{C}_i = \begin{bmatrix} f_{2i-1} & 0 \\ c_{2i-1} & f_{2i} \end{bmatrix}.$$

Here, we assume that n is even, say $n = 2m$. If n is not even, then the system can be padded with an extra equation, say $x_{n+1} = 1$ so that the number of rows is even.

Block Pentadiagonal Systems

The algorithm for this block tridiagonal system is similar to the one for tridiagonal systems. Hence, we have the forward elimination phase

$$\begin{cases} \mathbf{D}_i \leftarrow \mathbf{D}_i - \mathbf{A}_{i-1}\mathbf{D}_{i-1}^{-1}\mathbf{C}_{i-1} \\ \mathbf{B}_i \leftarrow \mathbf{B}_i - \mathbf{A}_{i-1}\mathbf{D}_{i-1}^{-1}\mathbf{B}_{i-1} \end{cases} \qquad (2 \le i \le m)$$

and the back substitution phase

$$\begin{cases} \mathbf{X}_n \leftarrow \mathbf{D}_n^{-1}\mathbf{B}_n \\ \mathbf{X}_i \leftarrow \mathbf{D}_i(\mathbf{B}_i - \mathbf{C}_i\mathbf{X}_{i+1}) \end{cases} \qquad (m-1 \le i \le 1)$$

Here

$$\mathbf{D}_i^{-1} = \frac{1}{\Delta}\begin{bmatrix} d_{2i} & -c_{2i-1} \\ -a_{2i-1} & d_{2i-1} \end{bmatrix},$$

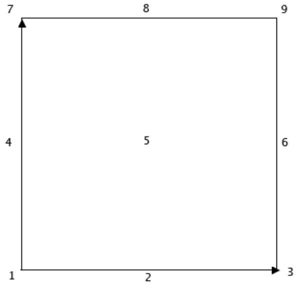
where $\Delta = d_{2i}d_{2i-1} - a_{2i-1}c_{2i-1}$.

Block Pentadiagonal Systems

The results from block pentadiagonal code are the same as those for procedure $Penta$, except for roundoff error. Also, this procedure can be used for symmetric pentadiagonal systems (in which the subdiagonals are the same as the superdiagonals).

In chapter 16 we discuss two-dimensional elliptic partial differential equations. For example, the Laplace equation is defined on the unit square.

A 3×3 mesh of points are placed over the unit square region, and they are ordered in the natural ordering (left-to-right and up).



Block Pentadiagonal Systems

In the Laplace equation, second partial derivatives are approximated by second order centered finite difference formulae. This results in an 9×9 system of linear equations having a sparse coefficient matrix with this nonzero pattern:

$$
\boldsymbol{A} = \left[\begin{array}{ccc|ccc|ccc}
\times & \times & & \times & & & & & \\
\times & \times & \times & & \times & & & & \\
& \times & \times & & & \times & & & \\
\hline
\times & & & \times & \times & & \times & & \\
& \times & & \times & \times & \times & & \times & \\
& & \times & & \times & \times & & & \times \\
\hline
& & & \times & & & \times & \times & \\
& & & & \times & & \times & \times & \times \\
& & & & & \times & & \times & \times
\end{array}\right]
$$

Here, nonzero entries are indicated by the $\times$ symbol, and zero entries are a blank. This matrix is block tridiagonal, and each nonzero block is either tridiagonal or diagonal. Other ordering of the mesh points result in sparse matrices with different patterns.

Summary

(1) For banded systems, such as tridiagonal, pentadiagonal and others, it is usual to develop special algorithms for implementing Gaussian elimination, since partial pivoting is not needed in many applications.

- The forward elimination procedure for a tridiagonal linear system $\mathbf{A} = tridiagonal[(a_i), (d_i), (c_i)]$ is

$$\begin{cases} d_i \leftarrow d_i - \left(\frac{a_{i-1}}{d_{i-1}}\right) c_{i-1} \\ b_i \leftarrow b_i - \left(\frac{a_{i-1}}{d_{i-1}}\right) b_{i-1} \qquad (2 \le i \le n) \end{cases}$$

- The backward substitution procedure is

$$x_i \leftarrow \frac{1}{d_i}(b_i - c_i x_{i+1}) \qquad (i = n-1, n-2, \ldots, 1)$$

Summary

(2) A **strictly diagonally dominant** matrix $\mathbf{A} = (a_{ij})_{n\times n}$ is one in which the magnitude of the diagonal entry is larger than the sum of the magnitudes of the off-diagonal entries in the sum row and this is true for all rows, namely,

$$|a_{ij}| > \sum_{j=1, j\neq i}^{n} |a_{ij}| \qquad (1 \leq i \leq n)$$

For strictly diagonally dominant tridiagonal coefficient matrices, partial pivoting is **not** necessary because zero divisors will **not** be encountered.
(3) The forward elimination and back substitution procedures for a pentadiagonal linear system
$\mathbf{A} = pentadiagonal[(e_i),(a_i),(d_i),(c_i),(f_i)]$ is similar to that for a tridiagonal system.

Additional Topics Concerning Systems of Linear Equations

In applications that involve partial differential equations, large linear systems arise with sparse coefficient matrices such as

$$\boldsymbol{A} = \begin{bmatrix} 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 \end{bmatrix}$$

Gaussian elimination may cause **fill-in** of the zero entries by nonzero values.

On the other hand, iterative methods preserve its sparse structure.

An $n \times n$ system of linear equations can be written in matrix form

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{8.1}$$

where the coefficient matrix $\boldsymbol{A}$ has the form

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{bmatrix}$$

Our main objective is to show that the naive Gaussian algorithm applied to $\boldsymbol{A}$ yields a factorization of $\boldsymbol{A}$ into a product of two simple matrices, one unit **lower triangular**

$$\boldsymbol{L} = \begin{bmatrix} 1 & & & & \\ \ell_{21} & 1 & & & \\ \ell_{31} & \ell_{32} & 1 & & \\ \vdots & \vdots & \ddots & \ddots & \\ \ell_{n1} & \ell_{n2} & \ell_{n,3} & \dots & 1 \end{bmatrix}$$

and the other **upper triangular**

$$\boldsymbol{U} = \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ & u_{22} & u_{23} & \cdots & u_{2n} \\ & & u_{33} & \cdots & u_{3n} \\ & & & \ddots & \vdots \\ & & & & u_{nn} \end{bmatrix}$$

In short, we refer to this as an $\boldsymbol{LU}$ **factorization** of $\boldsymbol{A}$; that is, $\boldsymbol{A} = \boldsymbol{LU}$.

Permutation matrices

Before showing how row exchanges can be used with the LU factorization approach to Gaussian elimination, we will discuss the fundamental properties of permutation matrices.

Definition

A **permutation matrix** is an $n \times n$ matrix consisting of all zeros, except for a single 1 in very row and column.

Equivalently, a permutation P is created by applying arbitrary row exchanges to the $n \times n$ identity matrix (or arbitrary column exchanges). For example

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

are the only 2×2 permutation matrices, and

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$
$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \quad \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

are the six 3×3 permutation matrices.

Permutation matrices

The next theorem tells us at a glance what action a permutation matrix causes when multiplied on the left another matrix.

Theorem

Let P be the $n \times n$ permutation matrix formed by a particular set of row exchanges applied to the identity matrix. Then, for any $n \times n$ matrix A, PA is the matrix obtained by applying exactly the same set of row exchanges to A.

For example, the permutation matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

is formed by exchanging rows 2 and 3 of the identity matrix. Multiplying an arbitrary matrix on the left with P has the effect of exchanging row 2 and 3:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = \begin{bmatrix} a & b & c \\ g & h & i \\ d & e & f \end{bmatrix}.$$

Permutation matrices

A good way to remember the Theorem is to imagine multiplying P times the identity matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

There are two different ways to view this equality

1. as a multiplication by the identity matrix (so we get the permutation matrix on the right);
2. as the permutation matrix acting on the rows of the identity matrix.

The content of the Theorem is that *the row exchanges caused by multiplication by P are exactly the ones involved in the construction of P.*

$PA = LU$ factorization

We put together what we know about Gaussian elimination into the $PA = LU$ factorization.
This is the matrix formulation of Gaussian elimination with the partial pivoting.

The $PA = LU$ factorization is the established workhorse for solving systems of linear equations.

As its names implies, the $PA = LU$ factorization is simply the LU factorization of a row-exchanged version of A.

- Under partial pivoting, the rows need exchanging are not known at the outset, so we must be careful about fitting the row exchange information into the factorization.
- In particular, we need to keep track of previous multipliers when a row exchange is made.

$PA = LU$ factorization

Example

Find the $PA = LU$ factorization of the matrix

$$A = \begin{bmatrix} 2 & 1 & 5 \\ 4 & 4 & -4 \\ 1 & 3 & 1 \end{bmatrix}$$

First, rows 1 and 2 need to be exchanged, according to partial pivoting:

$$\begin{bmatrix} 2 & 1 & 5 \\ 4 & 4 & -4 \\ 1 & 3 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 4 & 4 & -4 \\ 2 & 1 & 5 \\ 1 & 3 & 1 \end{bmatrix}, \qquad P = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Now we perform two row operations to eliminate the first column, but store the multipliers in the location of zero's.

$$\begin{bmatrix} 4 & 4 & -4 \\ \frac{1}{2} & -1 & 7 \\ \frac{1}{4} & 2 & 2 \end{bmatrix}$$

$PA = LU$ factorization

$$\begin{bmatrix} 4 & 4 & -4 \\ \frac{1}{2} & -1 & 7 \\ \frac{1}{4} & 2 & 2 \end{bmatrix}$$

Next we must make a comparison to choose the second pivot.
Since $|a_{22}| = 1 < 2 = |a_{32}|$, a row exchange is required before eliminating the second column.

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 4 & 4 & -4 \\ \frac{1}{4} & 2 & 2 \\ \frac{1}{2} & -1 & 7 \end{bmatrix} \Rightarrow \begin{bmatrix} 4 & 4 & -4 \\ \frac{1}{4} & 2 & 2 \\ \frac{1}{2} & -\frac{1}{2} & 8 \end{bmatrix}$$

This is the finished elimination. Now we can read off the $PA = LU$ factorization:

$$\underbrace{\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}}_{P} \underbrace{\begin{bmatrix} 2 & 1 & 5 \\ 4 & 4 & -4 \\ 1 & 3 & 1 \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{4} & 1 & 0 \\ \frac{1}{2} & -\frac{1}{2} & 1 \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} 4 & 4 & -4 \\ 0 & 2 & 2 \\ 0 & 0 & 8 \end{bmatrix}}_{U}$$

$PA = LU$ factorization to solve $Ax = b$

Using $PA = LU$ factorization to solve a system of equations $Ax = b$ is just a slight variant of the $A = LU$ version. Multiply through the equation $Ax = b$ by P on the left, and then proceed as before

$$\begin{aligned} PAx &= Pb \\ L\,\underbrace{Ux}_{c} &= Pb \end{aligned} \tag{8.2}$$

Solve

$$\begin{aligned} &1.\ Lc = Pb \quad \text{for } c \\ &2.\ Ux = c \quad \text{for } x \end{aligned} \tag{8.3}$$

The important thing, as mentioned earlier, is that the expensive part of the calculation, determining $PA = LU$, can be done without knowing b.
Since the resulting LU factorization is of PA, a row-permuted version of the equation coefficients, it is necessary to permute the RHS vector b in precisely the same way before proceeding with the back substitution stage. That is achieved by using PA in the first step of back substitution.

The value of the matrix formulation elimination is apparent: All the bookkeeping details of elimination and pivoting are

automatic and contained in the matrix equations.

$PA = LU$ factorization to solve $Ax = b$

Example. Use the $PA = LU$ factorization to solve the system $Ax = b$, where

$$A = \begin{bmatrix} 2 & 1 & 5 \\ 4 & 4 & -4 \\ 1 & 3 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 5 \\ 0 \\ 6 \end{bmatrix}$$

The $PA = LU$ factorization is known

$$\underbrace{\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}}_{P} \underbrace{\begin{bmatrix} 2 & 1 & 5 \\ 4 & 4 & -4 \\ 1 & 3 & 1 \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{4} & 1 & 0 \\ \frac{1}{2} & -\frac{1}{2} & 1 \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} 4 & 4 & -4 \\ 0 & 2 & 2 \\ 0 & 0 & 8 \end{bmatrix}}_{U}$$

It remains to complete the two back substitutions.

1. $Lc = Pb$: $$\underbrace{\begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{4} & 1 & 0 \\ \frac{1}{2} & -\frac{1}{2} & 1 \end{bmatrix}}_{L} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}}_{P} \begin{bmatrix} 5 \\ 0 \\ 6 \end{bmatrix} = \begin{bmatrix} 0 \\ 6 \\ 5 \end{bmatrix}$$
2. $Ux = c$: $$\underbrace{\begin{bmatrix} 4 & 4 & -4 \\ 0 & 2 & 2 \\ 0 & 0 & 8 \end{bmatrix}}_{U} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 6 \\ 8 \end{bmatrix}, \quad x = [-1, 2, 1].$$

Matlab $PA = LU$

MATLAB lu command accepts a square coefficient matrix A and return P, L and U.

$>> A = [2\ 1\ 5; 4\ 4\ \ -4; 1\ 3\ 1];$

$>> [L, U, P] = lu(A)$

$$L = \begin{array}{rrr} 1.0000 & 0 & 0 \\ 0.2500 & 1.0000 & 0 \\ 0.5000 & -0.5000 & 1.0000 \end{array}$$

$$U = \begin{array}{rrr} 4 & 4 & -4 \\ 0 & 2 & 2 \\ 0 & 0 & 8 \end{array}$$

$$P = \begin{array}{rrr} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{array}$$

$PA = LU$ factorization to solve $Ax = b$

Example. Solve the system $2x - 1 + 3x_2 = 4$, $3x_1 + 2x_2 = 1$ using $PA = LU$ factorization with partial pivoting.
In matrix form, this is the equation

$$\begin{bmatrix} 2 & 3 \\ 3 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \end{bmatrix}.$$

We begin by ignoring the RHS b. According to partial pivoting, rows 1 and 2 must be exchanged:

$$P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad A = \begin{bmatrix} 2 & 3 \\ 3 & 2 \end{bmatrix} \Rightarrow \begin{bmatrix} 3 & 2 \\ 2 & 3 \end{bmatrix} \rightarrow \begin{bmatrix} 3 & 2 \\ \mathbf{\frac{2}{3}} & \frac{5}{3} \end{bmatrix}.$$

Therefore, the $PA = LU$ factorization is

$$\underbrace{\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}}_{P} \underbrace{\begin{bmatrix} 2 & 3 \\ 3 & 2 \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} 1 & 0 \\ \mathbf{\frac{2}{3}} & 1 \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} 3 & 2 \\ 0 & \frac{5}{3} \end{bmatrix}}_{U}.$$

$PA = LU$ factorization to solve $Ax = b$

$$\underbrace{\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}}_{P} \underbrace{\begin{bmatrix} 2 & 3 \\ 3 & 2 \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} 1 & 0 \\ \mathbf{\frac{2}{3}} & 1 \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} 3 & 2 \\ 0 & \frac{5}{3} \end{bmatrix}}_{U}.$$

1. The first back substitution $Lc = Pb$ is

$$\underbrace{\begin{bmatrix} 1 & 0 \\ \mathbf{\frac{2}{3}} & 1 \end{bmatrix}}_{L} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}}_{P} \underbrace{\begin{bmatrix} 4 \\ 1 \end{bmatrix}}_{b} = \begin{bmatrix} 1 \\ 4 \end{bmatrix}$$

2. The second back substitution $Ux = c$ is

$$\underbrace{\begin{bmatrix} 3 & 2 \\ 0 & \frac{5}{3} \end{bmatrix}}_{U} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 \\ \frac{10}{3} \end{bmatrix}}_{c}$$

Therefore the solution is $x = [-1, 2]$.

Numerical Example

The system of equations (7.2) of section 7.1 can be written succinctly in matrix form

$$\begin{bmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 16 \\ 26 \\ -19 \\ -34 \end{bmatrix} \tag{8.4}$$

Furthermore, the operations that led from this system to equation (7.5) of Section 7.1, that is, the system

$$\begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 0 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 16 \\ -6 \\ -9 \\ -3 \end{bmatrix} \tag{8.5}$$

could be affected by an appropriate matrix multiplication.

Numerical Example

The forward elimination phase can be interpreted as starting from (8.1) as proceeding to

$$\boldsymbol{M}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{M}\boldsymbol{b}. \tag{8.6}$$

where $\boldsymbol{M}$ is a matrix chosen so that $\boldsymbol{M}\boldsymbol{A}$ is the coefficient matrix for system (8.5). Hence, we have

$$\boldsymbol{M}\boldsymbol{A} = \begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 0 & -3 \end{bmatrix} = \boldsymbol{U}$$

which is an upper triangular matrix.

Numerical Example

(1) The first step of naive Gaussian elimination results in equation (7.3) of Section 7.1 or the system

$$\begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & -12 & 8 & 1 \\ 0 & 2 & 3 & -14 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 16 \\ -6 \\ -27 \\ -18 \end{bmatrix}$$

This step can be accomplished by multiplying (8.1) by a lower triangular matrix M_1:

$$\boldsymbol{M_1 A x = M_1 b}$$

where

$$\boldsymbol{M_1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \mathbf{-2} & 1 & 0 & 0 \\ -\frac{\mathbf{1}}{\mathbf{2}} & 0 & 1 & 0 \\ \mathbf{1} & 0 & 0 & 1 \end{bmatrix}$$

Numerical Example

(2) To continue, step 2 resulted in equation (7.4) of Section 7.1 or the system

$$\begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 4 & -13 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 16 \\ -6 \\ -9 \\ -21 \end{bmatrix}$$

which is equivalent to

$$\boldsymbol{M_2 M_1 A x = M_2 M_1 b}$$

where

$$\boldsymbol{M_2} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 \\ 0 & \mathbf{-3} & 1 & 0 \\ 0 & \mathbf{\frac{1}{2}} & 0 & 1 \end{bmatrix}$$

Numerical Example

$$\boldsymbol{M_2} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 \\ 0 & -\mathbf{3} & 1 & 0 \\ 0 & \mathbf{\frac{1}{2}} & 0 & 1 \end{bmatrix}$$

Again, $\boldsymbol{M_2}$ differs from an identity matrix by the presence of the negatives of the multipliers in the second column from the diagonal down.

Numerical Example

(3) Finally, step 3 gives system (8.5), which is equivalent to

$$\boldsymbol{M_3 M_2 M_1 A x = M_3 M_2 M_1 b}$$

where $\boldsymbol{M_3}$

$$\boldsymbol{M_3} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \boldsymbol{-2} & 1 \end{bmatrix}$$

Now the forward elimination phase is complete, and with

$$\boldsymbol{M = M_3 M_2 M_1} \tag{8.7}$$

we have the upper triangular coefficient system (8.5).

Numerical Example

Using equations (8.6) and (8.7), we can give a different interpretation of the forward elimination phase of naive Gaussian elimination. Now we see that

$$A = M^{-1}U = M_1^{-1}M_2^{-1}M_3^{-1}U = LU$$

Since each M_k has such a special form, its inverse is obtained by simply changing the sign of the negative multiplier entries!

Numerical Example

Hence we have

$$\boldsymbol{L} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \mathbf{2} & 0 & 0 & 0 \\ \mathbf{\frac{1}{2}} & 0 & 1 & 0 \\ -\mathbf{1} & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & \mathbf{3} & 1 & 0 \\ 0 & -\mathbf{\frac{1}{2}} & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \mathbf{2} & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \mathbf{2} & 1 & 0 & 0 \\ \mathbf{\frac{1}{2}} & 3 & 1 & 0 \\ -\mathbf{1} & -\mathbf{\frac{1}{2}} & \mathbf{2} & 1 \end{bmatrix}$$

It is somewhat amazing that $\boldsymbol{L}$ is a unit triangular matrix composed of the multipliers.

Notice that in forming $\boldsymbol{L}$, we did not determine $\boldsymbol{M}^{-1} = \boldsymbol{L}$. (Why?)

Numerical Example

It is easy to verify that

$$\boldsymbol{LU} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \mathbf{2} & 1 & 0 & 0 \\ \mathbf{\frac{1}{2}} & 3 & 1 & 0 \\ \mathbf{-1} & \mathbf{-\frac{1}{2}} & \mathbf{2} & 1 \end{bmatrix} \begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 4 & -13 \end{bmatrix} = \begin{bmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{bmatrix} = \boldsymbol{A}$$

We see that $\boldsymbol{A}$ is **factored** or **decomposed** into a unit lower triangular matrix $\boldsymbol{L}$ and an upper triangular matrix $\boldsymbol{U}$.

The matrix $\boldsymbol{L}$ consists of the multipliers located in the positions of the elements they annihilated from $\boldsymbol{A}$, of unit diagonal elements, and of o upper triangular elements. In fact, we know the general form of $\boldsymbol{L}$ and can just write it down directly using the multipliers **without** forming the $\boldsymbol{M_k}$'s and the $\boldsymbol{M_k^{-1}}$'s.

The matrix $\boldsymbol{U}$ is upper triangular (not generally having unit diagonal) and is the final coefficient matrix after the forward elimination phase is completed.

Numerical Example

It should be noted that the pseudocode $Naive\_Gauss$ of Section 7.1 replaces the original coefficient matrix with its $\boldsymbol{LU}$ factorization.

- The elements of $\boldsymbol{U}$ are in the upper triangular part of (a_{ij}) array, including the diagonal.
- The entries below the main diagonal in $\boldsymbol{L}$ (that is, the multipliers) are found below the main diagonal in the (a_{ij}) array.

Since it is known that $\boldsymbol{L}$ has a unit diagonal, nothing is lost by storing the 1's.
(In fact, we have run out of room in the (a_{ij}) array anyway!)

Formal Derivation

To see formally how the Gaussian elimination (in naive form) leads to an $\boldsymbol{LU}$ factorization, it is necessary to show that each row operation used in the algorithm can be effected by multiplying $\boldsymbol{A}$ on the left by an elementary matrix.
Specifically, if we wish to subtract λ times row p from row q, we first apply this operation to the $n \times n$ identity matrix to create an elementary matrix $\boldsymbol{M_{qp}}$. Then we form the matrix product $\boldsymbol{M_{qp}A}$.

Formal Derivation

Before proceeding, let us verify that $\boldsymbol{M_{qp}A}$ is obtained by subtracting λ times row p from row q in matrix $\boldsymbol{A}$.
Assume that $p < q$, (for in the naive Gaussian elimination this is always true). Then the elements of $\boldsymbol{M_{qp}} = (m_{ij})$ are

$$m_{ij} = \begin{cases} 1 & \text{if } i = j \\ -\lambda & \text{if } i = q \text{ and } j = p \\ 0 & \text{in all other cases} \end{cases}$$

Therefore, the elements of $\boldsymbol{M_{qp}A}$ are given by

$$(\boldsymbol{M_{qp}A})_{ij} = \sum_{s=1}^{n} m_{is}a_{sj} = \begin{cases} a_{ij} & \text{if } i \neq q \\ a_{qj} - \lambda a_{pj} & \text{if } i = q \end{cases}$$

The qth row of $\boldsymbol{M_{qp}A}$ is the sum of the qth row of $\boldsymbol{A}$ and $-\lambda$ times the pth row of $\boldsymbol{A}$, as was to be proved.

Formal Derivation

The kth step of Gaussiann elimination corresponds to the matrix $\boldsymbol{M_k}$, which is the product of $n-k$ elementary matrices:

$$\boldsymbol{M_k = M_{nk} M_{n-1,k} \cdots M_{k+1,k}}$$

Notice that each elementary matrix $\boldsymbol{M_{ik}}$ here is lower triangular because $i > k$, and therefore $\boldsymbol{M_k}$ is also lower triangular.

- If we carry out the Gaussian forward elimination process on $\boldsymbol{A}$, the result will be an upper triangular matrix $\boldsymbol{U}$.
- On the other hand, the result is obtained by applying a succession of factors like $\boldsymbol{M_k}$ to the left of $\boldsymbol{A}$

Hence, the entire process is summarized by writing

$$\boldsymbol{M_{n-1} \cdots M_2 M_1 A = U}$$

Formal Derivation

Since each $\boldsymbol{M_k}$ is invertible, we have

$$\boldsymbol{A} = \boldsymbol{M_1^{-1} M_2^{-1} \cdots M_{n-1}^{-1} U}$$

Each $\boldsymbol{M_k}$ is lower triangular with 1's on its main diagonal (**unit lower triangular**).
Each inverse $\boldsymbol{M_k^{-1}}$ has the same property, and the same is true of their product.
Hence, the matrix

$$\boldsymbol{L} = \boldsymbol{M_1^{-1} M_2^{-1} \cdots M_{n-1}^{-1}} \tag{8.8}$$

is unit lower triangular, and we have

$$\boldsymbol{A} = \boldsymbol{LU}$$

This is the so-called $\boldsymbol{LU}$ **factorization** of $\boldsymbol{A}$. Our construction of it depends upon *not* encountering any 0 divisors in the algorithm. It is easy to give examples of matrices that have no $\boldsymbol{LU}$ factorization; one of the simplest is

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

Gaussian Elimination: based on the LU factorization $A = LU$

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n}, \\ a_{21} & a_{22} & \dots & a_{2n}, \\ \vdots & & & \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \Leftrightarrow A^{(1)}x = b^{(1)}$$

Assume that the pivot elements $\neq 0$.

<u>*Step 1*</u>: eliminate x_1 from equations 2,...,n. With $a_{11}^{(1)} \neq 0$, define multipliers $m_{i1} = \frac{a_{i1}^{(1)}}{a_{11}^{(1)}}$ and

$$\text{Row}\, i \leftarrow \text{Row}\, i - m_{i1} * \text{Row}\, 1 \Leftrightarrow \begin{cases} a_{ij}^{(2)} = a_{ij}^{(1)} - m_{i1}a_{1j}^{(1)}, \; j = 2, \dots, n \\ b_i^{(2)} = b_i^{(1)} - m_{i1}b_1^{(1)} \end{cases} \quad i = 2, \dots, n$$

$$\text{With} \quad A^{(2)} = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \dots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & \dots & a_{2n}^{(2)} \\ \vdots & \vdots & & \vdots \\ 0 & a_{n2}^{(2)} & \dots & a_{nn}^{(2)} \end{bmatrix}, b^{(2)} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ b_n^{(2)} \end{bmatrix}, \text{ the system is } A^{(2)}x = b^{(2)}$$

Repeat the process until $A^{(n)}$ is triangular matrix.

Step k:

$$A^{(k)} = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & & \dots & & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & & \dots & & a_{2n}^{(2)} \\ & \ddots & \ddots & & & \vdots \\ 0 & & 0 & a_{kk}^{(k)} & \dots & a_{kn}^{(k)} \\ & & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & a_{nk}^{(k)} & \dots & a_{nn}^{(k)} \end{bmatrix}, b^{(k)} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ b_k^{(k)} \\ \vdots \\ b_n^{(k)} \end{bmatrix}.$$

With $a_{kk}^{(k)} \neq 0$, define the multipliers $m_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}$ and eliminate x_k from equations $k+1, \dots, n$:

$$\texttt{Row } i \leftarrow \texttt{Row } i - m_{ik} * \texttt{Row } k \Leftrightarrow \begin{cases} a_{ij}^{(k+1)} = a_{ij}^{(k)} - m_{ik} a_{kj}^{(k)}, j = k+1, \dots, n \\ b_i^{(k+1)} = b_i^{(k)} - m_{ik} b_k^{(k)} \end{cases} \quad i = k+1, \dots, n$$

The system $A^{(k)}x = b^{(k)}$:
$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & & \dots & & a_{1n}^{(1)} \\ & \ddots & & & & \vdots \\ 0 & & a_{kk}^{(k)} & a_{kk}^{(k)} & \dots & a_{kn}^{(k)} \\ \vdots & & 0 & a_{k+1,k+1}^{(k+1)} & \dots & a_{k+1,n}^{(k+1)} \\ & & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & a_{n,k+1}^{(k+1)} & \dots & a_{nn}^{(k+1)} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_k \\ x_{k+1} \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ b_k^{(k)} \\ \vdots \\ b_n^{(k)} \end{bmatrix}.$$

After $k = 1, \dots, n$ we end up with an upper triangular system

$$A^{(n)}x = b^{(n)}: \quad \underbrace{\begin{bmatrix} a_{11}^{(1)} & \dots & & a_{1n}^{(1)} \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & \\ 0 & \dots & 0 & a_{nn}^{(n)} \end{bmatrix}}_{:=U\equiv A^{(n)}} \begin{bmatrix} x_1 \\ \vdots \\ \vdots \\ x_n \end{bmatrix} = \underbrace{\begin{bmatrix} b_1^{(1)} \\ \vdots \\ \vdots \\ b_n^{(n)} \end{bmatrix}}_{=g} \Leftrightarrow Ux = g.$$

Back substitution

$$\begin{bmatrix} u_{11} & \dots & & u_{1n} \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & \\ 0 & \dots & 0 & u_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} g_1^{(1)} \\ \vdots \\ \vdots \\ g_n^{(n)} \end{bmatrix} \Leftrightarrow \left\{ \begin{array}{r} u_{11}x_1 + u_{12}x_2 + \dots + u_{1,n-1}x_{n-1} + u_{1n}x_n = g_1 \\ \vdots \\ u_{n-1,n-1}x_{n-1} + u_{n-1,n}x_n = g_{n-1} \\ u_{nn}x_n = g_n \end{array} \right.$$

Solve for x_n, x_{n-1}, and backwards to x_1:

$$\begin{cases} x_n = \frac{g_n}{u_{nn}} \\ x_k = \frac{g_k - (u_{k,k+1} + \dots + u_{kn}x_n)}{u_{kk}}, \quad k = n-1, \dots, 1. \end{cases}$$

Theorem (the LU factorization theorem)

$$A = LU \text{ with } L = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ m_{21} & 1 & 0 & \dots & 0 \\ m_{31} & m_{32} & 1 & & \vdots \\ \vdots & & & \ddots & \\ m_{n1} & m_{n2} & \dots & m_{n,n\text{-}1} & 1 \end{pmatrix},$$

m_{ij} the multipliers from Gaussian elimination.

Proof: Recalling that $U = A^{(n)}$ we have

$$(LU)_{ij} = [m_{i1}, \dots, m_{i,i-1}, 1, 0, \dots, 0] \begin{bmatrix} u_{1j} \\ \vdots \\ u_{jj} \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Case 1. $i \le j$:

$$(LU)_{ij} = m_{i1}u_{1j} + \ldots + m_{i,i-1}u_{i-1,j} + u_{ij}$$

where $\quad u_{ij} = a_{ij}^{(i)}, \qquad j = i, \ldots, n, i = 1, \ldots, n$

Recall: $\qquad a_{ij}^{(2)} = a_{ij}^{(1)} - m_{i1}a_{1j}^{(1)}, \qquad j = 2, \ldots, n, i = 2, \ldots, n,$

or equivalently:

$$\begin{aligned} m_{i1}u_{1j} &= m_{i1}a_{1j}^{(1)} &&= a_{ij}^{(1)} - a_{ij}^{(2)} \\ m_{i2}u_{2j} &= m_{i2}a_{2j}^{(2)} &&= a_{ij}^{(2)} - a_{ij}^{(3)} \\ &\vdots && \vdots \\ m_{i,i-1}u_{i-1,j} &= m_{i,i-1}a_{i-1,j}^{(i-1)} &&= a_{ij}^{(i-1)} - a_{ij}^{(i)} \end{aligned}$$

hence

$$\begin{aligned} (LU)_{ij} &= m_{i1}u_{1j} + \ldots + m_{i,i-1}u_{i-1,j} + u_{ij} \\ &= \left(a_{ij}^{(1)} - a_{ij}^{(2)}\right) + \left(a_{ij}^{(2)} - a_{ij}^{(3)}\right) + \ldots + \left(a_{ij}^{(i-1)} - a_{ij}^{(i)}\right) + a_{ij}^{(i)} \\ &= a_{ij}^{(1)} = a_{ij}. \end{aligned}$$

Case 2. $i \ge j$:

$$\begin{aligned} (LU)_{ij} &= m_{i1}u_{1j} + \ldots + m_{ij}u_{jj} = \sum_{k=1}^{j-1} m_{ik}a_{kj}^{(k)} + m_{ij}a_{jj}^{(j)} \\ &= \sum_{k=1}^{j-1}\left(a_{ij}^{(k)} - a_{ij}^{(k+1)}\right) + a_{ij}^{(j)} = a_{ij}^{(1)} = a_{ij}. \end{aligned}$$

■

Pseudocode

The following is the pseudocode for carrying out the LU-factorization, which is sometimes called the **Doolittle factorization**

```
input $n, (a)_{ij}$
for $k = 1$ to $n$ do
    $\ell_{kk} \leftarrow 1$
    for $j = k$ to $n$ do
        $u_{kj} \leftarrow a_{kj} - \sum_{s=1}^{k-1} \ell_{ks} u_{sj}$
    end do
    for $i = k + 1$ to $n$ do
        $\ell_{ik} \leftarrow \frac{a_{ik} - \sum_{s=1}^{k-1} \ell_{is} u_{sk}}{u_{kk}}$
    end do
end do
output $(\ell_{ij}), (u_{ij})$
```

Proposition

If the factorization without pivoting exists $A = LU$, then it is unique.

Proof: Assuming $A = L_1U_1 = L_2U_2$, with L_1, L_2 invertible.

If A is nonsingular $0 \neq \det(A) = \det(L_1)\det(U_1) \Rightarrow \det(U_1) \neq 0$, and

$$L_1U_1 = L_2U_2 \Rightarrow \underbrace{L_2^{-1}L_1}_{\text{lower } \Delta} = \underbrace{U_2U_1^{-1}}_{\text{upper } \Delta} = \underbrace{D}_{\text{diagonal}} = I,$$

since L_i have 1 on the diagonal $\Rightarrow L_1 = L_2, U_1 = U_2$. ■

Example

Solve the linear system (by Gauss elimination):

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 2 & 3 \\ -1 & -3 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \\ 2 \end{bmatrix}.$$

$$\left[\begin{array}{ccc|c} 1 & 2 & 1 & 0 \\ 2 & 2 & 3 & 3 \\ -1 & -3 & 0 & 2 \end{array}\right] \xrightarrow[m_{31}=-1]{m_{21}=2} \left[\begin{array}{ccc|c} 1 & 2 & 1 & 0 \\ 0 & -2 & 1 & 3 \\ 0 & -1 & 1 & 2 \end{array}\right] \xrightarrow{m_{32}=\frac{1}{2}} \underbrace{\left[\begin{array}{ccc|c} 1 & 2 & 1 & 0 \\ 0 & -2 & 1 & 3 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{array}\right]}_{[U|g]}$$

$$\Rightarrow \qquad x_3 = 1, x_2 = -1, x_1 = 1$$

and the LU decomposition is

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & \frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \\ 0 & -2 & 1 \\ 0 & 0 & \frac{1}{2} \end{bmatrix} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 2 & 3 \\ -1 & -3 & 0 \end{bmatrix}$$

Solving Linear Systems Using $\boldsymbol{LU}$ Factorization

Once the LU factorization of $\boldsymbol{A}$ is available, we can solve the system

$$\boldsymbol{Ax} = \boldsymbol{b}$$

by writing

$$\boldsymbol{LUx} = \boldsymbol{b}$$

Then we solve two triangular systems

$$\boldsymbol{Lz} = \boldsymbol{b} \tag{8.9}$$

for $\boldsymbol{z}$ and

$$\boldsymbol{Ux} = \boldsymbol{z} \tag{8.10}$$

for $\boldsymbol{x}$. This is particularly useful for problems that involve the same coefficient matrix $\boldsymbol{A}$ and many different RHS vectors $\boldsymbol{b}$.

Solving Linear Systems Using LU Factorization

(1) Since L is unit lower triangular, z is obtained by the pseudocode

```
**real array** $(b_i)_n, (\ell_{ij})_{n\times n}, (z_i)_n$
**integer** $i, n$
$z_1 \leftarrow b_1$
**for** $i = 2$ **to** $n$ **do**
    $z_i \leftarrow b_i - \sum_{j=1}^{i-1} \ell_{ij} z_j$
**end for**
```

(2) Likewise, x is obtained by the pseudocode

```
**real array** $(u_{ij})_{n\times n}, (x_i)_n, (z_i)_n$
**integer** $i, n$
$x_n \leftarrow \frac{z_n}{u_{nn}}$
**for** $i = n - 1$ **to** $1$ **step** $-1$ **do**
    $x_i \leftarrow \frac{z_i - \sum_{j=i+1}^{n} u_{ij} x_j}{u_{ii}}$
**end for**
```

Solving Linear Systems Using $\boldsymbol{LU}$ Factorization

The first of these two algorithms applies to the forward phase Gaussian elimination to the RHS vector $\boldsymbol{b}$. [Recall that the ℓ_{ij}'s are the *multipliers* that have been stored in the array (a_{ij}).]
The easiest way to verify this assertion is to use equation (8.8) and to write the equation

$$\boldsymbol{Lz} = \boldsymbol{b}$$

in the form

$$\boldsymbol{M_1^{-1} M_2^{-1} \cdots M_1^{n-1} z = b}$$

From this, we get immediately

$$\boldsymbol{z = M_{n-1} \cdots M_2 M_1 b}$$

Thus the same operations used to reduce $\boldsymbol{A}$ to $\boldsymbol{U}$ are to be used on $\boldsymbol{b}$ in order to produce $\boldsymbol{z}$.

Solving Linear Systems Using LU Factorization

Another way to solve equation (8.9) is to note that what must be done is to form

$$\boldsymbol{M_{n-1}M_{n-2}\cdots M_2M_1b}$$

This can be accomplished using only the array (b_i) by putting the results back into $\boldsymbol{b}$; that is

$$\boldsymbol{b} \leftarrow \boldsymbol{M_k b}$$

We know what $\boldsymbol{M_k}$ looks like because it is made up of negative multipliers that have been saved in the array (a_{ij}). Clearly, we have

$$\boldsymbol{M_k b} = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & -a_{k+1,k} & 1 & & & \\ & & \vdots & & \ddots & & \\ & & -a_{ik} & & & 1 & \\ & & \vdots & & & & \ddots \\ & & -a_{nk} & & & & & 1 \end{bmatrix}$$

Solving Linear Systems Using *LU* Factorization

The entries b_1 to b_k are not changed by this multiplication, while b_i (for $i \geq k+1$) is replaced by $-a_{ik}b_k + b_i$. Hence, the following pseudocode updates the array (b_i) on the stored multipliers in the array a:

```
real array $(a_{ij})_{n\times n}, (b_i)_n$
integer $i, k, n$
for $k = 1$ to $n-1$ do
    for $i = k+1$ to $n$ do
        $b_i \leftarrow b_i - a_{ik}b_k$
    end for
end for
```

This pseudocode should be familiar. It is the process for updating $\boldsymbol{b}$ from Section 7.2.

The algorithm for solving equation (8.10) is the back substitution phase of the naive Gaussian elimination process.

I. Direct Methods

1. Forward Elimination

Forward solve for y: $Ly = b$, L lower Δ

$$\begin{pmatrix} \ell_{11} & 0 & 0 & \dots & 0 & 0 \\ \ell_{21} & \ell_{22} & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \ell_{n-1,1} & \ell_{n-1,2} & \ell_{n-1,3} & \dots & \ell_{n-1,n-1} & 0 \\ \ell_{n1} & \ell_{n2} & \ell_{n,3} & \dots & \ell_{n,n-1} & \ell_{n,n} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1,n-1} \\ y_{nn} \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1,n-1} \\ b_{nn} \end{pmatrix}$$

hence

$$\ell_{11}y_1 = b_1, \qquad\qquad y_1 = b_1/\ell_{11},$$

$$\ell_{21}y_1 + \ell_{22}y_2 = b_2, \qquad\qquad y_2 = (b_2 - \ell_{21}y_1)/\ell_{22},$$

and so on: $y_i = \frac{1}{\ell_{ii}}\Big(b_i - \sum_{j=1}^{i-1} \ell_{ij}y_j\Big), i = 2, \dots, n.$

Operation count for Forward Elimination

	$+/-$	$*/:$
y_1	0	1
y_2	1	2
$\vdots$	$\vdots$	$\vdots$
y_i	$i-1$	i
$\vdots$	$\vdots$	$\vdots$
total	$\sum_{i=1}^{n-1} i = \frac{(n-1)n}{2}$	$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

Total cost: n^2

An operations count for solving $Ax = b$ using LU factorization

How many operations are needed for LU factorization?

Step 1. $A^{(1)} \to A^{(2)}, b^{(1)} \to b^{(2)}$:

$$\left\{\begin{array}{l} a_{ij}^{(2)} = a_{ij}^{(1)} - m_{i1}a_{1j}^{(1)}, {\scriptstyle j = 2, \ldots, n} \\ b_i^{(2)} = b_i^{(1)} - m_{i1}b_1^{(1)} \end{array}\right. \quad {\scriptstyle i = 2, \ldots, n} \Rightarrow \begin{array}{l} (n-1)^2 \text{ additions and multiplications} \\ n-1 \text{ additions and multiplications} \end{array}$$

Step 2. $A^{(2)} \to A^{(3)} : (n-2)^2 +$ and $*$, $b^{(2)} \to b^{(3)} : n-2 +$ and $*$

Step $n-1$. $A^{(n)} \to A^{(n-1)} : 1 +$ and $*$, $b^{(n-1)} \to b^{(n)} : 1 +$ and $*$

To get $U = A^{(n)}$ we need:

- Divisions: $(n-1) + (n-2) + \cdots + 1 = \frac{n(n-1)}{2}$
- Additions: $(n-1)^2 + (n-2)^2 + \cdots + 1 = \frac{n(n-1)(2n-1)}{6}$
- Multiplications: $(n-1)^2 + (n-2)^2 + \cdots + 1 = \frac{n(n-1)(2n-1)}{6}$

An operations count for solving $Ax = b$ using LU factorization

To get $b^{(1)} \to b^{(n)}$ we need:

- Additions: $(n-1) + (n-2) + \cdots + 1 = \frac{n(n-1)}{2}$
- Multiplications: $(n-1) + (n-2) + \cdots + 1 = \frac{n(n-1)}{2}$

To solve $Ux = g$ we need:

- Divisions: n
- Additions: $\frac{n(n-1)}{2}$
- Multiplications: $\frac{n(n-1)}{2}$

In total:

- $*/:$ $\underbrace{\frac{n(n^2-1)}{3}}_{A\to U} + \underbrace{\frac{n(n-1)}{2}}_{b^{(1)}\to g} + \underbrace{\frac{n(n+1)}{2}}_{\text{Solve for } x} = \frac{n^3}{3} + O(n^2)$
- $+/-$ $\underbrace{\frac{n(n-1)(2n-1)}{6}}_{A\to U} + \underbrace{\frac{n(n-1)}{2}}_{b^{(1)}\to g} + \underbrace{\frac{n(n-1)}{2}}_{\text{Solve for } x} = \frac{n^3}{3} + O(n^2)$

An operations count for solving $Ax = b$ using LU factorization

Conclusion: The cost of solving $Ax = b$ is

$$\frac{2}{3}n^3 + O(n^2)$$

Note that if the size of the linear system doubles, the cost of only factoring the matrix increases by a factor of 8.
If n is large, the principal cost in the solution of a linear system $Ax = b$ is the factorization $A = LU$.

Remark

In practice we need to solve $Ax = b$ with varying vectors b:

$$Ax^{(i)} = b^{(i)} \qquad \text{for } 1 \le i \le m$$

- *Cost of LU: $\frac{2}{3}n^3 + O(n^2)$*
- *Cost of back substitution: mn^2*
- *Cost of forward elimination: mn^2*

Total cost: $\frac{2}{3}n^3 + 2mn^2 + O(n^2)$

An operations count for solving $Ax = b$ calculating an inverse A^{-1}

Computing A^{-1}

Is equivalent to solving

$$AX = I \Leftrightarrow Ax^{(i)} = e^i$$

where $x^{(i)}$ is the i^{th} column of A^{-1}.

Hence the cost is $(m = n)$:

$$\frac{2}{3}n^3 + 2n^3 + O(n^2) = \frac{8}{3}n^3 + O(n^2)$$

hence to compute

A^{-1} is 4 times more expansive than finding the solution of $Ax = b$.

LDL^T Factorization

In the LDL^T factorization, L is unit lower triangular, and D is a diagonal matrix. This factorization can be carried out if

1. A is symmetric and
2. has an ordinary LU factorization,

with L unit lower triangular.

To see this, we start with

$$LU = A = A^T = (LU)^T = U^T L^T$$

Since L is unit lower triangular, it is invertible, and we can write

$$U = L^{-1} U^T L^T.$$

Then

$$U(L^T)^{-1} = L^{-1} U^T.$$

Since the RHS of this equation is lower triangular and the LHS is upper triangular, both sides are diagonal, say, D. From the equation $U(L^T)^{-1} = D$, we have $U = DL^T$ and

$$A = LU = LDL^T.$$

$\boldsymbol{LDL^T}$ Factorization

We now derive the pseudocode for obtaining the $\boldsymbol{LDL^T}$ factorization of a symmetric matrix $\boldsymbol{A}$ in which $\boldsymbol{L}$ is unit lower triangular and $\boldsymbol{D}$ is diagonal. In our analysis, we write a_{ij} as generic elements of $\boldsymbol{A}$ and ℓ_{ij} as generic elements of $\boldsymbol{L}$. The diagonal $\boldsymbol{D}$ has elements d_{ii} or d_i.
From the equation $\boldsymbol{A} = \boldsymbol{LDL^T}$, we have

$$a_{ij} = \sum_{\nu=1}^{n}\sum_{\mu=1}^{n} \ell_{i\nu} d_{\nu\mu} \ell_{\mu j}^T = \sum_{\nu=1}^{n}\sum_{\mu=1}^{n} \ell_{i\nu} d_{\nu} \delta_{\nu\mu} \ell_{j\mu} = \sum_{\nu=1}^{n} \ell_{i\nu} d_\nu \ell_{j\nu} \quad (1 \le i, j \le n)$$

Use the fact that $\ell_{ij} = 0$ when $j > i$ and $\ell_{ii} = 1$ to continue the argument

$$a_{ij} = \sum_{\nu=1}^{\min(i,j)} \ell_{i\nu} d_\nu \ell_{j\nu} \quad (1 \le i, j \le n)$$

LDL^T Factorization

Assume now that $j \leq i$. Then

$$a_{ij} = \sum_{\nu=1}^{j} \ell_{i\nu} d_\nu \ell_{j\nu} = \sum_{\nu=1}^{j-1} \ell_{i\nu} d_\nu \ell_{j\nu} + \ell_{ij} d_j \ell_{jj} = \sum_{\nu=1}^{j-1} \ell_{i\nu} d_\nu \ell_{j\nu} + \ell_{ij} d_j \quad (1 \leq i, j$$

In particular, let $j = i$. We get

$$a_{ii} = \sum_{\nu=1}^{i-1} \ell_{i\nu} d_\nu \ell_{i\nu} + d_i \quad (1 \leq i \leq n)$$

Equivalently, we have

$$d_i = a_{ii} - \sum_{\nu=1}^{i-1} d_\nu \ell_{i\nu}^2 \quad (1 \leq i \leq n)$$

Particular cases of this are

$$d_1 = a_{11}, \quad d_2 = a_{22} - d_1 \ell_{21}^2, \quad d_3 = a_{33} - d_1 \ell_{31}^2 - d_2 \ell_{32}^2, \text{ etc.}$$

LDL^T Factorization

Now we can limit our attention to the cases $1 \le j < i \le n$, where we have

$$a_{ij} = \sum_{\nu=1}^{j-1} \ell_{i\nu} d_\nu \ell_{j\nu} + \ell_{ij} d_j \quad (1 \le i \le n)$$

Solving for ℓ_{ij}, we obtain

$$\ell_{ij} = \frac{a_{ij} - \sum_{\nu=1}^{j-1} \ell_{i\nu} d_\nu \ell_{j\nu}}{d_j} \quad (1 \le i \le n)$$

Taking $j = 1$, we have

$$\ell_{i1} = \frac{a_{i1}}{d_1} \quad (2 \le i \le n)$$

This formula produces column one in $\boldsymbol{L}$. Taking $j = 2$, we have

$$\ell_{i2} = \frac{a_{i2} - \ell_{i1} d_1 \ell_{21}}{d_2} \quad (3 \le i \le n)$$

This formula produces column two in $\boldsymbol{L}$.

LDL^T Factorization

The formal algorithm for the LDL^T factorization is as follows:

```
**integer** $i, j, n, \nu$;   **real array** $(a_{ij})_{1:n\times 1:n}, (\ell_{ij})_{1:n\times 1:n}, (d_i)_{1:n}$
**for** $j = 1$ **to** $n$
    $\ell_{jj} = 1$
    $d_j = a_{jj} - \sum_{\nu=1}^{j-1} d_\nu \ell_{j\nu}^2$
    **for** $i = j+1$ **to** $n$
        $\ell_{ji} = 0$
        $\ell_{ij} = \frac{a_{ij} - \sum_{\nu=1}^{j-1} \ell_{i\nu} d_\nu \ell_{j\nu}}{d_j}$
    **end for**
**end for**
```

LDL^T Factorization

Example

Determine the LDL^T factorization of the matrix: $A = \begin{bmatrix} 4 & 3 & 2 & 1 \\ 3 & 3 & 2 & 1 \\ 2 & 2 & 2 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$.

First, we determine the LU factorization:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{3}{4} & 1 & 0 & 0 \\ \frac{1}{2} & \frac{2}{3} & 1 & 0 \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} 4 & 3 & 2 & 1 \\ 0 & \frac{3}{4} & \frac{1}{2} & \frac{1}{4} \\ 0 & 0 & \frac{2}{3} & \frac{1}{3} \\ 0 & 0 & 0 & \frac{1}{2} \end{bmatrix} = LU$$

Then extract the diagonal elements from U and place them into a diagonal matrix D, writing

$$U = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & \frac{3}{4} & 0 & 0 \\ 0 & 0 & \frac{2}{3} & 0 \\ 0 & 0 & 0 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 1 & \frac{3}{4} & \frac{1}{2} & \frac{1}{4} \\ 0 & 1 & \frac{2}{3} & \frac{1}{3} \\ 0 & 0 & 1 & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} = DL^T$$

Clearly, we have $A = LDL^T$.

Extra: Permutation matrices

Multiplying by a permutation matrix will permute the order of the rows or columns of a matrix. We obtain permutation matrices by re-ordering the rows (or columns) of the identity matrix I. As an example, consider

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}.$$

Multiplying on the right by P a general matrix $A \in \mathcal{M}_{3\times n}$ will rearrange the order of the rows in A

$$PA = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ a_{21} & \cdots & a_{2n} \\ a_{31} & \cdots & a_{3n} \end{bmatrix} = \begin{bmatrix} a_{21} & \cdots & a_{2n} \\ a_{31} & \cdots & a_{3n} \\ a_{11} & \cdots & a_{1n} \end{bmatrix}$$

Extra: Permutation matrices

Every permutation matrix is nonsingular, since the rows still form a basis of $\mathbf{R}^n$.

When Gaussian elimination with row pivoting is performed on a matrix $\boldsymbol{A}$, the result is expressible as $\boldsymbol{PA} = \boldsymbol{LU}$, where $\boldsymbol{L}$ is lower triangular and $\boldsymbol{U}$ is upper triangular. The matrix $\boldsymbol{PA}$ is $\boldsymbol{A}$ with its rows rearranged. If we have the $\boldsymbol{LU}$-factorization of $\boldsymbol{PA}$, how do we solve the system

$$\boldsymbol{Ax} = \boldsymbol{b}?$$

First write it as $\boldsymbol{PAx} = \boldsymbol{Pb}$, then $\boldsymbol{LUx} = \boldsymbol{Pb}$. Let $\boldsymbol{y} = \boldsymbol{Ux}$, so that our problem is now $\boldsymbol{Ly} = \boldsymbol{Pb}$. That equation is easily solved. Then the equation $\boldsymbol{Ux} = \boldsymbol{y}$ is eeasily solved for $\boldsymbol{x}$.

Matlab and Maple produce factorizations of the form

$$\boldsymbol{PA} = \boldsymbol{LU}$$

upon command.

Extra: Modifying the LU factorization

Example

The Matlab function $lu(A)$ gives

$$lu(A) = [L, U, P].$$

For

$$A = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 2 & 3 \\ -1 & -3 & 0 \end{bmatrix},$$

$$\begin{bmatrix} 1 & 0 & 0 \\ -.5 & 1 & 0 \\ .5 & -.5 & 1 \end{bmatrix} \begin{bmatrix} 2 & 2 & 3 \\ 0 & -2 & 1.5 \\ 0 & 0 & .25 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 2 & 3 \\ -1 & -3 & 0 \end{bmatrix}$$

Extra: Scaling

If the coefficient matrix A in $Ax = b$ vary greatly in size, it is likely that propagation of rounding errors will be significant. Usually this is avoided by scaling the matrix A so that the maximum element is each row is of approximately equal magnitude, and similarly for the columns.

If $B = D_1 A D_2$ is the result of row and columns scaling in A, with D_1, D_2 diagonal matrices with entries the scaling constants, then solving $Ax = b \Leftrightarrow D_1 A D_2 (D_2^{-1} x) = D_1 b$ for x is equivalent to solving

$$Bz = D_1 b \qquad x = D_2 z.$$

For row scaling: seeking the coefficients such that

$$\max_{1 \le j \le n} |b_{ij}| \approx 1, i = 1, \dots, n,$$

the most straightforward way is

$$b_{ij} = \frac{a_{ij}}{s_i},\ j = 1, \dots n, \qquad \text{where } s_i = \max_{1 \le j \le n} |a_{ij}|, i = 1, \dots, n.$$

Extra: Variants of Gaussian Elimination - Compact Methods

If A is nonsingular and pivoting is not needed, then $A = LU$ is given by the Gaussian elimination. In order to reduce the error, we need to rewrite the Gaussian elimination in a more compact form without going through the elimination process: with

$$L = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ m_{21} & 1 & 0 & \dots & 0 \\ m_{31} & m_{32} & 1 & & \vdots \\ \vdots & & & \ddots & \\ m_{n1} & m_{n2} & \dots & m_{n,n\text{-}1} & 1 \end{pmatrix}, U = \begin{pmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ 0 & u_{22} & u_{23} & \dots & u_{2n} \\ \vdots & 0 & \ddots & & \vdots \\ & \vdots & 0 & u_{n-1,n-1} & u_{n-1,n} \\ 0 & 0 & \dots & 0 & u_{n,n} \end{pmatrix}$$

this means multiplying LU and match the result with A.

If L, U are only upper and lower triangular, there is nonuniqueness in the choice of L, U:

$$A = L_1U_1 = L_2U_2 \quad \Rightarrow \quad L_2^{-1}L_1 = U_2U_1^{-1} = D$$

with D a diagonal matrix, tied directly to the choice of the diagonal elements of either L or U. Once they have been chosen, D is uniquely determined.

Extra: Variants of Gaussian Elimination - Compact Methods

Step 1a.

$$u_{1j} = a_{1j}, \quad j = 1, \ldots, n.$$

Step 1.b. Solve for the 1st column of L: multiply rows $2:n$ of L with column 1 of U, then solve

$$m_{i1}u_{11} = a_{i1}, \quad i = 2, n$$

for the coefficients m_{i1}.

Step 2a. Solve for the 2nd row of U. Begin by multiplying row 2 of L with columns $2:n$ of U

$$m_{21}u_{1j} + u_{2j} = a_{2j}, \quad j = 2, \ldots, n$$

and solve for u_{2j} for $j = 2, \ldots, n$.

Step 2.b. Solve for the 2nd column of L: multiply rows $3:n$ of L with column 2 of U, then solve

$$m_{i1}u_{12} + m_{i2}u_{22} = a_{i2}, \quad i = 3, n$$

for the coefficients m_{i2} for $i = 3, n$.

Extra: Variants of Gaussian Elimination - Compact Methods

- Doolitle's method: diagonal elements of L are all 1 (gives the same decomposition $A = LU$ as in Gaussian elimination)
- Crout's method: all diagonal elements of U are all 1

With these L, U we solve $Ax = b$ by solving the systems

$$Lz = b \quad \text{and} \quad Ux = z$$

first being a lower triangular system (forward substitution), second upper triangular (back substitution).

The Cholesky factorization

Any symmetric matrix that has an $\boldsymbol{LU}$ factorization in which $\boldsymbol{L}$ is unit lower triangular, has an $\boldsymbol{LDL^T}$ factorization.

Definition

The matrix A is positive definite if

$$(Ax, x) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_i x_j > 0, \quad \forall x \in \mathbb{R}^n, x \neq 0.$$

The Cholesky factorization $\boldsymbol{A = LL^T}$ is a simple consequence of it for the case in which $\boldsymbol{A}$ is symmetric and positive definite.

Suppose that the factorization $\boldsymbol{A = LU}$ the matrix $\boldsymbol{L}$ is lower triangular and the matrix $\boldsymbol{U}$ is upper triangular.

- When $\boldsymbol{L}$ is unit lower triangular, it is called the **Doolittle factorization**.
- When $\boldsymbol{U}$ is unit upper triangular, it is called the **Crout factorization**.
- In the case $\boldsymbol{A}$ is symmetric positive definite and $\boldsymbol{U = L^T}$, it is called the **Cholesky factorization**.

The Cholesky factorization

André Louis Cholesky proved

Theorem (Cholesky theorem on LL^T factorization)

If A is real, **symmetric** *and* **positive definite**, *there is a (***unique***) lower triangular matrix*

$$L = \begin{pmatrix} \ell_{11} & 0 & 0 & \dots & 0 \\ \ell_{21} & \ell_{21} & 0 & \dots & 0 \\ \ell_{31} & \ell_{32} & \ell_{33} & & \vdots \\ \vdots & & & \ddots & \\ \ell_{n1} & \ell_{n2} & \dots & \ell_{n,n-1} & \ell_{nn} \end{pmatrix}$$

with diagonal elements $\ell_{ii} > 0$ such that

$$A = LL^T.$$

The Cholesky factorization

Multiply LL^T and match to A:

Step 1. Row 1 multiplied with column 1:

$$\ell_{11}{}^2 = a_{11} > 0.$$

Step 2. Row 1 of L multiplied with columns $2 : n$ of L^T:

$$\ell_{11}\ell_{i1} = a_{i1}, \quad i = 2, \ldots, n$$

and solve for all ℓ_{i1}.

Continuing for $i = 2, \ldots, n$ yields

$$\ell_{ij} = \frac{a_{ij} - \sum_{k=1}^{j-1} \ell_{ik}\ell_{jk}}{\ell_{jj}}, \quad j = 1, \ldots, i-1$$

$$\ell_{ii} = \left(a_{ii} - \sum_{k=1}^{i-1} \ell_{ik}^2 \right)^{\frac{1}{2}}.$$

Due to the symmetry of A, the number of arithmetic operations is $\approx \frac{1}{3}n^3$, half of what is needed with a general matrix A.

The Cholesky factorization

The algorithm for the **Cholesky factorization** will be then as follows:

```
input $n, (a_{ij})$
for $k = 1$ to $n$ do
    $\ell_{kk} = \left( a_{kk} - \sum_{s=1}^{k-1} \ell_{ks}^2 \right)^{\frac{1}{2}}$
    for $i = k+1$ to $n$ do
        $\ell_{ik} \leftarrow \frac{a_{ik} - \sum_{s=1}^{k-1} \ell_{is}\ell_{ks}}{\ell_{kk}}$
    end do
end do
output $(\ell_{ij})$
```

The Cholesky factorization

The Cholesky theorem guarantees that $\ell_{ii} > 0$. Observe that the equation

$$\ell_{ii} = \left(a_{ii} - \sum_{k=1}^{i-1} \ell_{ik}^2 \right)^{\frac{1}{2}}$$

gives the following bound

$$a_{ii} = \sum_{k=1}^{i} \ell_{ik}^2 \geq \ell_{ij}^2 \qquad (j \leq i)$$

from which we conclude that

$$|\ell_{ij}| \leq \sqrt{a_{ii}} \qquad (1 \leq j \leq i)$$

Hence, any element of $\boldsymbol{L}$ is bounded by the square of a corresponding diagonal element in $\boldsymbol{A}$. This implies that the elements of $\boldsymbol{L}$ do not become large relative to $\boldsymbol{A}$ even without any pivoting. In the Cholesky algorithm, (and the Doolittle algorithms), the dot products of vectors should be computed in double precision in order to avoid a buildup of roundoff errors.

The Cholesky method

Example For the Hilbert matrix of order 3:

$$A = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix},$$

the Cholesky decomposition is (check `chol(hilb(3))`)

$$A = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{\sqrt{12}} & 0 \\ \frac{1}{3} & \frac{1}{\sqrt{12}} & \frac{1}{\sqrt{180}} \end{bmatrix} \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ 0 & \frac{1}{\sqrt{12}} & \frac{1}{\sqrt{12}} \\ 0 & 0 & \frac{1}{\sqrt{180}} \end{bmatrix}.$$

Remark The advantage of using compact methods is: reduced number of rounding errors from $O(n^3)$ to $O(n^2)$.
Both the elements ℓ_{ij} and u_{ij} involve inner products $\sum_{k=1}^{m} \alpha_k \beta_k$ that can be accumulated in extended (double) precision, so instead of $2m$ rounding errors we have only one by rounding at the very end.

Summary

(1) If $\boldsymbol{A} = (a_{ij})$ is an $n \times n$ matrix such that the forward elimination phase of the naive gaussian algorithm can be applied to $\boldsymbol{A}$ without encountering any zero divisors, then the resulting matrix can be denoted by $\tilde{\boldsymbol{A}} = (\tilde{a}_{ij})$, where

$$\boldsymbol{L} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ \tilde{a}_{21} & 1 & 0 & \dots & 0 \\ \tilde{a}_{31} & \tilde{a}_{32} & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \tilde{a}_{n1} & \tilde{a}_{n2} & \dots & \tilde{a}_{n,n-1} & 1 \end{bmatrix}.$$

and

$$\boldsymbol{U} = \begin{bmatrix} \tilde{a}_{11} & \tilde{a}_{12} & \tilde{a}_{13} & \dots & \tilde{a}_{1n} \\ 0 & \tilde{a}_{22} & \tilde{a}_{23} & \dots & \tilde{a}_{2n} \\ 0 & 0 & \tilde{a}_{33} & \dots & \tilde{a}_{3n} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & \tilde{a}_{n,n} \end{bmatrix}.$$

This is the $\boldsymbol{LU}$ **factorization** of $\boldsymbol{A}$, so that $\boldsymbol{A} = \boldsymbol{LU}$, where $\boldsymbol{L}$ is lower unit triangular and $\boldsymbol{U}$ is upper triangular. When we carry out the Gaussian forward elimination process on $\boldsymbol{A}$, the result is upper triangular matrix $\boldsymbol{U}$. The matrix $\boldsymbol{L}$ is the unit lower triangular matrix whose entries are negatives of the multipliers in the locations of the elements they zero out.

Summary

(2) We can also give a formal description as follows. The matrix $\boldsymbol{U}$ can be obtained by applying a succession of matrices $\boldsymbol{M_k}$ to the left of $\boldsymbol{A}$. The kth step of Gaussian elimination corresponds to a unit lower triangular matrix $\boldsymbol{M_k}$, which is the product of $n-k$ elementary matrices

$$\boldsymbol{M_k = M_{nk}M_{n-1,k}\cdots M_{k+1,k}}$$

where each **elementary matrix** $\boldsymbol{M_{ik}}$ is unit lower triangular. If $\boldsymbol{M_{kp}A}$ is obtained by subtracting λ times row p from row q in matrix $\boldsymbol{A}$ with $p<q$, then the elements of $\boldsymbol{M_{qp}}= (m_{ij})$ are

$$m_{ij} = \begin{cases} 1 & \text{if } i=j \\ -\lambda & \text{if } i=q \text{ and j=p} \\ 0 & \text{in all other cases} \end{cases}$$

Summary

The entire Gaussian elimination process is summarized by writing

$$M_{n-1} \cdots M_2 M_1 A = U$$

Since each M_k is invertible, we have

$$A = M_1^{-1} M_2^{-1} \cdots M_n^{-1} U$$

Each M_k is a unit lower triangular matrix and the same is true for each inverse M_k^{-1}, as well as their products. hence, the matrix

$$L = M_1^{-1} M_2^{-1} \cdots M_n^{-1}$$

is unit lower triangular.

Summary

(3) For symmetric matrices, we have the $\boldsymbol{LDL^T}$-factorization, and for symmetric positive definite matrices, we have the $\boldsymbol{LL^T}$, which is also known as Cholesky factorization.

Summary

(4) If the LU factorization of $\boldsymbol{A}$ is available, we can solve the system

$$\boldsymbol{Ax} = \boldsymbol{b}$$

by solving two triangular systems

$$\begin{cases} \boldsymbol{Ly} = \boldsymbol{b} & \text{for } \boldsymbol{y} \\ \boldsymbol{Ux} = \boldsymbol{y} & \text{for } \boldsymbol{x} \end{cases}$$

This is useful for problems that involve the same coefficient matrix $\boldsymbol{A}$ and many different RH vectors $\boldsymbol{b}$. For example, let $\boldsymbol{B}$ be an $n \times m$ matrix of the form

$$\boldsymbol{B} = [\boldsymbol{b}^{(1)}, \boldsymbol{b}^{(2)}, \ldots, \boldsymbol{b}^{(m)}]$$

where each column corresponds to a RHS of the m linear systems

$$\boldsymbol{Ax}^{(j)} = \boldsymbol{b}^{(j)} \quad (1 \le j \le m)$$

Thus, we can write

$$\boldsymbol{A}[\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(m)}] = [\boldsymbol{b}^{(1)}, \boldsymbol{b}^{(2)}, \ldots, \boldsymbol{b}^{(m)}]$$

or

$$\boldsymbol{AX} = \boldsymbol{B}$$

Summary

A special case of this is to compute the inverse of an $n \times n$ invertible matrix $\boldsymbol{A}$. We write

$$\boldsymbol{AX} = \boldsymbol{I}$$

where $\boldsymbol{I}$ is the identity matrix. If $\boldsymbol{x}^{(j)}$ denotes the jth column of $\boldsymbol{X}$ and $\boldsymbol{I}^{(j)}$ denotes the jth column of $\boldsymbol{I}$, this can be written as

$$\boldsymbol{A}[\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(n)}] = [\boldsymbol{I}^{(1)}, \boldsymbol{I}^{(2)}, \dots, \boldsymbol{I}^{(n)}]$$

or as n linear systems of equations of the form

$$\boldsymbol{Ax}^{(j)} = \boldsymbol{I}^{(j)} \quad (1 \le j \le n)$$

We can use the $\boldsymbol{LU}$ factorization to solve these n systems efficiently obtaining

$$\boldsymbol{A}^{-1} = [\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(n)}]$$

Summary

(5) When Gaussian elimination with row-pivoting is performed on a matrix $\boldsymbol{A}$, the result is expressible as

$$\boldsymbol{PA} = \boldsymbol{LU}$$

where $\boldsymbol{P}$ is a **permutation matrix**, $\boldsymbol{L}$ is unit lower triangular, and $\boldsymbol{U}$ is upper triangular. Here the matrix $\boldsymbol{PA}$ is $\boldsymbol{A}$ with its rows interchanged. We can solve the system $\boldsymbol{Ax} = \boldsymbol{b}$ by solving

$$\begin{cases} \boldsymbol{Ly} = \boldsymbol{Pb} & \text{for } \boldsymbol{y} \\ \boldsymbol{Ux} = \boldsymbol{y} & \text{for } \boldsymbol{x} \end{cases}$$

Extra: Tridiagonal systems

$$A = \begin{bmatrix} a_1 & c_1 & 0 & 0 & \dots & 0 \\ b_2 & a_2 & c_2 & 0 & & \\ 0 & b_3 & a_3 & c_3 & & \vdots \\ \vdots & & \ddots & & & \\ & & & b_{n-1} & a_{n-1} & c_{n-1} \\ 0 & \dots & & 0 & b_n & a_n \end{bmatrix}$$

Occur in many applications from ODE and PDE.

Remark: the inverse of a tridiagonal matrix is usually a dense matrix

$$A = \begin{bmatrix} -1 & 1 & 0 & 0 & \dots & 0 \\ 1 & -2 & 1 & 0 & & \\ 0 & 1 & -2 & 1 & & \vdots \\ \vdots & & \ddots & & & \\ & & & 1 & -2 & 1 \\ 0 & \dots & & 0 & 1 & -\frac{n-1}{n} \end{bmatrix}, \qquad (A^{-1})_{ij} = \max\{i,j\}$$

Extra: Tridiagonal systems

Factoring tridiagonal matrix $A = LU$, L and U have very simple forms:

$$L = \begin{bmatrix} \alpha_1 & 0 & 0 & 0 & \dots & 0 \\ b_2 & \alpha_2 & 0 & 0 & & \\ 0 & b_3 & \alpha_3 & 0 & & \vdots \\ \vdots & & \ddots & & & \\ & & & b_{n-1} & \alpha_{n-1} & 0 \\ 0 & \dots & & 0 & b_n & \alpha_n \end{bmatrix}, U = \begin{bmatrix} 1 & \gamma_1 & 0 & 0 & \dots & 0 \\ 0 & 1 & \gamma_2 & 0 & & \\ 0 & 0 & 1 & \gamma_3 & & \vdots \\ \vdots & & \ddots & & & \\ & & & 0 & 1 & \gamma_{n-1} \\ 0 & \dots & & 0 & 0 & 1 \end{bmatrix}$$

By multiplication we get:

$$\begin{array}{ll} a_1 = \alpha_1, \alpha_1\gamma_1 = c_1 & \\ a_i = \alpha_i + b_i\gamma_{i-1}, & i = 2, \dots, n \\ \alpha_i\gamma_i = c_i, & i = 2, 3, \dots, n-1 \end{array} \quad \Rightarrow \quad \begin{array}{l} \alpha_1 = a_1, \gamma_1 = \frac{c_1}{\alpha_1} \\ \alpha_i = a_i - b_i\gamma_{i-1}, \gamma_i = \frac{c_i}{\alpha_i}, i = 2, \dots, n-1 \\ \alpha_n = a_n - b_n\gamma_{n-1} \end{array}$$

To factor $A = LU$ it takes: $\left\{ \begin{array}{ll} + : & n-1 \\ * : & n-1 \\ / : & n-1 \end{array} \right. \Rightarrow 3n-3$ operations.

Extra: Tridiagonal systems

Solving $Ax = f$ as $LUx = f$, i.e.,

$$Lz = f, \qquad Ux = z$$

requires:

$$\begin{aligned} z_1 &= \frac{f_1}{\alpha_1}, \quad z_i = \frac{f_i - b_i z_{i-1}}{\alpha_i}, \qquad i = 2, \ldots, n \\ x_n &= z_n, \quad x_i = z_i - \gamma_i x_{i+1}, \qquad i = n-1, \ldots, 1 \end{aligned}$$

hence it takes $\left\{ \begin{array}{ll} + : & 2n-2 \\ * : & 2n-2 \\ / : & n \end{array} \right. \Rightarrow 5n - 4$ operations to solve $Ax = f$

the first time and additional $3n - 2$ for each different RHS f.

Remark: Even if A^{-1} is given, computing

$$x = A^{-1} f \Leftrightarrow x_i = \sum_{j=1}^{n} (A^{-1})_{ij} f_j, i = 1, \ldots, n$$

means $2(n-1)^2$ operations.

Extra: Tridiagonal systems

Theorem

Assume $\{a_i, b_i, c_i\}$ *satisfy*

$$\begin{aligned} &|a_1| > |c_1| > 0 \\ &|a_i| \geq |b_i| + |c_i|, \quad b_i, c_i \neq 0 \quad i = 2, \ldots, n-1 \\ &|a_n| > |b_n| > 0 \end{aligned}$$

then A *is nonsingular,*

$$\begin{aligned} &|\gamma_i| < 1, \quad i = 1, \ldots, n-1 \\ &|a_i| - |b_i| < |\alpha_i| < |a_i| + |b_i|, \quad i = 2, \ldots, n \end{aligned} \tag{8.11}$$

Remark that (8.11) implies $|\alpha_i| > |c_i|$ and justifies the decomposition of A.

Error in solving linear systems

Example. Find the **backward error** and the **forward error** for the approximate solution $x_c = [1; 1]$ of the system

$$\begin{bmatrix} 1 & 1 \\ 3 & -4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}.$$

The correct solution is $x = [2; 1]$. In the infinity norm, the **backward error** is

$$\|b - Ax_c\|_\infty = \left\| \begin{bmatrix} 3 \\ 2 \end{bmatrix} - \begin{bmatrix} 1 & 1 \\ 3 & -4 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\|_\infty = \left\| \begin{bmatrix} 1 \\ 3 \end{bmatrix} \right\|_\infty = 3,$$

and the **forward error** is

$$\|x - x_c\|_\infty = \left\| \begin{bmatrix} 2 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\|_\infty = \left\| \begin{bmatrix} 2 \\ 1 \end{bmatrix} \right\|_\infty = 1.$$

$$cond(A) = 3.5776$$

In other cases, the backward and forward errors can be of different orders of magnitude.

Error in solving linear systems

Example. Find the **backward error** and the **forward error** for the approximate solution $x_c = [-1; 3.0001]$ of the system

$$\begin{bmatrix} 1 & 1 \\ 1.0001 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 2.0001 \end{bmatrix}.$$

* First, using Gaussian elimination, the solution is $[x_1, x_2] = [1; 1]$.

** The **backward error** in the infinity norm is

$$\|b - Ax_c\|_\infty = \left\| \begin{bmatrix} 2 \\ 2.0001 \end{bmatrix} - \begin{bmatrix} 1 & 1.0001 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} -1 \\ 3.0001 \end{bmatrix} \right\|_\infty = \left\| \begin{bmatrix} -0.0001 \\ 0.0001 \end{bmatrix} \right\|_\infty = \mathbf{0}.0001.$$

*** The **forward error** is the infinity norm of the difference

$$x - x_c = \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \begin{bmatrix} -1 \\ 3.0001 \end{bmatrix} = \begin{bmatrix} 2 \\ -2.0001 \end{bmatrix}$$

which is $\mathbf{2}.0001$.

$$cond(A) = 4.0002e + 04$$

Error in solving linear systems

On a computer, Gaussian elimination involves rounding errors through its arithmetic operations, and these errors lead to error in the computed solution of the linear system being solved.

- We now will briefly look at a method for predicting and correcting the error in the computed solution.
- Then we examine the sensitivity of the solution to small changes such as rounding errors.

Examples

1. The linear system
$$\begin{cases} 7x & +10y & = 1 \\ 5x & +7y & = .7 \end{cases}$$ has the solution $x = 0, \quad y = .1$
while the perturbed system
$$\begin{cases} 7\hat{x} & +10\hat{y} & = 1.01 \\ 5\hat{x} & +7\hat{y} & = .69 \end{cases}$$ has the solution $\hat{x} = -.17, \quad \hat{y} = .22$
A relatively small change in the RHS has led to a relatively large change in the solution.

2. The Hilbert matrix and its approximation
$$H_3 = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix}, \quad \hat{H}_3 = \begin{bmatrix} 1.000 & .5000 & .3333 \\ .5000 & .3333 & .2500 \\ .3333 & .2500 & .2000 \end{bmatrix}$$
have inverse
$$H_3^{-1} = \begin{bmatrix} 9 & -36 & 30 \\ -36 & 192 & -180 \\ 30 & -180 & 180 \end{bmatrix}, \quad \hat{H}_3^{-1} = \begin{bmatrix} 9.062 & -36.32 & 30.30 \\ -36.32 & 193.7 & -181.6 \\ 30.30 & -181.6 & 181.5 \end{bmatrix}.$$
(a change in H_3 in the 5^{th} decimal place (by rounding the fractions to four decimal digits) yields with a change in H_3^{-1} in the 3^{rd} decimal place.)

$$cond(A) = 222.9955, cond(hilb(4)) = 524.0568, cond(hilb(4)) = 1.5514e+04.$$

The sensitivity equation, condition number

The sensitivity of the solution x of the linear system $Ax = b$ to changes in the RHS b:

$$Ax = b, \quad A\tilde{x} = b + r \qquad \Rightarrow \qquad Ae = r, \text{ where } e = \tilde{x} - x.$$

The relative error satisfies:

$$\frac{\|\tilde{x}-x\|}{\|x\|} \equiv \frac{\|e\|}{\|x\|} = \begin{cases} \frac{\|A^{-1}r\|}{\|x\|} \leq \frac{\|A^{-1}\|\|r\|}{\|x\|} = \|A^{-1}\|\|A\|\frac{\|r\|}{\|A\|\|x\|} \leq \underbrace{\|A^{-1}\|\|A\|}_{\text{condition number}}\frac{\|r\|}{\|b\|}, \\ \geq \|e\|\frac{1}{\|A^{-1}\|\|b\|} \geq \frac{\|r\|}{\|A\|}\frac{1}{\|A^{-1}\|\|b\|} = \frac{1}{\text{cond}(A)}\frac{\|r\|}{\|b\|}, \end{cases}$$

i.e.,

$$\frac{1}{\text{cond}(A)}\frac{\|r\|}{\|b\|} \leq \frac{\|e\|}{\|x\|} \leq \text{cond}(A)\frac{\|r\|}{\|b\|}.$$

Moreover, $\forall A$ nonsingular, $\exists b$ and r for which either of the inequalities can be made equality.

Condition number

cond(A) can vary, but is always greater than 1:

$$1 = \|I\| = \|AA^{-1}\| \leq \|A\|\|A^{-1}\| = \text{cond}(A).$$

- If cond$(A) \approx 1$, then

$$\frac{1}{\text{cond}(A)}\frac{\|r\|}{\|b\|} \leq \frac{\|e\|}{\|x\|} \leq \text{cond}(A)\frac{\|r\|}{\|b\|}$$

  implies:

  small "relative" changes in RHS $b \Rightarrow$ small "relative" changes in x.

- If cond(A) is very large, then $(\exists)$ b, r for which

  $\frac{\|r\|}{\|b\|}$ is small but $\frac{\|\tilde{x}-x\|}{\|x\|}$ is large.

Lower bound on condition number

Since the $\text{cond}(A)$ depends on the $\|\cdot\|$, sometimes another definition of condition number is used:

$$\text{cond}(A)_* := \frac{\max_{\lambda\in\sigma(A)}|\lambda|}{\min_{\lambda\in\sigma(A)}|\lambda|} = \rho(A)\rho(A^{-1}) \leq \|A\|\|A^{-1}\| = \text{cond}(A)$$

<u>Remark</u>. For the system $\begin{cases} 7x & +10y & =1 \\ 5x & +7y & =.7 \end{cases}$ we have

$$\text{cond}(A)_1 = \text{cond}(A)_\infty = \|A\|_\infty \left\| \begin{pmatrix} -7 & 10 \\ 5 & -7 \end{pmatrix} \right\|_\infty = 17\cdot 17 = 289$$

$$\text{cond}(A)_2 \approx 223 \qquad \text{cond}(A)_* \approx 198$$

which explains the large relative changes in x compared with the relative changes in b.

Theorem(Gastinel)

Let A be a nonsingular matrix. Then

$$\frac{1}{\text{cond}(A)} = \min\left\{\frac{\|A-B\|}{\|A\|} : B \text{ a singular matrix}\right\}$$

or equivalently

$$\frac{1}{\|A\|} = \min\{\|P\| : A+P \text{ is singular}\}.$$

The Theorem states that $\frac{1}{\text{cond}(A)}$ measures how close in a 'relative' error sense A is to a singular matrix B, the ultimate ill-conditioning:

- for $\lambda = 0$, v the corresponding eigenvector $Bv = 0 \cdot v \Rightarrow$ even for zero perturbation ($r = 0$) of the RHS b, $(\exists)$ nonzero perturbations of the solution ($Be = 0$): $v = \tilde{x} - x \neq 0$
- $(\exists) b$ such that $Bx = b$ is not solvable.

Perturbation theorem

Theorem

Let $Ax = b$ denote a nonsingular linear system, and $\delta A, \delta b$ be perturbations of A and b such that

$$\|\delta A\| < \frac{1}{\|A^{-1}\|}. \qquad \left(\Leftrightarrow \frac{\|\delta A\|}{\|A\|} < \frac{1}{\mathit{cond}(A)}\right)$$

Then $A + \delta A$ is nonsingular, and the perturbation $\delta x = \tilde{x} - x$ defined by

$$(A + \delta A)(x + \delta x) = b + \delta b$$

satisfies:

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\mathit{cond}(A)}{1 - \mathit{cond}(A)\frac{\|\delta A\|}{\|A\|}} \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|}\right).$$

- if cond(A) is relatively small, say $1 \leq$cond$(A) \leq 10$, then small relative changes in the data (A and b) will lead to small relative changes in the solution
- if cond(A) is large, then there are choices of b and δb for which the relative change in x will be very large.

The Hilbert matrix

$$H_n = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{n} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \cdots & \frac{1}{n+1} \\ \frac{1}{3} & \frac{1}{4} & \cdots & & \\ \vdots & & \ddots & & \vdots \\ \frac{1}{n} & \frac{1}{n+1} & & \cdots & \frac{1}{2n-1} \end{bmatrix}$$

It is a very ill-conditioned matrix for larger values of n:

n	$\text{cond}_2(A)$
3	5.24 E + 2
6	1.50 E + 7
9	4.93 E + 11

Backward error analysis

Using backward error analysis, Wilkinson showed that the computed solution $\hat{x}$ to $Ax = b$ is the exact solution to a perturbed system

$$(A + \delta A)\hat{x} = b$$

with

$$\frac{\|\delta A\|_\infty}{\|A\|_\infty} \leq 1.01(n^3 + 3n^2)\rho u$$

where $\rho = \dfrac{1}{\|A\|_\infty \displaystyle\max_{1\leq i,j,k\leq n} |a_{ij}^{(k)}|}$, u the unit round of the computer arithmetic.

In practice ρ rarely gets very large, so by the Perturbation Theorem: for matrices A with a condition number that is not too large, the use of Gaussian elimination is generally safe.

In practice, a better empirical bound is $\frac{\|\delta A\|_\infty}{\|A\|_\infty} \leq nu$.

The residual correction method

When solving the original system $Ax = b$, if the matrices L, U have been saved, then we can solve $Ae = r$ at a relatively small computation cost, i.e., $2n^2$ compared to $\frac{n^3}{3}$ (n^2 to find e and same to compute $r = b - A\hat{x}$).

In the practical implementation of this procedure for computing the error e, there are two possible sources of difficulty.

1. The computation of the residual r in $r = b - A\hat{x}$ will involve loss-of-significance errors. Each component
$$r_i = b_i - \sum_{j=1}^{n} a_{ij}\hat{x}_j$$
will be very close to zero, and this occurs by *subtraction of the nearly equal quantities*. To avoid obtaining very inaccurate value for r_i, the calculation must be carried out in *higher precision arithmetic*.

2. Solving $Ae = r$ will involve same rounding errors that led to the error in the calculated value $\hat{x}$. Hence we will not obtain e, but only an approximation $\hat{e}$. How closely $\hat{e}$ approximates e depends on the sensitivity or conditioning of the matrix.

The residual correction method

Example.
$$\begin{aligned} 0.729x_1 &+0.81x_2 &+0.9x_3 &= 0.6867 \\ x_1 &+x_2 &+x_3 &= 0.8338 \\ 1.331x_1 &+1.21x_2 &+1.1x_3 &= 1.000 \end{aligned}$$

Using a *four-digit* decimal machine with rounding, the true solution is

$$x_1 = 0.2245, x_2 = 0.2814, x_3 = 0.3279$$

while the solution with Gaussian elimination without pivoting is

$$\hat{x}_1 = 0.2251, \hat{x}_2 = 0.2790, \hat{x}_3 = 0.3295$$

Calculating the residual using eight-digit floating point decimal arithmetic and rounding to 4 significant digits:

$$r = [0.00006210; 0.0002000; 0.0003519]$$

and then

$$\hat{e} = [-0.0004471; 0.002150; -0.001504].$$

The true error is

$$e = [-0.0007; 0.0024; -0.0016]$$

but $\hat{e}$ gives a good idea of the size of the error in e in the computed solution $\hat{x}$.

$$cond(A) = 927.1856$$

The residual correction method

A further use of this error estimation is to define an **iterative method** for improving the computed value x. Let $x^{(0)} = \hat{x}$, the initial computed value for x, generally obtained by Gaussian elimination and define

$$r^{(0)} = b - Ax^{(0)}$$

and as before

$$Ae^{(0)} = r^{(0)}, \quad e^{(0)} = x - x^{(0)}.$$

Solving by Gaussian elimination, we obtain an approximate value $\hat{e}^{(0)} \approx e^{(0)}$. Using it, we define an improved approximation

$$x^{(1)} = x^{(0)} + \hat{e}^{(0)}$$

Now we repeat the entire process, calculating

$$r^{(1)} = b - Ax^{(1)}$$

$$x^{(2)} = x^{(1)} + \hat{e}^{(1)}$$

where $\hat{e}^{(1)}$ is the approximating solution of

$$Ae^{(1)} = r^{(1)}, \quad e^{(1)} = x - x^{(1)}.$$

Continue this process until there is no further decrease in the size of $e^{(k)}, k \geq 0$.

The residual correction method

Given approximate solution: $\widehat{x}$

$(*)$ Compute residual: $\hat{r} = b - A\widehat{x}$

Perform a few extra calculations

Use $\hat{r}$ to improve $\widehat{x}$ to a better approximation

$\widehat{x} \Rightarrow$ better approximation to x

Test if the new $\widehat{x}$ is good enough (typically, if $\|\hat{r}\| <$Tolerance)

If not, go to $(*)$ and repeat.

The residual correction method

As an example (that we will analyze in the next section) consider the method known as first order Richardson, or FOR.
In FOR, we pick the number $\rho > 0$, rewrite $Ax = b$ as

$$\rho(x - x) = b - Ax,$$

then iterate:

```
Guess $x^{\mathrm{OLD}}$
Compute $r = b - Ax^{\mathrm{OLD}}$
$\rho(x^{\mathrm{NEW}} - x^{\mathrm{OLD}}) = r$
Test if $x^{\mathrm{NEW}}$ is acceptable.
If not, continue.
```

Mathematically, this is written:

$$\rho(x^{n+1} - x^{n}) = b - Ax^{n}. \tag{8.12}$$

The residual correction method

Example. Using a computer with four-digit floating point arithmetic with rounding, use the Gaussian elimination with pivoting to solve

$$\begin{array}{rrrl} x_1 & +0.5x_2 & +0.3333x_3 & =1 \\ 0.5x_1 & +0.3333x_2 & +0.25x_3 & =0 \\ 0.3333x_1 & +0.25x_2 & +0.2x_3 & =0 \end{array}$$

Then

$$x^{(0)} = [8.968; -35.77; 29.77]$$
$$r^{(0)} = [-0.005341; -0.004359; -0.0005344]$$
$$\hat{e}^{(0)} = [0.09216; -0.5442; 0.5239]$$
$$x^{(1)} = [9.060; -36.31; 30.29]$$
$$r^{(1)} = [-0.0006570; -0.0003770; -0.0001980]$$
$$\hat{e}^{(1)} = [0.001707; -0.01300; 0.01241]$$
$$x^{(2)} = [9.062; -36.32; 30.30]$$

The iterate $x^{(2)}$ is the correctly rounded solution of the system.

This illustrates the usefulness of the residual correction method.

1. **direct methods**: are good for dense systems of small to moderate size. This generally means the entire coefficient matrix for the linear system can be stored in the computers memory. Then Gaussian elimination can be applied to solve the linear system.

2. **iteration methods**: used with large sparse systems and large dense systems. Discretizations of boundary value problems for PDEs often lead to large sparse systems, and there is generally a pattern to the sparsity.
   - Traditional iteration - Jacobi, Gauss-Seidel, SOR, red/black iteration, line iteration
   - Multigrid iteration - Uses many levels of discretization
   - *Conjugate gradient* iteration and variants of it.

In this section, a completely different strategy for solving a nonsingular linear system

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{8.13}$$

is explored.

This alternative approach is often used on enormous problems that arise in solving partial differential equations numerically.

In that subject, systems having 100,000 unknowns and 100,000 equations arise routinely.

Extra: Poisson's Equation

Let $\Omega \subset \mathbb{R}^2, \Gamma = \partial\Omega$. The BVP for Poisson's equation with Dirichlet boundary conditions:

$$\begin{cases} \Delta u(x,y) = g(x,y), & (x,y) \in \Omega \\ u(x,y) = f(x,y), & (x,y) \in \Gamma \end{cases} \tag{8.14}$$

For $\Omega = \{(x,y) : 0 < x, y < 1\}$, $N \in \mathbb{N}, h := \frac{1}{N}$ we introduce the rectangular mesh on $\overline{\Omega} = \Omega \cup \Gamma$:

$$(x_j, y_k) = \left(\frac{j}{N}, \frac{k}{N}\right), \qquad j,k = 0,1,\ldots,N.$$

Recall:

$$\Delta u(x,y) = \frac{\partial^2 u(x,y)}{\partial x^2} + \frac{\partial^2 u(x,y)}{\partial y^2},$$

and that for $\phi \in C^4([t-h,t+h])$:

$$\phi''(t) = \frac{\phi(t+h) - 2\phi(t) + \phi(t-h)}{h^2} - \frac{h^2}{12}\phi^{(4)}(\xi), \quad \xi \in [t-h, t+h].$$

Extra: Poisson's Equation

On the mesh points inside Ω $((x_j, y_k) \in \Omega)$ we obtain

$$\frac{u(x_{j+1},y_k)-2u(x_j,y_k)+u(x_{j-1},y_k)}{h^2}+\frac{u(x_j,y_{k+1})-2u(x_j,y_k)+u(x_j,y_{k-1})}{h^2}$$
$$-\frac{h^2}{12}\left(\frac{\partial^4 u(\xi_j,y_k)}{\partial x^4}+\frac{\partial^4 u(x_j,\eta_k)}{\partial y^4}\right)=g(x_j,y_k), \tag{8.15}$$

where $\xi_j \in [x_{j-1}, x_{j+1}], \eta_k \in [y_{k-1}, y_{k+1}]$.
This leads to the a set of equations in the unknowns $u_h \approx u$:

$$\{u_h(x_j, y_k) : 1 < j, k < N-1\}$$

given by

$$\begin{cases} \frac{u_h(x_{j+1},y_k)-2u_h(x_j,y_k)+u_h(x_{j-1},y_k)}{h^2}+\frac{u_h(x_j,y_{k+1})-2u_h(x_j,y_k)+u_h(x_j,y_{k-1})}{h^2} \\ \qquad = g(x_j, y_k), \quad (x_j, y_k) \in \Omega \\ u_h(x_j, y_k) = f(x_j, y_k), \quad (x_j, y_k) \in \Gamma. \end{cases}$$

Extra: Poisson's Equation

We have a system of $(N+1)^2$ linear equations in $(N+1)^2$ unknowns

$$\{u_h(x_j,y_k) : 0 \le j,k \le N\}$$

i.e.,

$$\begin{cases} u_h(x_j,y_{k-1}) + u_h(x_{j-1},y_k) - 4u_h(x_j,y_k) \\ \quad +u_h(x_{j+1},y_k) + u_h(x_j,y_{k+1}) = h^2 g(x_j,y_k), (x_j,y_k), \quad (x_j,y_k) \in \Omega \\ u_h(x_j,y_k) = f(x_j,y_k), \quad (x_j,y_k) \in \Gamma \end{cases}$$

that has a unique solution. Indeed, the homogeneous linear system

$$\begin{cases} v_h(x_j,y_{k-1}) + v_h(x_{j-1},y_k) - 4v_h(x_j,y_k) \\ \quad +v_h(x_{j+1},y_k) + v_h(x_j,y_{k+1}) = 0, \quad (x_j,y_k) \in \Omega \\ u_h(x_j,y_k) = 0, \quad (x_j,y_k) \in \Gamma \end{cases} \tag{8.16}$$

has only the zero solution.

Extra: Poisson's Equation

Proof:

- Define
$$v_h(\overline{x}_j, \overline{y}_k) = \max_{\Omega} v_h(x_j, y_k) \equiv M$$
and assume that $M > 0$. From (8.16):
$$v(x_j, y_k) = \tfrac{1}{4}\Big(v_h(x_j, y_{k-1}) + v_h(x_{j-1}, y_k) + v_h(x_{j+1}, y_k) + v_h(x_j, y_{k+1})\Big)$$
we get that $v_h = M$ at all 4 points surrounding $(\overline{x}_j, \overline{y}_k)$.
At the boundary we get a contradiction $\Rightarrow M \leq 0$.
- Similarly
$$\min_{\Omega} v_h(x_j, y_k) \geq 0$$

and therefore $v_h(x_j, y_k) = 0$ in Ω. ■

If $u \in C^4(\overline{\Omega})$, $(\exists)$ c such that
$$\max_{\Omega} |u(x_j, y_k) - u_h(x_j, y_k)| \leq ch^2.$$

Vector and Matrix Norms

We present a brief overview of vector and matrix norms because they are useful in the discussion of error s and in the stopping criteria for iterative methods.
Norms can be defined on any vector space but we usually use $\mathbb{R}^n$ or $\mathbb{C}^n$.
A vector norm $\|\boldsymbol{x}\|$ can be thought of as the **length** or **magnitude** of a vector $\boldsymbol{x} \in \mathbb{R}^n$.
A **vector norm** is any mapping from $\mathbb{R}^n$ to $\mathbb{R}$ ($\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$) that obeys these three properties:

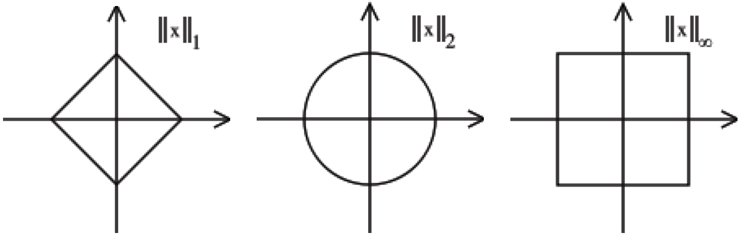
1. $\|\boldsymbol{x}\| > 0$ if $\boldsymbol{x} \neq \mathbf{0}$
2. $\|\alpha \boldsymbol{x}\| = |\alpha| \|\boldsymbol{x}\|$
3. $\|\boldsymbol{x} + \boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\|$ (**triangle inequality**)

for any vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ and scalars $\alpha \in \mathbb{R}$.

Vector and Matrix Norms

Examples of vector norms for the vector $\boldsymbol{x} = (x_1, x_2, \dots, x_n)^T \in \mathbb{R}^n$ are

- $\|\boldsymbol{x}\|_1 = \sum_{i=1}^{n} |x_i|$ **ℓ_1-vector norm**
- $\|\boldsymbol{x}\|_2 = \left(\sum_{i=1}^{n} |x_i^2|\right)^{\frac{1}{2}}$ **Euclidean / ℓ_2-vector norm**
- $\|\boldsymbol{x}\|_\infty = \max_{\leq i \leq n} |x_i|$ **ℓ_∞-vector norm**



Vector and Matrix Norms

For $n \times n$ matrices, we can also have **matrix norms**, subject to the same requirements

1. $\|\boldsymbol{A}\| > 0$ if $\boldsymbol{A} \neq \boldsymbol{0}$
2. $\|\alpha \boldsymbol{A}\| = |\alpha|\|\boldsymbol{A}\|$
3. $\|\boldsymbol{A} + \boldsymbol{B}\| \leq \|\boldsymbol{A}\| + \|\boldsymbol{B}\|$ (**triangular inequality**)

for matrices $\boldsymbol{A}, \boldsymbol{B}$ and scalars α.

We usually prefer matrix norms that are related to a vector norm.

For a vector norm $\|\cdot\|$, the **subordinate matrix norm** is defined by

$$\|\boldsymbol{A}\| \equiv \sup_{\|\boldsymbol{x}\|=1} \{\|\boldsymbol{A}\boldsymbol{x}\| : \boldsymbol{x} \in \mathbb{R}^n\}$$

Here $\boldsymbol{A}$ is an $n \times n$ matrix.

Vector and Matrix Norms

Additional properties that hold are

- $\|\boldsymbol{I}\| = 1$
- $\|\boldsymbol{A}\boldsymbol{x}\| = \|\boldsymbol{A}\|\|\boldsymbol{x}\|$
- $\|\boldsymbol{A}\boldsymbol{B}\| \leq \|\boldsymbol{A}\|\|\boldsymbol{B}\|$

Example of matrix norms, for an $n \times n$ matrix $\boldsymbol{A}$, are

- $\|\boldsymbol{A}\|_1 = \max\limits_{1\leq j\leq n} \sum\limits_{i=1}^{n} |a_{ij}|$ **ℓ_1-matrix norm**
- $\|\boldsymbol{A}\|_2 = \max\limits_{1\leq i} |\sigma_i| = \left[\rho(\boldsymbol{A}^T\boldsymbol{A})\right]^{\frac{1}{2}}$ **spectral / ℓ_2-matrix norm**
- $\|\boldsymbol{A}\|_\infty = \max\limits_{1\leq i\leq n} \sum\limits_{j=1}^{n} |a_{ij}|$ **ℓ_∞-matrix norm**

where σ_i are the **singular values** of $\boldsymbol{A}$, eigenvalues of $\boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{A}$. There are two meanings associated with the notation $\|\cdot\|_p$, one for vectors and another for matrices. The context will determine which one is intended.

Basic Iterative Methods

The iterative-method strategy produces a sequence of approximate solution vectors

$$\boldsymbol{x}^{(0)}, \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots$$

for system (8.13).

The numerical procedure is designed so that, in principle,

- **the sequence of vectors converges to the actual solution.**
- **The process can be stopped when sufficient precision has been attained.**

This stands in contrast to the Gaussian elimination algorithm, which has no provision for stopping midway and offering up an approximate solution.

Basic Iterative Methods

A general iterative algorithm for solving system (8.13) goes as follows.

- Select a **nonsingular matrix Q**, and
- having chosen an arbitrary starting vector $\boldsymbol{x}^{(0)}$,
- generate vectors $\boldsymbol{x}^{(0)}, \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots$ recursively from the equation

$$\boldsymbol{Q}\boldsymbol{x}^{(k)} = (\boldsymbol{Q} - \boldsymbol{A})\boldsymbol{x}^{(k-1)} + \boldsymbol{b} \tag{8.17}$$

  with integers $k \geq 1$.

To see this is sensible, suppose that the sequence $\boldsymbol{x}^{(k)}$ does converge, to a vector $\boldsymbol{x}^*$, say.
Then, by taking the limit as $k \to \infty$ in system (8.17), we get

$$\boldsymbol{Q}\boldsymbol{x}^* = (\boldsymbol{Q} - \boldsymbol{A})\boldsymbol{x}^* + \boldsymbol{b}$$

This leads to $\boldsymbol{A}\boldsymbol{x}^* = \boldsymbol{b}$. Thus, if the sequence converges, its limit is a solution to the original system (8.13).

Basic Iterative Methods

An outline of the pseudocode for carrying out the general iterative procedure (8.17) is

```
$\boldsymbol{x} \leftarrow \boldsymbol{x}^{(0)}$
**for** $k = 1$ **to** $k_{max}$ **do**
    $\boldsymbol{y} \leftarrow \boldsymbol{x}$
    $\boldsymbol{c} \leftarrow (\boldsymbol{Q} - \boldsymbol{A})\boldsymbol{x} + \boldsymbol{b}$
    solve $\boldsymbol{Q}\boldsymbol{x} = \boldsymbol{c}$
    **output** $k, \quad \boldsymbol{x}$
    **if** $\|\boldsymbol{x} - \boldsymbol{y}\| < \varepsilon$ **then**
        **output** "convergence"
        **stop**
    **end if**
**end for**
**output** "maximum iteration reached"
**end**
```

Basic Iterative Methods

In choosing the nonsingular matrix $\boldsymbol{Q}$, we are forced by the following considerations:

- System (8.17) should be easy to solve for $\boldsymbol{x}^{(k)}$, knowing the RHS.
- matrix $\boldsymbol{Q}$ should be chosen to ensure that the sequence $\boldsymbol{x}^{(k)}$ converges, no matter what initial vector is used.
  Ideally, this convergence will be rapid.

Basic Iterative Methods

For the analysis of the method described by system (8.17), we write

$$\boldsymbol{x}^{(k)} = \boldsymbol{Q}^{-\mathbf{1}}\left[(\boldsymbol{Q}-\boldsymbol{A})\boldsymbol{x}^{(k-1)} + \boldsymbol{b}\right]$$

or

$$\boldsymbol{x}^{(k)} = \mathcal{G}\boldsymbol{x}^{(k-1)} + \boldsymbol{k} \tag{8.18}$$

where the iteration matrix and the vector are

$$\mathcal{G} = \boldsymbol{I} - \boldsymbol{Q}^{-\mathbf{1}}\boldsymbol{A}, \qquad \boldsymbol{k} = \boldsymbol{Q}^{-\mathbf{1}}\boldsymbol{b}$$

Notice that in the pseudocode we do note compute $\boldsymbol{Q}^{-\mathbf{1}}$.
We are using $\boldsymbol{Q}^{-\mathbf{1}}$ to facilitate the analysis.

Basic Iterative Methods

Now let $\boldsymbol{x}$ the solution of system (8.13). Since $\boldsymbol{A}$ is nonsingular, $\boldsymbol{x}$ exists and is unique. We have from equation (8.17),

$$\begin{aligned}\boldsymbol{x}^{(k)} - \boldsymbol{x} &= (\boldsymbol{I} - \boldsymbol{Q}^{-1}\boldsymbol{A})\boldsymbol{x}^{(k-1)} - \boldsymbol{x} + \boldsymbol{Q}^{-1}\boldsymbol{b} \\ &= (\boldsymbol{I} - \boldsymbol{Q}^{-1}\boldsymbol{A})\boldsymbol{x}^{(k-1)} - (\boldsymbol{I} - \boldsymbol{Q}^{-1}\boldsymbol{A})\boldsymbol{x} \\ &= (\boldsymbol{I} - \boldsymbol{Q}^{-1}\boldsymbol{A})(\boldsymbol{x}^{(k-1)} - \boldsymbol{x})\end{aligned}$$

One can interpret $\boldsymbol{e}^k \equiv \boldsymbol{x}^{(k)} - \boldsymbol{x}$ as the current **error vector**. Thus we have

$$\boldsymbol{e}^k = (\boldsymbol{I} - \boldsymbol{Q}^{-1}\boldsymbol{A})\boldsymbol{e}^{k-1} \tag{8.19}$$

We want $\boldsymbol{e}^k$ to become *smaller* as k increases. Equation (8.19) shows that $\boldsymbol{e}^k$ will be *smaller* than $\boldsymbol{e}^{k-1}$ if $\boldsymbol{I} - \boldsymbol{Q}^{-1}\boldsymbol{A}$ is *small*, in some sense. In turn, that means that $\boldsymbol{Q}$ should be *close* to $\boldsymbol{A}$.
(Norms can be used to make *small* and *close* precise.)

Jacobi iteration

Rewrite the system (8.13)

$$Ax = b \qquad \Leftrightarrow \qquad \sum_{1\le i\le n} a_{ij}x_j = b_i, \quad 1 \le i \le n \tag{8.20}$$

as solving the ith equation for the ith unknown

$$x_i = \frac{1}{a_{ii}}\left(b_i - \sum_{j\ne i, j=1}^{n} a_{ij}x_j\right), \qquad i = 1, \dots, n \tag{8.21}$$

assuming that $a_{ii} \ne 0$. (We can usually rearrange the eqs so that is the case.)

The Jacobi iteration

$$x_i^{(m+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j\ne i, j=1}^{n} a_{ij}x_j^{(m)}\right), \qquad i = 1, \dots, n \tag{8.22}$$

for $m = 0, 1, \dots,$ with given $x^{(0)}$.

Jacobi iteration

Example

Apply the Jacobi method to $3x + y = 5, x + 2y = 5$.

Using the initial guess $(x^{(0)}, y^{(0)}) = (0,0)$ we have $\begin{array}{l} x = \frac{5-y}{3} \\ y = \frac{5-x}{2} \end{array}$ and:

$$\begin{bmatrix} x^{(0)} \\ y^{(0)} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} x^{(1)} \\ y^{(1)} \end{bmatrix} = \begin{bmatrix} \frac{5-y^{(0)}}{3} \\ \frac{5-x^{(0)}}{2} \end{bmatrix} = \begin{bmatrix} \frac{5}{3} \\ \frac{5}{2} \end{bmatrix}$$

$$\begin{bmatrix} x^{(2)} \\ y^{(2)} \end{bmatrix} = \begin{bmatrix} \frac{5-y^{(1)}}{3} \\ \frac{5-x^{(1)}}{2} \end{bmatrix} = \begin{bmatrix} \frac{5}{6} \\ \frac{5}{3} \end{bmatrix}, \begin{bmatrix} x^{(3)} \\ y^{(3)} \end{bmatrix} = \begin{bmatrix} \frac{5-y^{(2)}}{3} \\ \frac{5-x^{(2)}}{2} \end{bmatrix} = \begin{bmatrix} \frac{10}{9} \\ \frac{25}{12} \end{bmatrix},$$

$$\ldots, \begin{bmatrix} x^{(m)} \\ y^{(m)} \end{bmatrix} \longrightarrow \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \text{the exact solution.}$$

Jacobi iteration

Example

Apply the Jacobi method to $x + 2y = 5, 3x + y = 5$.

Solve the 1st equation for x and 2nd for y, $\begin{array}{l} x = 5 - 2y \\ y = 5 - 3x \end{array}$.

The two equations are iterated as before, but the results are quite different:

$$\begin{bmatrix} x^{(0)} \\ y^{(0)} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} x^{(1)} \\ y^{(1)} \end{bmatrix} = \begin{bmatrix} 5 - 2y^{(0)} \\ 5 - 3x^{(0)} \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$$

$$\begin{bmatrix} x^{(2)} \\ y^{(2)} \end{bmatrix} = \begin{bmatrix} 5 - 2y^{(1)} \\ 5 - 3x^{(1)} \end{bmatrix} = \begin{bmatrix} -5 \\ -10 \end{bmatrix}, \begin{bmatrix} x^{(3)} \\ y^{(3)} \end{bmatrix} = \begin{bmatrix} 5 - 2y^{(2)} \\ 5 - 3x^{(2)} \end{bmatrix} = \begin{bmatrix} 25 \\ 20 \end{bmatrix}.$$

In this case, the Jacobi method fails, as the iteration diverges.

Why?

Jacobi iteration

Summary: in the kth equation solve for x_k in terms of the remaining unkowns.

Example: Apply the Jacobi method to
$$\begin{array}{rrrl} 9x_1 & +x_2 & +x_3 & = b_1 \\ 2x_1 & +10x_2 & +3x_3 & = b_2 \\ 3x_1 & +4x_2 & +11x_3 & = b_3 \end{array}$$

In this case
$$\begin{cases} x_1 &= \frac{1}{9}(b_1 - x_2 - x_3) \\ x_2 &= \frac{1}{10}(b_2 - 2x_1 - 3x_3) \\ x_3 &= \frac{1}{11}(b_3 - 3x_1 - 4x_2) \end{cases}$$

Let $x^{(0)} = [x_1^{(0)}, x_2^{(0)}, x_3^{(0)}]$ be an initial guess of the true solution x.

Then define an iteration sequence
$$\begin{cases} x_1^{(k+1)} &= \frac{1}{9}(b_1 - x_2^{(k)} - x_3^{(k)}) \\ x_2^{(k+1)} &= \frac{1}{10}(b_2 - 2x_1^{(k)} - 3x_3^{(k)}) \\ x_3^{(k+1)} &= \frac{1}{11}(b_3 - 3x_1^{(k)} - 4x_2^{(k)}) \end{cases}$$
for $k = 0, 1, 2, \cdots$.

`D = diag(A)`

`eig(eye(3)-inv(diag(D))*A)` $=$ -0.4472 , 0.1159, 0.3313

This is the *Jacobi iteration method* or the *method of simultaneous replacements*.

In the study of iterative methods, a most important issue is the convergence property.

We will give a number of the iterations for the case

$$b = [10; 19; 0]$$

which yields

$$x = [1; 2; -1]$$

In the table

$$\text{Error} = \|x - x^{(k)}\|$$

and

$Ratios$ denotes the ratio of successive values of the error.

Jacobi iteration

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$	Error	Ratio
0	0	0	0	2.00E+0	
1	1.1111	1.9000	0	1.00E+0	0.500
2	0.9000	1.6778	-0.9939	3.22E-1	0.322
3	1.0335	2.0182	-0.8556	1.44E-1	0.448
4	0.9819	1.9496	-1.0162	5.04E-2	0.349
5	1.0074	2.0085	-0.9768	2.32E-2	0.462
6	0.9965	1.9915	-1.0051	8.45E-3	0.364
7	1.0015	2.0022	-0.9960	4.03E-3	0.477
8	0.9993	1.9985	-1.0012	1.51E-3	0.375
9	1.0003	2.0005	-0.9993	7.40E-4	0.489
10	0.9999	1.9997	-1.0003	2.83E-4	0.382
30	1.0000	2.0000	-1.0000	3.01E-11	0.447
31	1.0000	2.0000	-1.0000	1.35E-11	0.447

Jacobi iteration

Let $A = L + D + U$, with D the main diagonal of A, L the lower triangular of A (entries below diagonal), U the upper triangle (entries above main diagonal); (different than those in the LU factorization).

$$\begin{aligned} Ax &= b \\ (D+L+U)x &= b \\ Dx &= b-(L+U)x \\ x &= D^{-1}(b-(L+U)x) \end{aligned}$$

Jacobi fixed point iteration

$x^{(0)}$ the initial guess

$$x^{(m+1)} = D^{-1}(b-(L+U)x^{(m)}), \quad m = 0, 1, \ldots$$

Example: $\begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \end{bmatrix} \Rightarrow \begin{bmatrix} x^{(m+1)} \\ y^{(m+1)} \end{bmatrix} = D^{-1}(b-(L+U)x^{(m)})$

$$= \begin{bmatrix} 1/3 & 0 \\ 0 & 1/2 \end{bmatrix}\left(\begin{bmatrix} 5 \\ 5 \end{bmatrix} - \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} x^{(k)} \\ y^{(k)} \end{bmatrix}\right) = \begin{bmatrix} \frac{5-y^{(k)}}{3} \\ \frac{5-x^{(k)}}{2} \end{bmatrix}$$

Jacobi iteration

Example: Let $\boldsymbol{A} = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 3 & -1 \\ 0 & -1 & 2 \end{bmatrix}, \boldsymbol{b} = \begin{bmatrix} 1 \\ 8 \\ -5 \end{bmatrix}$.

Carry out a number of the Jacobi iteration, starting with the 0 initial vector.

Rewriting the equations, we have the Jacobi method:

$$\begin{aligned} x_1^{(k)} &= \tfrac{1}{2} x_2^{(k-1)} + \tfrac{1}{2} \\ x_2^{(k)} &= \tfrac{1}{3} x_1^{(k-1)} + \tfrac{1}{3} x_3^{(k-1)} + \tfrac{8}{3} \\ x_3^{(k)} &= \tfrac{1}{2} x_2^{(k-1)} - \tfrac{5}{2} \end{aligned}$$

Taking the initial vector to be $\boldsymbol{x}^{(0)} = [0, 0, 0]^T$, we find (with the aid of a computer program or a programable calculator) that

$$\begin{aligned} \boldsymbol{x}^{(0)} &= [0, 0, 0]^T \\ \boldsymbol{x}^{(1)} &= [0.5000, 2.6667, -2.5000]^T \\ \boldsymbol{x}^{(2)} &= [1.8333, 2.000, -1.1667]^T \\ &\vdots \\ \boldsymbol{x}^{(21)} &= [2.0000, 3.0000, -1.0000]^T \end{aligned}$$

The actual solution (to four decimal places rounded) is obtained.

Jacobi iteration

In the Jacobi iteration $\boldsymbol{Q}$ is taken to be the diagonal of $\boldsymbol{A}$:

$$\boldsymbol{Q} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

Now

$$\boldsymbol{Q}^{-1} = \begin{bmatrix} \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{2} \end{bmatrix}, \quad \boldsymbol{Q}^{-1}A = \begin{bmatrix} 1 & -\frac{1}{2} & 0 \\ 0 & 1 & -\frac{1}{3} \\ 0 & -\frac{1}{2} & 1 \end{bmatrix}$$

The Jacobi iterative matrix and constant vector are

$$\boldsymbol{B} = \boldsymbol{I} - \boldsymbol{Q}^{-1}\boldsymbol{A} = \begin{bmatrix} 0 & \frac{1}{2} & 0 \\ \frac{1}{3} & 0 & \frac{1}{3} \\ 0 & \frac{1}{2} & 0 \end{bmatrix}, \quad \boldsymbol{h} = \boldsymbol{Q}^{-1}\boldsymbol{b} = \begin{bmatrix} \frac{1}{2} \\ \frac{8}{3} \\ -\frac{5}{2} \end{bmatrix}$$

One can see that $\boldsymbol{Q}$ is *close* to $\boldsymbol{A}$, $\boldsymbol{Q}^{-1}\boldsymbol{A}$ is *close* to $\boldsymbol{I}$, and $\boldsymbol{I} - \boldsymbol{Q}^{-1}\boldsymbol{A}$ is *small*. We write the Jacobi method

$$\boldsymbol{x}^{(k)} = \boldsymbol{B}\boldsymbol{x}^{(k-1)} + \boldsymbol{h}$$

Gauss-Seidel iteration

In the **Jacobi** method the equations are solved **in order**.
The components $x_i^{(m)}$ and the corresponding new values $x_i^{(m+1)}$ can be used immediately in their place.

The Gauss-Seidel iteration

is defined by

$$x_i^{(m+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(m+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(m)} \right), \qquad i = 1, \ldots, n \tag{8.23}$$

for $m = 0, 1, \ldots,$ with given $x^{(0)}$.

The idea for (8.23) is to *make immediate use of each newly computed iterate* in (8.22).

Gauss-Seidel iteration

Example: Apply the Gauss-Seidel method to $3x + y = 5, x + 2y = 5$.

$$\begin{bmatrix} x^{(0)} \\ y^{(0)} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} x^{(1)} \\ y^{(1)} \end{bmatrix} = \begin{bmatrix} \frac{5-y^{(0)}}{3} \\ \frac{5-x^{(1)}}{2} \end{bmatrix} = \begin{bmatrix} \frac{5}{3} \\ \frac{5}{3} \end{bmatrix}$$

$$\begin{bmatrix} x^{(2)} \\ y^{(2)} \end{bmatrix} = \begin{bmatrix} \frac{5-y^{(1)}}{3} \\ \frac{5-x^{(2)}}{2} \end{bmatrix} = \begin{bmatrix} \frac{10}{9} \\ \frac{35}{18} \end{bmatrix}, \begin{bmatrix} x^{(3)} \\ y^{(3)} \end{bmatrix} = \begin{bmatrix} \frac{5-y^{(2)}}{3} \\ \frac{5-x^{(3)}}{2} \end{bmatrix} = \begin{bmatrix} \frac{55}{54} \\ \frac{215}{108} \end{bmatrix}.$$

Gauss-Seidel iteration

$x^{(0)}$ the initial guess

$$x^{(m+1)} = D^{-1}(b - Ux^{(m)} - Lx^{(m+1)}), \quad m = 0, 1, \ldots$$

or $$x^{(m+1)} = -(L + D)^{-1}Ux^{(m)} + (L + D)^{-1}b.$$

Gauss-Seidel iteration

Example: Let $\boldsymbol{A} = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 3 & -1 \\ 0 & -1 & 2 \end{bmatrix}$, $\boldsymbol{b} = \begin{bmatrix} 1 \\ 8 \\ -5 \end{bmatrix}$.
Carry out a number of the Gauss-Seidel iteration, starting with the 0 initial vector.

The idea of the Gauss-Seidel iteration is simply to accelerate the convergence by incorporating each vector as soon as it has been computed. Obviously, it would be more efficient in the Jacobi method to use the updated value $x_1^{(k)}$ in the second equation instead of the old value $x_1^{(k-1)}$. Similarly, $x_2^{(k)}$ could be used in the third equation in place of $x_2^{(k-1)}$. Using the new iterates as soon as they become available, we have the Gauss-Seidel method:

$$\begin{aligned} x_1^{(k)} &= \tfrac{1}{2}x_2^{(k-1)} + \tfrac{1}{2} \\ x_2^{(k)} &= \tfrac{1}{3}x_1^{(k)} + \tfrac{1}{3}x_3^{(k-1)} + \tfrac{8}{3} \\ x_3^{(k)} &= \tfrac{1}{2}x_2^{(k)} - \tfrac{5}{2} \end{aligned}$$

Starting with the initial vector zero, some of the iterates are

$$\begin{aligned} \boldsymbol{x}^{(0)} &= [0, 0, 0]^T \\ \boldsymbol{x}^{(1)} &= [0.5000, 2.8333, -1.0833]^T \\ \boldsymbol{x}^{(2)} &= [1.9167, 2.9444, -1.0278]^T \\ &\vdots \\ \boldsymbol{x}^{(9)} &= [2.0000, 3.0000, -1.0000]^T \end{aligned}$$

In this example, the convergence of the Gauss-Seidel method is approximately **twice as fast** as that of the Jacobi method.

Gauss-Seidel iteration

In the Gauss-Seidel iterative method, $\boldsymbol{Q}$ is chosen as the lower triangular part of $\boldsymbol{A}$, including the diagonal. Using the data from the previous example, we now find that

$$\boldsymbol{Q} = \begin{bmatrix} 2 & 0 & 0 \\ -1 & 3 & 0 \\ 0 & -1 & 2 \end{bmatrix}$$

The usual row operations give us

$$\boldsymbol{Q}^{-1} = \begin{bmatrix} \frac{1}{2} & 0 & 0 \\ \frac{1}{6} & \frac{1}{3} & 0 \\ \frac{1}{12} & \frac{1}{6} & \frac{1}{2} \end{bmatrix}, \quad \boldsymbol{Q}^{-1}A = \begin{bmatrix} 1 & -\frac{1}{2} & 0 \\ 0 & \frac{5}{6} & -\frac{1}{3} \\ 0 & -\frac{1}{2} & \frac{5}{6} \end{bmatrix}$$

Again, we emphasize that in a practical problem we would not compute $\boldsymbol{Q}^{-1}$. The Gauss-Seidel iterative matrix and constant vector are

$$\mathcal{L} = \boldsymbol{I} - \boldsymbol{Q}^{-1}\boldsymbol{A} = \begin{bmatrix} 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{6} & \frac{1}{3} \\ 0 & \frac{1}{12} & \frac{1}{6} \end{bmatrix}, \quad \boldsymbol{h} = \boldsymbol{Q}^{-1}\boldsymbol{b} = \begin{bmatrix} \frac{1}{2} \\ \frac{17}{6} \\ -\frac{13}{12} \end{bmatrix}$$

We write the Gauss-Seidel method

$$\boldsymbol{x}^{(k)} = \mathcal{L}\boldsymbol{x}^{(k-1)} + \boldsymbol{h}$$

SOR iteration

An acceleration of the Gauss-Seidel method is possible by the introduction of a relaxation factor ω, resulting in the

Successive OverRelaxation (SOR) method

$$x_i^{(m+1)} = \omega \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(m+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(m)} \right) + (1-\omega) x_i^{(m)}, \tag{8.24}$$

$$i = 1, \dots, n$$

for $m = 0, 1, \dots,$ with given $x^{(0)}$.

The SOR method with $\omega = 1$ reduces to the Gauss-Seidel method.

SOR iteration

Example: Let $\boldsymbol{A} = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 3 & -1 \\ 0 & -1 & 2 \end{bmatrix}$, $\boldsymbol{b} = \begin{bmatrix} 1 \\ 8 \\ -5 \end{bmatrix}$. Carry out a number of the SOR iteration using $\omega = 1.1$.

$$\begin{aligned} x_1^{(k)} &= \omega\left[\tfrac{1}{2}x_2^{(k-1)} + \tfrac{1}{2}\right] + (1-\omega)x_1^{(k-1)} \\ x_2^{(k)} &= \omega\left[\tfrac{1}{3}x_1^{(k)} + \tfrac{1}{3}x_3^{(k-1)} + \tfrac{8}{3}\right] + (1-\omega)x_2^{(k-1)} \\ x_3^{(k)} &= \omega\left[\tfrac{1}{2}x_2^{(k)} - \tfrac{5}{2}\right] + (1-\omega)x_3^{(k-1)} \end{aligned}$$

Starting with the initial vector zero, some of the iterates are

$$\begin{aligned} \boldsymbol{x}^{(0)} &= [0, 0, 0]^T \\ \boldsymbol{x}^{(1)} &= [0.5500, 3.1350, -1.0257]^T \\ \boldsymbol{x}^{(2)} &= [2.2193, 3.0574, -0.9658]^T \\ &\vdots \\ \boldsymbol{x}^{(7)} &= [2.0000, 3.0000, -1.0000]^T \end{aligned}$$

In this example, the convergence of the SOR method is faster than the Gauss-Seidel method.

SOR iteration

In the SOR method, $\boldsymbol{Q}$ is chosen as the lower triangular part of $\boldsymbol{A}$ including the diagonal, but each diagonal element a_{ij} is replaced by $\frac{a_{ij}}{\omega}$, where ω is the so-called **relaxation factor**. From the previous example, this means that

$$\boldsymbol{Q} = \begin{bmatrix} \frac{20}{11} & 0 & 0 \\ -1 & \frac{30}{11} & 0 \\ 0 & -1 & \frac{20}{11} \end{bmatrix}$$

Now

$$\boldsymbol{Q}^{-1} = \begin{bmatrix} \frac{11}{20} & 0 & 0 \\ \frac{121}{600} & \frac{11}{30} & 0 \\ \frac{1331}{12000} & \frac{121}{600} & \frac{11}{20} \end{bmatrix}, \quad \boldsymbol{Q}^{-1}A = \begin{bmatrix} \frac{11}{10} & -\frac{11}{20} & 0 \\ \frac{11}{300} & \frac{539}{600} & -\frac{11}{30} \\ \frac{121}{6000} & \frac{671}{12000} & \frac{539}{600} \end{bmatrix}$$

The SOR iterative matrix and constant vector are

$$\mathcal{L}_\omega = \boldsymbol{I} - \boldsymbol{Q}^{-1}\boldsymbol{A} = \begin{bmatrix} -\frac{1}{10} & \frac{11}{20} & 0 \\ -\frac{11}{300} & \frac{61}{600} & \frac{11}{30} \\ -\frac{121}{6000} & -\frac{671}{12000} & \frac{61}{600} \end{bmatrix}, \quad \boldsymbol{h} = \boldsymbol{Q}^{-1}\boldsymbol{b} = \begin{bmatrix} \frac{11}{20} \\ \frac{627}{200} \\ -\frac{4103}{4000} \end{bmatrix}$$

We write the SOR method as

$$\boldsymbol{x}^{(k)} = \mathcal{L}_\omega \boldsymbol{x}^{(k-1)} + \boldsymbol{h}$$

Pseudocode

```
procedure $Jacobi(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{x})$
real $kmax \leftarrow 100, \delta \leftarrow 10^{-10}, \varepsilon \leftarrow \frac{1}{2} \times 10^{-4}$
integer $i, j, kmax, n;$      real $diag, sum;$
real array $(\boldsymbol{A})_{1:n \times 1:n}, (\boldsymbol{b})_{1:n}, (x)_{1:n}, (y)_{1:n}$
$n \leftarrow size(\boldsymbol{A})$
for $k = 1$ to $kmax$ do
  $y \leftarrow x$
  for $i = 1$ to $n$ do
    $sum \leftarrow b_i$
    $diag \leftarrow A_{ii}$
    if $|diag| < \delta$ then
      output "diagonal element too small"
      return
    end if
    for $j = 1$ to $n$ do
      if $j \neq i$ then
        $sum \leftarrow sum - A_{ij} y_j$
      end if
    end for
    $x_i \leftarrow \frac{sum}{diag}$
  end for
  output $k, x$
  if $\|\boldsymbol{x} - \boldsymbol{y}\| < \varepsilon$ then
    output $k, x$
    return
  end if
end for
output "maximum iterations reached"
return
end $Jacobi$
```

Pseudocode

Here, the vector $\boldsymbol{y}$ contains the old iterate values, and the vector $\boldsymbol{x}$ contains the updated ones. The values of $kmax, \delta$ and ε are set either in a parameter statement or as a global variables.

Pseudocode

The pseudocode for the procedure $Gauss\_Seidel(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{x})$ would be the same as that for the Jacobi pseudocode above, except that the innermost j-loop would be replaced by the following

```
for $j = 1$ to $i - 1$ do
   $sum \leftarrow sum - a_{ij}x_j$
end for
for $j = i + 1$ to $n$ do
   $sum \leftarrow sum - a_{ij}x_j$
end for
```

Pseudocode

The pseudocode for procedure $SOR(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{x})$ would be the same as that for the Gauss-Seidel pseudocode with the statement following the j-loop replaced by the following:

```
$x_i \leftarrow \frac{sum}{diag}$
$x_i \leftarrow \omega x_i + (1 - \omega) y_i$
```

Convergence Theorems

For the analysis of the method described by system (8.17), i.e.,

$$\boldsymbol{Q}\boldsymbol{x}^k = (\boldsymbol{Q} - \boldsymbol{A})\boldsymbol{x}^{(k-1)} + \boldsymbol{b},$$

we write

$$\boldsymbol{x}^k = \boldsymbol{Q}^{-1}[(\boldsymbol{Q} - \boldsymbol{A})\boldsymbol{x}^{(k-1)} + \boldsymbol{b}] \tag{8.25}$$

or

$$\boldsymbol{x}^k = \mathcal{G}\boldsymbol{x}^{(k-1)} + \boldsymbol{h}$$

where the iteration matrix ad vector are

$$\mathcal{G} = \boldsymbol{I} - \boldsymbol{Q}^{-1}\boldsymbol{A}, \qquad \boldsymbol{h} = \boldsymbol{Q}^{-1}\boldsymbol{b}.$$

Notice that in the pseudocode, we do **not** compute $\boldsymbol{Q}^{-1}$. The matrix $\boldsymbol{Q}^{-1}$ is used to facilitate the analysis. Now let $\boldsymbol{x}$ be the solution of system (8.13). Since $\boldsymbol{A}$ is nonsingular, $\boldsymbol{x}$ exists and is unique. We have from equation (8.25)

$$\begin{aligned}\boldsymbol{x}^{(k)} - \boldsymbol{x} &= (\boldsymbol{I} - \boldsymbol{Q}^{-1}\boldsymbol{A})\boldsymbol{x}^{(k-1)} - \boldsymbol{x} + \boldsymbol{Q}^{-1}\boldsymbol{b}\\ &= (\boldsymbol{I} - \boldsymbol{Q}^{-1}\boldsymbol{A})\boldsymbol{x}^{(k-1)} - (\boldsymbol{I} - \boldsymbol{Q}^{-1}\boldsymbol{A})\boldsymbol{x}\\ &= (\boldsymbol{I} - \boldsymbol{Q}^{-1}\boldsymbol{A})(\boldsymbol{x}^{(k-1)} - \boldsymbol{x})\end{aligned}$$

Convergence Theorems

$$\boldsymbol{x}^{(k)} - \boldsymbol{x} = (\boldsymbol{I} - \boldsymbol{Q}^{-1}\boldsymbol{A})(\boldsymbol{x}^{(k-1)} - \boldsymbol{x})$$

One can interpret $\boldsymbol{e}^{(k)} \equiv \boldsymbol{x}^{(k)} - \boldsymbol{x}$ as the current **error vector**.

Thus, we have

$$\boldsymbol{e}^{(k)} = (\boldsymbol{I} - \boldsymbol{Q}^{-1}\boldsymbol{A})\boldsymbol{e}^{(k-1)} \tag{8.26}$$

We want $\boldsymbol{e}^{(k)}$ to become *smaller* as k increases.
Equation (8.26) shows that $\boldsymbol{e}^{(k)}$ will be *smaller* than $\boldsymbol{e}^{(k-1)}$ if $\boldsymbol{I} - \boldsymbol{Q}^{-1}\boldsymbol{A}$ is *small*, in some sense.
In turn, that means that $\boldsymbol{Q}^{-1}\boldsymbol{A}$ should be *close* to $\boldsymbol{I}$.
Thus, $\boldsymbol{Q}$ should be *close* to $\boldsymbol{A}$.
(Norms can be used to make *small* and *close* precise.)

Convergence Theorems

Theorem (Spectral radius theorem)

The sequence generated by

$$\boldsymbol{Q}\boldsymbol{x}^{(k)} = (\boldsymbol{Q} - \boldsymbol{A})\boldsymbol{x}^{(k-1)} + \boldsymbol{b}$$

converges, no matter what starting point $\boldsymbol{x}^{(0)}$ is selected,
if an only if
all eigenvalues of $\boldsymbol{I} - \boldsymbol{Q}^{-1}\boldsymbol{A}$ lie in the open unit disc, $|z| < 1$, in the complex plane.

The conclusion of this theorem can also be written as

$$\rho(\boldsymbol{I} - \boldsymbol{Q}^{-1}\boldsymbol{A}) < 1$$

where ρ is the **spectral radius** function:

Definition (Spectral Radius)

$\forall \boldsymbol{M} \in \mathcal{M}(n \times n)$ with eigenvalues λ_i,

$$\rho(\boldsymbol{M}) := \max_{1 \le i \le n} |\lambda_i|.$$

Convergence Theorems

Example

Determine whether the Jacobi, Gauss-Seidel and SOR methods (with $\omega = 1.1$) of the previous examples converge for all initial iterates.

$\boldsymbol{A} = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 3 & -1 \\ 0 & -1 & 2 \end{bmatrix}, \boldsymbol{b} = \begin{bmatrix} 1 \\ 8 \\ -5 \end{bmatrix}.$

For the **Jacobi** method, we can easily compute the eigenvalues of the relevant matrix $\boldsymbol{B}$. The steps are

$$\det(\boldsymbol{B} - \lambda \boldsymbol{I}) = \det \begin{bmatrix} -\lambda & \frac{1}{2} & 0 \\ \frac{1}{3} & -\lambda & \frac{1}{3} \\ 0 & \frac{1}{2} & -\lambda \end{bmatrix} = -\lambda^3 + \frac{1}{6}\lambda + \frac{1}{6}\lambda = 0$$

The eigenvalues are $\lambda = 0, \pm\sqrt{\frac{1}{3}} \approx \pm 0.5774$. Thus, by the preceding theorem, the Jacobi iteration succeeds for ay starting vector in this example.

Convergence Theorems

For the **Gauss-Seidel** method, the eigenvalues of the iteration matrix $\mathcal{L}$ are determined from

$$\det(\mathcal{L}-\lambda \boldsymbol{I}) = \det\begin{bmatrix} -\lambda & \frac{11}{20} & 0 \\ 0 & \frac{1}{6}-\lambda & \frac{1}{3} \\ 0 & \frac{1}{12} & \frac{1}{6}-\lambda \end{bmatrix} = -\lambda\left(\frac{1}{6}-\lambda\right)^2 + \frac{1}{36}\lambda = 0$$

$\lambda = 0, 0, \frac{1}{3}$ hence Gauss-Seidel will also converge for any initial vector.

```
D=diag(D); eig(eye(3)-inv(diag(D))*A)
```

Convergence of Jacobi

With $e^{(m)} = x - x^{(m)}$ we have from (8.21) and (8.22)

$$e_i^{(m+1)} = \frac{-1}{a_{ii}} \sum_{j \neq i, j=1}^{n} a_{ij} e_j^m$$

or, in matrix form

$$e^{(m+1)} = M e^{(m)}, \quad m \geq 0$$

$$M = -\begin{bmatrix} 0 & \frac{a_{12}}{a_{11}} & \cdots & & \frac{a_{1n}}{a_{11}} \\ \frac{a_{21}}{a_{22}} & 0 & & & \frac{a_{2n}}{a_{22}} \\ \vdots & \ddots & & & \vdots \\ & & & 0 & \frac{a_{n-1,n}}{a_{n-1,n-1}} \\ \frac{a_{n1}}{a_{nn}} & & \cdots & \frac{a_{n,n-1}}{a_{nn}} & 0 \end{bmatrix}$$

which implies

$$e^{(m)} = M^m e^{(0)}, \qquad m \geq 0.$$

Convergence of Jacobi

For arbitrary $x^{(0)}$, the Jacobi iteration converges:

$$x^{(m)} \to x \quad \Leftrightarrow \quad \rho(M) < 1$$

which is satisfied if $\|M\| < 1$ for some matrix norm.

1. $\|M\|_\infty < 1$, i.e., A is strictly diagonally dominant:

$$\sum_{j \neq i, j=1}^{n} |a_{ij}| < |a_{ii}|, \qquad i = 1, \ldots, n$$

   implying

$$\|x - x^{(m+1)}\|_\infty \leq \|M\|_\infty \|x - x^{(m)}\|_\infty, \quad m \geq 0.$$

2. $\|M\|_1 < 1$, i.e.,

$$\sum_{i=1, j \neq i}^{n} \left| \frac{a_{ij}}{a_{ii}} \right| < 1, \qquad j = 1, \ldots, n$$

   implying

$$\|x - x^{(m+1)}\|_1 \leq \|M\|_1 \|x - x^{(m)}\|_1, \quad m \geq 0.$$

Convergence of Jacobi

Theorem

Let $A \in \mathscr{M}_n$ be strictly diagonally dominant. Then

1. *$(\forall)b$ and $(\forall)x^{(0)}$, the Jacobi method converges to the (unique) solution.*
2. *A is a nonsingular matrix.*

Proof: **1)** We will check $\rho(D^{-1}(L+U)) < 1$. Let λ, v be an eigenvalue and the corresponding eigenvector of $D^{-1}(L+U)$: $(L+U)v = \lambda D v$. Let $1 \le m \le n$ be the index such that $v_m = 1$ and all other components less than 1. Since $(L+U)_{mm} = 0$, calculating the mth component, we have

$$\underbrace{\sum_{i\neq m} |(L+U)_{mi}|}_{<|d_{mm}|} \geq |\sum_{i=1}^{n} (L+U)_{mi} v_i| = |\lambda d_{mm} v_m| = |\lambda||d_{mm}| \Rightarrow |\lambda| < 1. \blacksquare$$

Convergence of Gauss-Seidel

From (8.21) and (8.23) we have that the error satisfies

$$e_i^{(m+1)} = -\sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} e_j^{m+1} - \sum_{j=i+1}^{n} \frac{a_{ij}}{a_{ii}} e_j^{m}, \tag{8.27}$$

for $i = 1, \ldots, n$ or again as

$$e^{(m+1)} = \widetilde{M} e^{(m)}, \quad m \geq 0$$

for a matrix $\widetilde{M}$. Bounding $\|\widetilde{M}\|$ is more difficult than bounding $\|M\|$. Introduce

$$\alpha_i = \sum_{j=1}^{i-1} \left| \frac{a_{ij}}{a_{ii}} \right|, \quad \beta_i = \sum_{j=i+1}^{n} \left| \frac{a_{ij}}{a_{ii}} \right|,$$

with $\alpha_1 = \beta_n = 0$. From (8.27) we get

$$|e_i^{(m+1)}| \leq \alpha_i \|e^{(m+1)}\|_\infty + \beta_i \|e^{(m)}\|_\infty, \quad i = 1, \ldots, n. \tag{8.28}$$

Convergence of Gauss-Seidel

If k is the index for which

$$|e_k^{(m+1)}| = \|e^{(m+1)}\|_\infty$$

then (8.28) implies

$$\|e^{(m+1)}\|_\infty \le \frac{\beta_k}{1-\alpha_k}\|e^{(m)}\|_\infty \le \underbrace{\max_i \frac{\beta_i}{1-\alpha_i}}_{\eta}\|e^{(m)}\|_\infty \le \eta^{m+1}\|e^{(0)}\|_\infty .$$

Assuming that the Jacobi iteration converges because $\|M\|_\infty = \max_i(\alpha_i + \beta_i) < 1$, since

$$0 \le \frac{\alpha_i(1-\alpha_i-\beta_i)}{1-\alpha_i} = (\alpha_i+\beta_i) - \frac{\beta_i}{1-\alpha_i}$$

we have that

$$\eta = \max_i \frac{\beta_i}{1-\alpha_i} \le \|M\|_\infty < 1$$

and the (probably faster) convergence of Gauss-Seidel iteration.

Convergence of Gauss-Seidel

Theorem

Let $A \in \mathscr{M}_n$ be strictly diagonally dominant. Then

1. *$(\forall)b$ and $(\forall)x^{(0)}$, the Gauss-Seidel method converges to a solution.*
2. *A is a nonsingular matrix.*

Proof: **1)** We will check $\rho((L+D)^{-1}U) < 1$. Let λ, v be an eigenvalue and the corresponding eigenvector of $(L+D)^{-1}U$: $\lambda(D+L)v = Uv$. Let $1 \leq m \leq n$ be the index such that $v_m = 1$ and all other components less than 1. Calculating the mth component, we have

$$|\lambda| \sum_{i>m} |a_{mi}| < |\lambda| \Big(|a_{mm}| - \sum_{i<m} |a_{mi}| \Big) \leq |\lambda| \Big(|a_{mm}| - \sum_{i<m} |a_{mi} v_i| \Big)$$

$$\leq |\lambda| \Big| a_{mm} + \sum_{i<m} a_{mi} v_i \Big| = \Big| \sum_{i>m} a_{mi} v_i \Big| \leq \Big| \sum_{i>m} a_{mi} \Big| \Rightarrow |\lambda| < 1. \blacksquare$$

Exercise:

$$A = \begin{bmatrix} 3 & 1 & -1 \\ 1 & -5 & 2 \\ 1 & 6 & 8 \end{bmatrix}, \quad B = \begin{bmatrix} 3 & 2 & 6 \\ 1 & 8 & 1 \\ 9 & 2 & -2 \end{bmatrix}.$$

Note that A is diagonally dominant, B is not.
But if we interchange 1st and 3rd rows of B, the iteration will converge.

A general framework

Assuming that A can be decomposed as

$$A = N - P$$

then the system $Ax = b$ can be rewritten as

$$Nx = b + Px$$

which yields an iteration

$$Nx^{(m+1)} = b + Px^{(m)}, m = 0, 1, \ldots.$$

The idea being that solving the linear system $Nz = g$ is much easier than solving $Ax = b$.
The error satisfies

$$Ne^{(m+1)} = Pe^{(m)} \Leftrightarrow e^{(m+1)} = N^{-1}Pe^{(m)} \equiv Me^{(m)}$$

$$e^{(m)} = M^m e^{(0)}, \qquad m \geq 0$$

hence the method converges iff the spectral radius of $M = N^{-1}P$ satisfies

$$\rho(M) < 1.$$

Jacobi: Let $P = N - A$ with

$$N = \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \vdots & & \ddots & \\ 0 & 0 & \dots & a_{nn} \end{bmatrix}.$$

Solving $Nz = g \Leftrightarrow$ linear system with a diagonal coefficient matrix.

Gauss-Seidel: Let $P = N - A$ with

$$N = \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ a_{21} & a_{22} & \dots & 0 \\ \vdots & & \ddots & \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}.$$

Solving $Nz = g \Leftrightarrow$ linear system with a lower triangular coefficient matrix.

Theorem

Let A be Hermitian with positive diagonal elements. Then the Gauss-Seidel method for solving $Ax = b$ converges $(\forall)x^{(0)}$ iff the matrix A is positive definite.

Checking convergence

For an iteration scheme $x^{(m+1)} = Mx^{(m)}$ we have a recursive error formula

$$x - x^{(m+1)} = M(x - x^{(m)}), \qquad m \geq 0$$

and

$$x^{(m+1)} - x^{(m)} = M(^{(m)} - x^{(m-1)}) \quad \Rightarrow \quad \underbrace{\frac{\|x^{(m+1)} - x^{(m)}\|}{\|x^{(m)} - x^{(m-1)}\|}}_{c_m} \leq \|M\| = c$$

Approximating $c_m \approx c < 1$ we can estimate the error by

$$\|x - x^{(m+1)}\| \leq \frac{c}{1-c}\|x^{(m)} - x^{(m+1)}\|$$

since

$$\|x^{(m)} - x^{(m+1)}\| = \|x^{(m)} - x + x - x^{(m+1)}\| \geq \|x^{(m)} - x\| - \|x - x^{(m+1)}\|$$
$$\geq \tfrac{1}{c}\|x^{(m+1)} - x\| - \|x - x^{(m+1)}\|$$

Cost comparison

For $Ax = b$ what is the cost of iteration vs. Gaussian elimination $(\frac{2}{3}n^3)$?
Jacobi & Gauss-Seidel: $2n^2$/iteration if A is dense, for Poisson's equation the cost being $5n$ operations/iteration.
Hence we need to know $\nu(n)$ the number of iterations/iteration.
In order to have the rate of convergence

$$\|x - x^{(m)}\| \leq \varepsilon \|x - x^{(0)}\|$$

since

$$\|x - x^{(m+1)}\| \leq c\|x - x^{(m)}\| \leq c^{m+1}\|x - x^{(0)}\|, \quad m \geq 0 \text{ for some } c < 1$$

we should require

$$c^m \leq \varepsilon, \quad m \geq \frac{-\log \varepsilon}{-\log c} \equiv m^*$$

The total cost of solving $Ax = b$ by iteration is

$$\text{Cost}(c, \varepsilon, n) = m^* \nu(n)$$

when

Cost comparison

When

$$\text{Cost }(c, \varepsilon, n) \leq \frac{2}{3}n^3?$$

Assuming $\nu(n) = 2n^2$ (as when A is dense):

$$m^* \cdot 2n^2 \leq \frac{2}{3}n^3$$

the iteration is more efficient than Gaussian elimination if

$$n \geq n_c = 3\frac{-\log u}{-\log c}$$

solving for $\varepsilon \approx u$, the unit round.

Slow convergence

Let $c = 1 - \delta$ for some positive $\delta \approx 0$. Then

$$-\log(c) = -\log(1-\delta) \approx \delta$$

Then the cost becomes

$$Cost(c, \varepsilon, n) = m^* \nu(n) \approx \frac{-\log \varepsilon}{\delta} \nu(n).$$

Then doubling δ to 2δ will halve the total cost. This is of importance for linear systems $Ax = b$ that arise from discretization of Poissons equation. In that case, $\delta \to 0$ as $N \to \infty$.

Acceleration Methods

Successive Over-Relaxation (SOR)

SOR takes Gauss-Seidel direction toward the solution and overshoots to try to speed the convergence.

With $\omega \in \mathbb{R}$ the *acceleration parameter*, $\omega > 1$ for **over-relaxation**:

$$x^{(m+1)} = \omega x^{(m+1)}_{\text{Gauss-Seidel}} + (1-\omega)x^{(m)}, \quad \text{i.e.,}$$

$$x_i^{(m+1)}{}_{\text{Gauss-Seidel}} = \frac{1}{a_{ii}}\Big(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(m+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(m)}\Big)$$

$$x_i^{(m+1)} = \omega x_i^{(m+1)}{}_{\text{Gauss-Seidel}} + (1-\omega)x_i^{(m)}$$

Remark: SOR with $\omega = 1$ is exactly Gauss-Seidel.

SOR

$$x^{(m+1)} = (D+\omega L)^{-1}[(1-\omega)D - \omega U] + \omega(D+\omega L)^{-1}b$$

- If $\omega \notin (0,2)$, then the SOR method does not converge.
- If A is SPD, $\omega \in (0,2)$, then SOR converges for any $x^{(0)}$.

Example

$$\begin{bmatrix} 3 & 1 & -1 \\ 2 & 4 & 1 \\ -1 & 2 & 5 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \\ 1 \end{bmatrix}, \quad \text{exact solution: } [2, -1, 1].$$

Gauss-Seidel and SOR:

$$\begin{aligned} u^{(m+1)} &= \tfrac{4-v^{(m)}+w^{(m)}}{3}, & u^{(m+1)} &= (1-\omega)u^{(m)} + \omega\tfrac{4-v^{(m)}+w^{(m)}}{3} \\ v^{(m+1)} &= \tfrac{1-2u^{(m+1)}-w^{(m)}}{4}, & u^{(m+1)} &= (1-\omega)v^{(m)} + \omega\tfrac{1-2u^{(m+1)}-w^{(m)}}{4} \\ w^{(m+1)} &= \tfrac{1+u^{(m+1)}-2v^{(m)}}{5}, & w^{(m+1)} &= (1-\omega)w^{(m)} + \omega\tfrac{1+u^{(m+1)}-2v^{(m)}}{5} \end{aligned}$$

with $[u^{(0)}, v^{(0)}, w^{(0)}] = [0, 0, 0]$

Gauss-Seidel and SOR with $\omega = 1.25$

$$\begin{aligned} u^{(1)} &= \tfrac{4-0+0}{3} = \tfrac{4}{3} \approx 1.333, & u^{(1)} &\approx 1.6667 \\ v^{(1)} &= -\tfrac{5}{12} \approx -0.4167, & u^{(1)} &\approx -0.7292 \\ w^{(1)} &= \tfrac{19}{30} \approx 0.6333, & w^{(1)} &\approx 1.0312 \\ u^{(2)} &= \tfrac{101}{60} \approx 1.6833, & u^{(2)} &\approx 1.9835 \\ v^{(2)} &= -\tfrac{3}{4} \approx -0.7500, & u^{(2)} &\approx -1.0672 \\ w^{(2)} &= \tfrac{251}{300} \approx 0.8367, & w^{(2)} &\approx 1.0216 \end{aligned}$$

Jacobi iteration for Poisson's equation

$$\mathbf{u}_h^{(m+1)} = Q\mathbf{u}_h^{(m)} + \mathbf{G}_h$$

where

$$Q = \frac{1}{4}\begin{bmatrix} T & I & 0 & \dots & 0 & 0 \\ I & T & I & & & \\ & & & & & \vdots \\ 0 & I & T & I & & \vdots \\ \vdots & & & \ddots & & \\ 0 & & & I & T & I \\ 0 & & \dots & & I & T \end{bmatrix}, I \in \mathscr{M}_{n-1}, T = \begin{bmatrix} 0 & 1 & \dots & & 0 \\ 1 & 0 & 1 & & \\ \vdots & & \ddots & & \vdots \\ & & 1 & 0 & 1 \\ 0 & & \dots & 1 & 0 \end{bmatrix},$$

$$\mathbf{u}_h^{(m)} = \begin{bmatrix} \mathbf{u}_h^{(m)}(*, y_1) \\ \mathbf{u}_h^{(m)}(*, y_2) \\ \vdots \\ \mathbf{u}_h^{(m)}(*, y_{N-1}) \end{bmatrix}, \mathbf{u}_h^{(m)}(*, y_k) = \begin{bmatrix} \mathbf{u}_h^{(m)}(x_1, y_k) \\ \mathbf{u}_h^{(m)}(x_2, y_k) \\ \vdots \\ \mathbf{u}_h^{(m)}(x_{N-1}, y_k) \end{bmatrix},$$

$$\mathbf{G}_h = \begin{bmatrix} \frac{1}{4}\{\mathbf{B}(y_1) - h^2\mathbf{G}_h(*, y_1)\} \\ \frac{1}{4}\{\mathbf{B}(y_2) - h^2\mathbf{G}_h(*, y_2)\} \\ \vdots \\ \frac{1}{4}\{\mathbf{B}(y_{N-1}) - h^2\mathbf{G}_h(*, y_{N-1})\} \end{bmatrix}, \mathbf{B}(y_k) = \begin{bmatrix} f(x_0, y_k) \\ 0 \\ \vdots \\ 0 \\ f(x_N, y_k) \end{bmatrix}, \mathbf{G}_h(*, y_k) = \begin{bmatrix} g(x_1, y_k) \\ g(x_2, y_k) \\ \vdots \\ g(x_{N-1}, y_k) \end{bmatrix}.$$

Jacobi iteration for Poisson's equation

The original discretization for Poisson's equation can be rewritten as

$$\mathbf{u}_h^{(m+1)} = Q\mathbf{u}_h^{(m)} + \mathbf{G}_h,$$

hence the linear system $Ax = b$, with $A = I - Q$ (the diagonal elements of A are positive).
The matrix $M \equiv Q$ in the Jacobi iteration, hence we have convergence iff

$$\rho(Q) < 1.$$

Since $\|Q\|_1 = \|Q\|_\infty = 1$, we compute $\|Q\|_2 = \rho_2(Q)$

Gauss-Seidel iteration for Poisson's equation

Letting N be the lower triangular portion of $A = I - Q$ and $P = N - A$, we write

$$N\mathbf{u}_h^{(m+1)} = \mathbf{G}_h + P\mathbf{u}_h^{(m)}$$

Rates of convergence

- Jacobi: $\xi = 1 - 2\sin^2\left(\frac{\pi}{2N}\right)$
- Gauss-Seidel: ξ^2
- SOR: $\omega^* - 1$ where $\omega = \frac{2}{1+\sqrt{1-\xi^2}}$

Example. Compare Jacobi, Gauss-Seidel and SOR on this 6×6 system

$$\begin{bmatrix} 3 & -1 & 0 & 0 & 0 & \frac{1}{2} \\ -1 & 3 & -1 & 0 & \frac{1}{2} & 0 \\ 0 & -1 & 3 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & 0 \\ 0 & \frac{1}{2} & 0 & -1 & 3 & -1 \\ \frac{1}{2} & 0 & 0 & 0 & -1 & 3 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{bmatrix} = \begin{bmatrix} \frac{5}{2} \\ \frac{3}{2} \\ 1 \\ 1 \\ \frac{3}{2} \\ \frac{5}{2} \end{bmatrix}$$

The solution $x = [1, 1, 1, 1, 1, 1]$. The approximate solution vectors x_6, after running six steps of each of the three methods:

Jacobi	Gauss-Seidel	SOR
0.9879	0.9950	0.9989
0.9846	0.9946	0.9993
0.9674	0.9969	1.0004
0.9674	0.9996	1.0009
0.9846	1.0016	1.0009
0.9879	1.0013	1.0004

The parameter $\omega = 1.1$ for SOR.

Sparse matrix computations

Example. Use the Jacobi method to solve the 100,000 equation version of the previous example.

Let $n \in \mathbb{N}$ an even integer, then consider the $n \times n$ matrix A with 3 on the main diagonal, -1 on the super- and subdiagonal, and $\frac{1}{2}$ in the $(i, n+1-i)$ position for all $i = 1 : n$, except for $i = \frac{n}{2}$ and $\frac{n}{2} + 1$. For $n = 12$

$$A = \begin{bmatrix}
3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \\
-1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 \\
0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\
0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 3 & -1 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{2} & 0 & -1 & 3 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 \\
0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 \\
0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 \\
\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 3
\end{bmatrix}.$$

Define the vector $b = [2.5, 1.5, \ldots, 1.5, 1.0, 1.0, 1.5, \ldots, 1.5, 2.5]$, where there are $n-4$ repetitions of 1.5 and 2 repetitions of 1.0.
Note that if $n = 6$, A and b define the previous system.
The solution of this general system is $[1, \ldots, 1]$.

No row of A has more than 4 nonzero entries. Since fewer than $4n$ of the n^2 potential entries are nonzero, we may call the matrix A **sparse**.

Sparse matrix computations

If we want to solve this system for $n = 100,000$ or more, what are the options?

- Treating A as a full matrix means storing $n^2 = 10^{10}$ entries, each as a floating point double precision number requiring 8 bytes of storage. Hence 8×10^{10} which is $\approx$ 80 gigabytes. So, depending on computer, it may be impossible to fit the entire n^2 entries into RAM.
- Not only the size, but also time poses a problem. The number of operations required by Gaussian elimination will be $\mathcal{O}(n^3) \approx 10^{15}$. If your machine runs on the order of a few GHz(10^9 cycles per second), an upper bound on the number of floating point operations per second is around 10^8. Therefore, $10^{15}/10^8 = 10^7$ is a reasonable guess at the number of seconds required for Gaussian elimination.
  There are 3×10^7 seconds in a year.

Gaussian elimination is not an overnight computation.

Sparse matrix computations

On the other hand, one step of an iterative method will require approximately $2 \times 4n = 800,000$ operations, two for each nonzero matrix entry.
We could do 100 steps of Jacobi iteration and still finish with fewer than 10^8 operations, which should take roughly a second or less.
For the system just defined, with $n = 100,000$ the following `jacobi.m` needs only 50 steps to converge from a starting guess of $(0, \dots, 0)$ to the solution $(1, \dots, 1)$ within six correct decimal places.
The 50 steps require less than 1 second on a typical PC.

Sparse matrix computations

```
% Program 2.1 Sparse matrix setup
% Input:  n = size of system
% Outputs:  sparse matrix a, r.h.s.  b
function [a,b] = sparsesetup(n)
e = ones(n,1);
n2=n/2;
a = spdiags([-e 3*e -e],-1:1,n,n); % Entries of a
c=spdiags([e/2],0,n,n);c=fliplr(c);a=a+c;
a(n2+1,n2) = -1; a(n2,n2+1) = -1; % Fix up 2 entries
b=zeros(n,1); % Entries of r.h.s.  b
b(1)=2.5;b(n)=2.5;b(2:n-1)=1.5;b(n2:n2+1)=1;
```

```
% Program 2.2 Jacobi Method
% Inputs:  full or sparse matrix a, r.h.s.  b,
% number of Jacobi iterations k
% Output:  solution x
function x = jacobi(a,b,k)
n=length(b); % find n
d=diag(a); % extract diagonal of a
r=a-diag(d); % r is the remainder
x=zeros(n,1); % initialize vector x
for j=1:k % loop for Jacobi iteration
  x=(b-r*x)./d;
end % End of Jacobi iteration loop
```

The conjugate gradient method

Definition

- $A \in \mathscr{M}_n(\mathbb{R})$ is **symmetric** if $A = A^T$,
- and is **positive definite** if $x^T A x > 0, \forall x \neq 0$.

Example: $A = \begin{bmatrix} 2 & 2 \\ 2 & 5 \end{bmatrix}$ is SPD, $B = \begin{bmatrix} 2 & 4 \\ 4 & 5 \end{bmatrix}$ is not positive definite.
$x^T A x = 2(x_1 + x_2)^2 + 3x_2^2$, $x^T B x = 2(x_1 + 2x_2)^2 - 3x_2^2$.

Definition

- The **inner product** of two vectors $\boldsymbol{u} = (u_1, u_2, \ldots, u_n)$ and $\boldsymbol{u} = (v_1, v_2, \ldots, v_n)$ is defined as
$$\langle \boldsymbol{u}, \boldsymbol{v} \rangle = \boldsymbol{u}^{\boldsymbol{T}} \boldsymbol{v} = \sum_{i=1}^n u_i v_i$$
- The vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ are **orthogonal** is
$$\langle \boldsymbol{u}, \boldsymbol{v} \rangle = 0.$$

Definition

- The $\boldsymbol{A}$**-inner product** of two vectors is defined as $\boldsymbol{u}$ and $\boldsymbol{v}$
$$\langle \boldsymbol{u}, \boldsymbol{v} \rangle_A = \langle \boldsymbol{Au}, \boldsymbol{v} \rangle = \boldsymbol{u^T A^T v}$$
- Two vectors $\boldsymbol{u} \neq \boldsymbol{0} \neq \boldsymbol{v}$ are $\boldsymbol{A}$**-conjugate** if
$$\langle \boldsymbol{u}, \boldsymbol{v} \rangle_A = 0$$
- $A \in \mathscr{M}_n(\mathbb{R})$ is **positive definite** if
$$\langle \boldsymbol{u}, \boldsymbol{u} \rangle_A > 0, \quad \forall u \neq 0.$$

Definition

- A **quadratic form** is a scalar quadratic function of a vector
$$f(\boldsymbol{x}) = \tfrac{1}{2}\langle x, x\rangle_A - \langle b, x\rangle + c.$$
- The **gradient** of a quadratic form
$$f'(\boldsymbol{x}) = \nabla f(\boldsymbol{x}) = \left[\frac{\partial f(\boldsymbol{x})}{\partial x_1}, \frac{\partial f(\boldsymbol{x})}{\partial x_2}, \ldots, \frac{\partial f(\boldsymbol{x})}{\partial x_n}\right]^T.$$

We can derive
$$f'(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{A}^T\boldsymbol{x} + \frac{1}{2}\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}$$
which reduces to
$$f'(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}$$
if $\boldsymbol{A}$ is symmetric.

Optimization theory

Assume A is SPD

To solve $Ax = b$ is the same as finding the minimum of the energy function
$$f(x) = \frac{1}{2}x^T Ax - b^T x.$$

If x^* is a solution, $Ax^* = b$,

$$f(x) = \frac{1}{2}\underbrace{(x^* - x)^T A(x^* - x)}_{\geq 0} \underbrace{- \frac{1}{2}b^T x^*}_{=f(x^*)}$$

then f and $x \to (x^* - x)^T A(x^* - x)$ reach the minimum at the same point.

Gradient method with complete line search

We want to minimize f on an open set $M \subset \mathbb{R}^n$:

```
choose $x_0 \in M$
for $k \geq 0$ do
(i) find the direction
        $d_k = -\nabla f(x_k)$
(ii) line search: find a point $t = \alpha_k$ on the line
     $\{x_k + td_k : t \in \mathbb{R}\} \cap M$ where $f$ reaches a local minimum
(iii) $x_{k+1} = x_k + \alpha_k d_k$
```

Here $f = \frac{1}{2}x^T Ax - b^T x \Rightarrow \nabla f = Ax - b \Rightarrow d_k = b - Ax_k$ and we can solve directly for α_k:

$$\alpha_k = \frac{d_k^T d_k}{d_k^T A d_k}$$

$(b = Ax_{k+1} = Ax_k + \alpha_k Ad_k = b - d_k + \alpha_k Ad_k)$.

Gradient method with complete line search

Let define the **energy norm**:

$$\|x\|_A = \sqrt{x^T A x}.$$

Recall that

$f(x) = \frac{1}{2}(x^* - x)^T A(x^* - x) + f(x^*) = \frac{1}{2}\|x^* - x\|_A^2 + f(x^*)$, hence

$$\|x_{k+1} - x^*\|_A^2 = 2f(x_{k+1}) - 2f(x^*) = 2f(x_k + \alpha_k d_k) - 2f(x^*)$$

$$\overset{\alpha_k = \frac{d_k^T d_k}{\|d_k\|_A}}{=} \|x_k - x^*\|_A^2 \left(1 - \frac{(d_k^T d_k)^2}{\|d_k\|_A^2 \|x_k - x^*\|_A^2}\right)$$

Now

$$\|x_k - x^*\|_A^2 = (x_k - x^*)^T A(x_k - x^*) = (A^{-1}d_k)^T A A^{-1} d_k = d_k^T A^{-1} d_k,$$

since $d_k = b - Ax_k = A(x^* - x_k) \Leftrightarrow x_k - x^* = -A^{-1}d_k$.

Kantorovitch inequality: Let A be SPD. Then $\forall u \neq 0$

$$\frac{(u^T A u)(u^T A^{-1} u)}{(u^T u)^2} \leq \left(\frac{1}{2}\sqrt{\text{cond}(A)} + \frac{1}{2}\frac{1}{\sqrt{\text{cond}(A)}}\right)^2.$$

Gradient method with complete line search

Hence

$$\left(\frac{\|x_{k+1}-x^*\|_A}{\|x_k-x^*\|_A}\right)^2 = 1 - \frac{(d_k^T d_k)}{(d_k^T A d_k)(d_k^T A^{-1} d_k)} \le \left(\frac{\text{cond}(A)-1}{\text{cond}(A)+1}\right)^2$$

Theorem: convergence

$$\|x_k - x^*\|_A \le \left(\frac{\text{cond}(A)-1}{\text{cond}(A)+1}\right)^k \|x_0 - x^*\|_A$$

Remark: if $\text{cond}(A)$ is large:

$$\frac{\text{cond}(A)-1}{\text{cond}(A)+1} = \left(1-\frac{1}{\text{cond}(A)}\right)\left(\frac{1}{1+\frac{1}{\text{cond}(A)}}\right)$$

$$\approx \left(1-\frac{1}{\text{cond}(A)}\right)\left(1-\frac{1}{\text{cond}(A)}\right) \approx 1-\frac{2}{\text{cond}(A)}.$$

Conjugate Gradient Method (CG) 1952

Use directions that are conjugate

Definition

x and y are conjugate (A-orthogonal) if $x^T A y = 0$.

If $d_0, d_1, \ldots, d_{n-1}$ are conjugate (basis in $\mathbb{R}^n$), then

$$x^* = \sum_{j=0}^{n-1} \alpha_j d_j \Rightarrow Ax^* = \sum_{j=0}^{n-1} \alpha_j A d_j, \quad d_i^T A x^* = \sum_{j=0}^{n-1} \alpha_j \underbrace{d_i^T A d_j}_{=0, i \neq j}$$

hence $$\alpha_i = \frac{d_i^T A x^*}{\|d_i\|_A^2} = \frac{d_i^T b}{\|d_i\|_A}.$$

Lemma

$d_0, d_1, \ldots, d_{n-1}$ conjugate directions, $x_0 \in \mathbb{R}^n$, and $\forall k \geq 0$

$$x_{k+1} = x_k + \alpha_k d_k, \text{ with } \alpha_k = -\frac{d_k^T (Ax_k - b)}{\|d_k\|_A^2}.$$

The sequence $\{x_k\}_{k \geq 0}$ **computes** $A^{-1}b$ after n steps: $x_n = A^{-1}b$.

Conjugate Gradient Method

Proof: Write $x^* - x_0 = \sum_{j=0}^{n-1} \beta_j d_j$ and prove that $\beta_j = \frac{d_j^T A(x^* - x_0)}{\|d_j\|_A^2}$, so

$$x^* - x_0 = \sum_{j=0}^{n-1} \frac{d_j^T A(x^* - x_0)}{\|d_j\|_A^2} d_j.$$

But $$x_n - x_0 = \sum_{j=0}^{n-1} \alpha_j d_j = \sum_{j=0}^{n-1} \frac{d_j^T(-Ax_j + b)}{\|d_j\|_A^2} d_j$$

hence

$$x^* - x_n = \sum_{j=0}^{n-1} \frac{d_j^T A(x_j - x_0)}{\|d_j\|_A^2} d_j = \sum_{j=\mathbf{1}}^{n-1} \frac{d_j^T A(x_j - x_0)}{\|d_j\|_A^2} d_j = 0$$

since $x_j - x_0$ is A-orthogonal to d_j, as a linear combination of $d_0, \ldots, d_{j-1}$:

$$x_j = x_0 + \sum_{k=0}^{j-1} \alpha_k d_k \quad \blacksquare$$

Remark: x_k minimizes f not only on the line $\{x_{k-1} + td_{k-1} : t \in \mathbb{R}\}$, but also over $x_0 + V_k$, where $V_k = span\{d_0, d_1, \dots, d_{k-1}\}$.

CG algorithm

- $x_0 \in \mathbb{R}^n, d_0 = -g_0 = b - Ax_0$
- for $k \geq 0$ do

$$\begin{aligned}
&\alpha_k = \tfrac{g_k^T g_k}{\|d_k\|_A^2} \\
&x_{k+1} = x_k + \alpha_k d_k \\
&g_{k+1} = g_k + \alpha_k Ad_k \\
&\beta_k = \frac{g_{k+1}^T g_{k+1}}{g_k^T g_k}, \quad d_{k+1} = -g_{k+1} + \beta_k d_k
\end{aligned}$$

The title of the method comes from what the Conjugate Gradient Method is really doing: sliding down the slopes of a quadratic paraboloid in n dimensions.

- The "**gradient**" means that is finding the direction of fastest decline by using calculus; and
- "**Conjugate**" means, not quite that its individual steps are orthogonal to one another, but that at least the residuals r_i are.

Example

$$\begin{bmatrix} 2 & 2 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 6 \\ 3 \end{bmatrix}.$$

Using the CG algorithm we get:

$$x_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad d_0 = \begin{bmatrix} 6 \\ 3 \end{bmatrix}, g_0 = \begin{bmatrix} -6 \\ -3 \end{bmatrix}$$

$$\alpha_0 = \frac{\begin{bmatrix} -6 \\ -3 \end{bmatrix}^T \begin{bmatrix} -6 \\ -3 \end{bmatrix}}{\begin{bmatrix} 6 \\ 3 \end{bmatrix}^T \begin{bmatrix} 2 & 2 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} 6 \\ 3 \end{bmatrix}} = \frac{45}{6 \cdot 18 + 3 \cdot 27} = \frac{5}{21}$$

$$x_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \frac{5}{21} \begin{bmatrix} 6 \\ 3 \end{bmatrix} = \begin{bmatrix} \frac{10}{7} \\ \frac{5}{7} \end{bmatrix}$$

$$g_1 = \begin{bmatrix} -6 \\ -3 \end{bmatrix} + \frac{5}{21} \begin{bmatrix} 18 \\ 27 \end{bmatrix} = 12 \begin{bmatrix} -\frac{1}{7} \\ \frac{2}{7} \end{bmatrix}$$

Example

$$\beta_0 = \frac{g_1^T g_1}{g_0^T g_0} = \frac{144 \cdot 5/49}{36+9} = \frac{16}{49}$$

$$d_1 = 12 \begin{bmatrix} \frac{1}{7} \\ -\frac{2}{7} \end{bmatrix} + \frac{16}{49} \begin{bmatrix} 6 \\ 3 \end{bmatrix} = \begin{bmatrix} \frac{180}{49} \\ -\frac{120}{49} \end{bmatrix}$$

$$\alpha_1 = \frac{\begin{bmatrix} -\frac{12}{7} \\ \frac{24}{7} \end{bmatrix}^T \begin{bmatrix} -\frac{12}{7} \\ \frac{24}{7} \end{bmatrix}}{\begin{bmatrix} \frac{180}{49} \\ -\frac{120}{49} \end{bmatrix}^T \begin{bmatrix} 2 & 2 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} \frac{180}{49} \\ -\frac{120}{49} \end{bmatrix}} = \frac{7}{10}$$

$$x_2 = \begin{bmatrix} \frac{10}{7} \\ \frac{5}{7} \end{bmatrix} + \frac{7}{10} \begin{bmatrix} \frac{180}{49} \\ -\frac{120}{49} \end{bmatrix} = \begin{bmatrix} 4 \\ -1 \end{bmatrix}$$

$$g_2 = 12 \begin{bmatrix} -\frac{1}{7} \\ \frac{2}{7} \end{bmatrix} + \frac{7}{10} \begin{bmatrix} 2 & 2 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} \frac{180}{49} \\ -\frac{120}{49} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Since $g_2 = b - Ax_2 = 0$, the solution is $x_2 = [4, -1]$.

Remarks

CG is simpler than Gaussian elimination in the sense that writing the code - there are no triple loops.
Both are direct methods: give the correct solution in a finite number of steps.
Operation count: if A is dense CG takes n^3 compared to $\frac{1}{3}n^3$ for Gaussian elimination.

But, if A **is sparse**, assuming that n is too large for the $\frac{n^3}{3}$ operations for Gaussian elimination to be feasible:

- Gaussian elimination must run to completion to give x
- CG gives an approximation x_i at each step.

Monitoring the residuals g_i, a good enough solution may be found to avoid completing all n steps (behaves like an *iterative method*).

Properties of CG-method

If $g_{k-1} \neq 0$:

(i) $d_{k-1} \neq 0$

(ii) if $V_k = span\{g_0, Ag_0, \dots, A^{k-1}g_0\}$, then
$V_k = span\{g_0, g_1, \dots, g_{k-1}\}$
$V_k = span\{d_0, d_1, \dots, d_{k-1}\}$

(iii) $d_k^T A d_\ell = 0, k \neq \ell$

(iv) $f(x_k) = \min_{z \in V_k} f(x_0 + z)$.

Convergence depends on

$$\left(\frac{\text{cond}(A) - 1}{\text{cond}(A) + 1}\right).$$

CG performs worse than Gaussian elimination on ill-conditioned matrices with partial pivoting.

Preconditioning

The method fell out of favor shortly after its discovery because of its susceptibility to accumulation of round-off errors when A is an ill-conditioned matrix.

In fact, its performance on ill-conditioned matrices is inferior to Gaussian elimination with partial pivoting.

This obstruction is relieved by *preconditioning*, which essentially changes the problem to one of solving a better-conditioned matrix system, after which the Conjugate Gradient is applied.

Preconditioning

Solving $Ax = b$ with $\text{cond}(A)$ large, by multiplying with a matrix C:

$$CAx = Cb, \qquad \text{with } \text{cond}(CA) \ll \text{cond}(A), \text{ and } C \approx A^{-1}.$$

PCG algorithm

- $x_0 \in \mathbb{R}^n, g_0 = b - Ax_0$
  $$d_0 = -C^{-1}g_0 = -h_0$$
- for $k \geq 0$ do
  $$\begin{aligned} x_{k+1} &= x_k + \alpha_k d_k, \qquad \alpha_k = \frac{g_k^T h_k}{\|d_k\|_A^2} \\ g_{k+1} &= g_k + \alpha_k A d_k \\ h_{k+1} &= C^{-1} g_{k+1} \\ d_{k+1} &= -h_{k+1} + \frac{g_{k+1}^T h_{k+1}}{g_k^T h_k} d_k \end{aligned}$$

The convergence depends now on $\text{cond}(CA)$.

Example **Jacobi**: $C = D^{-1}$

SOR: $C = \dfrac{1}{\omega(2-\omega)}(D - \omega L)D^{-1}(D - \omega L^T)$, where $A = D - L - L^T$.

GMRES: solver for non symmetric matrices.

Example. Apply the Conjugate Gradient Method method to solve the 100,000 equation from a previous example.

$$A = \begin{bmatrix} 3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \\ -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 3 & -1 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & -1 & 3 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 \\ \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 3 \end{bmatrix}.$$

After 20 steps of the CG, the difference between the computed solution x and the true solution $(1, \ldots, 1)$ is less than 10^{-9} in the vector infinity norm.

The total time of execution was less than 1 second on a PC.

Vector Spaces

Examples:

- $\mathbb{R}^n, \mathbb{C}^n \ni v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} = [v_1, \ldots, v_n]^T$
- $\mathcal{P}_n = \{p \mid p(x)$ a polynomial of degree $\leq n\}$
- $C[a,b] = \{f | f(x)$ continuous on $[a,b]\}$

Operations: $+, \cdot$ by scalars (commutative and distributive laws).

Definition

$\mathscr{W} \subset \mathscr{V}$ is a subspace if is a vector space on its own w.r.t. vector addition and scalar multiplication inherited from $\mathscr{V}$.

Basis

Definition

- Vectors $v_1, \ldots, v_m$ are dependent iff $\exists$ scalars $c_1, \ldots, c_m$ (not all zero) s.t.
$$c_1 v_1 + \ldots + c_m v_m = 0.$$
- The vectors $v_1, \ldots, v_m$ are independent if they are **not dependent**.

Definition

The set of vectors $\{v_1, \ldots, v_m\}$ is a basis for the vector space $\mathscr{V}$ if $\forall v \in \mathscr{V} \exists!$ (unique) choice of coefficients $c_1, \ldots, c_m$ s.t.
$$v = c_1 v_1 + \ldots + c_m v_m.$$
If such a basis exists $\Rightarrow \mathscr{V}$ is a finite dimensional space, of dimension m.

Theorem

If $\mathscr{V}$ has a finite basis $\{v_1, \ldots, v_m\}$, then every basis for $\mathscr{V}$ has exactly m elements.

Basis

- $\mathbb{R}^n$ and $\mathbb{C}^n$ are n-dimensional with basis:
$$e_i = [0,\ldots,0,\underbrace{1}_{i^{th}\text{position}},0,\ldots,0] \qquad i = 1,\ldots,n$$
- $\dim(\mathscr{P}_n) = n+1$, with
  - standard basis:
$$\{1, x, x^2, \ldots, x^N\}$$
  - the *Legendre polynomials* $\{P_0(x),\ldots,P_n(x)\}$:
$$P_n(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}\left[(x^2-1)^n\right]$$
  - *Chebyshev polynomials* $\{T_0(x),\ldots,T_N(x)\}$.
- The multivariable polynomials
$$\sum_{i=j\leq n} a_{ij}x^i y^j$$
of degree $\leq n$ are a vector space of dimension
$$\tfrac{1}{2}(n+1)(n+2)$$
- $C[a,b]$ is infinite dimensional

Matrices

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}_{m\times n}$$

Recall:

- $A^T = A$ symmetric $\Leftrightarrow A = (a_{ij}) = (a_{ji})$
- $A^T = -A$ skew-symmetric
- $A = A^* \equiv \overline{A}^T$ hermitian (Ex.:$\begin{bmatrix} 3 & 2+i \\ 2-i & 1 \end{bmatrix}$)
- $U \in \mathbb{R}^{n\times n}$ orthogonal: $UU^T = U^TU = I$.

  Rotation matrix: $\begin{bmatrix} \cos\theta & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}$ rotates a vector θ degrees.
- $U \in \mathbb{C}^{n\times n}, U$ unitary: $UU^* = U^*U = I$

Remark

Let $U = [u_1|u_2|\dots|u_n]$, $u_i = i^{th}$ column of U. If $UU^T = U^TU = I$ (U is unitary) $\Rightarrow$ the columns $\{u_1, u_2 \dots, u_n\}$ form an orthonormal basis for $\mathbb{R}^n$.

$$U^T = \begin{bmatrix} u_1^T \\ u_2^T \\ \vdots \\ u_n^T \end{bmatrix} \Rightarrow U^TU = \begin{bmatrix} u_1^T \\ u_2^T \\ \vdots \\ u_n^T \end{bmatrix} [u_1|u_2|\dots|u_n] = (u_i^T u_j) = I$$

$$\Rightarrow \underbrace{u_i^T u_i}_{u_i \cdot u_i} = 1, \quad \underbrace{u_i^T u_j}_{u_i \cdot u_j} = 0, i \neq j \quad \Rightarrow u_i \perp u_j \text{ orthogonal in } \mathbb{R}^n$$

For complex matrices, U is unitary if $U^*U = UU^* = I$. The columns (and the rows) of U form an orthogonal basis in $\mathbb{C}^n$.

Linear systems

The linear system of m equations and n unknowns $x_1, \ldots, x_n$

$$\begin{cases} a_{11}x_1 + & \ldots + & a_{1n}x_n = b_1 \\ \vdots & & \\ a_{m1}x_1 + & \ldots + & a_{mn}x_n = b_m \end{cases} \quad \text{can be written as} \quad Ax = b$$

$$A = \begin{bmatrix} a_{11} & \ldots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \ldots & a_{mn} \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}.$$

Theorem

$A \in \mathbb{R}^{n\times n}$, $\mathscr{V} \subset \mathbb{R}^n$. *The following statements are equivalent:*

(a) $Ax = b$ *has a unique solution* $x \in \mathscr{V}$, $\forall b \in \mathscr{V}$.

(b) $Ax = b$ *has a solution* $x \in \mathscr{V}$, $\forall b \in \mathscr{V}$.

(c) $Ax = 0 \Rightarrow x = 0$

(d) A^{-1} *exists* $\Leftrightarrow \det(A) \neq 0 \Leftrightarrow$ *rank* $A = n$.

Linear systems

Definition

- *Row rank* of A is the number of independent rows in A (as elements in $\mathbb{R}^n$).
- *Column rank* of A is the number of independent columns in A

It can be shown *row rank* A = *column rank* A :=**rank**(A)

Remark: $Ax \equiv x_1 A_{\star 1} + \ldots + x_n A_{\star n}$, with $A_{\star 1}, \ldots, A_{\star n}$ the columns of A.

Inner product

$$(x,y) := \sum_{i=1}^{n} x_i y_i = x^T y = y^T x, \qquad \text{for } x, y \in \mathbb{R}^n$$

$$(x,y) := \sum_{i=1}^{n} x_i \overline{y}_i = y^* x, \qquad \text{for } x, y \in \mathbb{C}^n$$

- Euclidean norm: $\|x\|_2 = (x,x)^{\frac{1}{2}} = \left(\sum_{i=1}^{n} |x_i|^2\right)^{\frac{1}{2}}, x \in \mathbb{R}^n$ or $\mathbb{C}^n$
- Cauchy-Schwarz ineq.: $|(x,y)| \leq \|x\|_2 \|y\|_2, \quad x, y \in \mathscr{V}$

  $0 \leq (x + \alpha y)^2, \forall \alpha \in \mathbb{R}$
- Triangular inequality: $\|x+y\|_2 \leq \|x\|_2 + \|y\|_2, \quad x, y \in \mathscr{V}$

Angles

In real inner product spaces: θ is the angle:

$$\cos\theta = \frac{(x,y)}{\|x\|_2\|y\|_2}, \qquad 0 \leq \theta\pi$$

$x, y \in \mathbb{V}$ are orthogonal if

$$(x,y) = 0$$

If $\{u_1, \ldots, u_n\}$ is an orthogonal basis for $\mathbb{R}^n$, and $x \in \mathbb{R}^n$

$$x = c_1 u_1 + \ldots + c_n u_n$$

then

$$x = (x, u_1)u_1 + \ldots + (x, u_n)u_n$$

since $(x, u_k) = c_k = |x||u_k|\cos\theta_k = |x|\cos\theta_k$.

Eigenvalues and eigenvectors

Definition

1. λ is an eigenvalue for $A \in \mathscr{M}_{n\times n}$ if
$$Ax = \lambda x, \qquad \text{for some } x \neq 0.$$
2. x is called an eigenvector of A associated with the eigenvalue λ.

Example

- $\begin{bmatrix} 1 & 3 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 4 \begin{bmatrix} 1 \\ 1 \end{bmatrix}$
- $\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 0 \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

$$A = \begin{bmatrix} 1.25 & 0.75 \\ 0.75 & 1.25 \end{bmatrix}, \quad \boldsymbol{\lambda_1 = 2}, v^{(1)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \boldsymbol{\lambda_2 = 0.5}, v^{(2)} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}.$$

For any given $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ we can write

$$x = c_1 v^{(1)} + c_2 v^{(2)}$$

with

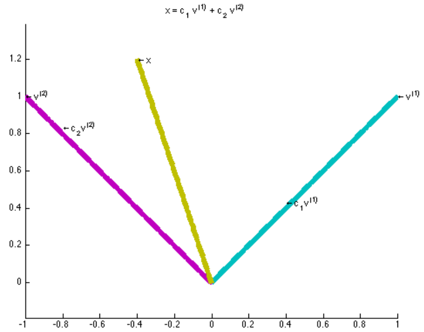
$$c_1 = \tfrac{x_1 + x_2}{2}, \quad c_2 = \tfrac{x_2 - x_1}{2}$$

and

$$Ax = \mathbf{2} c_1 v^{(1)} + \mathbf{0.5} c_2 v^{(2)}$$

Ax is based on doubling the length of the $c_1 v^{(1)}$ part of x and halving the $c_2 v^{(2)}$ component.

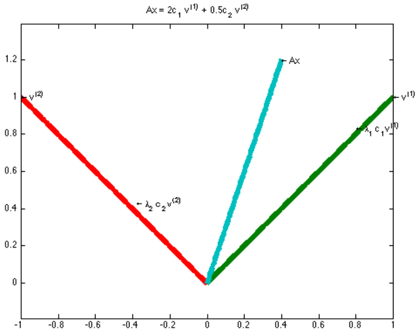
$c1 = .4, c2 = .8.$



(a)

Figure: (a) $x = c_1 v^{(1)} + c_2 v^{(2)}$ and (b) $Ax = 2c_1 v^{(1)} + 0.5c_2 v^{(2)}$

$c1 = .4, c2 = .8.$



(a)

Figure: (a) $x = c_1 v^{(1)} + c_2 v^{(2)}$ and (b) $Ax = 2c_1 v^{(1)} + 0.5c_2 v^{(2)}$

The problem which led to calculating Ax might have been better formulated in the coordinate system $v^{(1)}, v^{(2)}$.

Characteristic polynomial

Solving $Ax = \lambda x$ for $x \neq 0$ is equivalent to the linear homogeneous system:

$$(A - \lambda I)x = 0$$

which has a solution iff

$$\mathscr{P}_n \ni \underbrace{det(A - \lambda I)}_{\text{characteristic polynomial}} = 0. \qquad \text{(characteristic equation)} \tag{8.29}$$

$$\begin{aligned} \text{So } f(\lambda) = det(A - \lambda I) &= det \begin{bmatrix} a_{11} - \lambda & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} - \lambda \end{bmatrix} \\ &= (a_{11} - \lambda) \dots (a_{nn} - \lambda) + O(\lambda^{n-2}) \\ &= (-1)^n \lambda^n + (-1)^{n-1} \underbrace{(a_{11} + \ldots + a_{nn})}_{Tr(A)} \lambda^{n-1} + \ldots + det(A) \qquad (8.30) \\ &= (-1)^n (\lambda - \lambda_1)(\lambda - \lambda_2) \dots (\lambda - \lambda_n), \text{where } \lambda_i \text{ are eigenvalues}, 1 \leq i \leq n. \end{aligned}$$

$n = 2$

For the case $n = 2$

$$f(\lambda) = det \begin{bmatrix} \lambda - a_{11} & -a_{12} \\ -a_{21} & \lambda - a_{22} \end{bmatrix}$$

$$= (\lambda - a_{11})(\lambda - a_{22}) - a_{21}a_{12} = \lambda^2 - (a_{11} + a_{22})\lambda + a_{11}a_{22} - a_{21}a_{12}$$

Example

For the earlier example A

$$f(\lambda) = det \begin{bmatrix} \lambda - 1.25 & -0.75 \\ -0.75 & \lambda - 1.25 \end{bmatrix} = \lambda^2 - 2.5\lambda + 1$$

which has the roots $\lambda = .5, 2$.

Application: Linear ODEs

Theorem

If λ is an eigenvalue of the matrix $\boldsymbol{A}$ and if $\boldsymbol{v}$ is a corresponding eigenvector, then one solution of the DE

$$\boldsymbol{x}' = \boldsymbol{A}\boldsymbol{x}$$

is

$$\boldsymbol{x}(t) = e^{\lambda t}\boldsymbol{v}.$$

Characteristic equation

$$det(A-\lambda I) \in \mathscr{P}_n \Rightarrow \begin{cases} A \text{ has at least one eigenvalue} \\ A \text{ has at most } n \text{ distinct eigenvalues} \end{cases}$$

Definition

- algebraic multiplicity of λ: multiplicity of λ as a root of (8.29)
- geometric multiplicity of λ: number of independent eigenvectors associated with λ

Example

1. $A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \lambda = 1 \begin{cases} \text{algebraic multiplicity} = 2 \\ \text{geometric multiplicity} = 2 \end{cases}$
2. $A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \lambda = 1 \begin{cases} \text{algebraic multiplicity} = 2 \\ \text{geometric multiplicity} = 1 \end{cases}$

Matlab

$$[V, D] = \texttt{eig}(A), \qquad A = \begin{bmatrix} 1 & 3 & -7 \\ -3 & 4 & 1 \\ 2 & -5 & 3 \end{bmatrix}$$

returns

- the eigenvectors in the array V as columns, and
- the eigenvalues in the diagonal array D:

```
>> [V,D]=eig(A)
V =
0.7267                0.7267                0.7916
-0.0680 + 0.4533i    -0.0680 - 0.4533i    0.5137
-0.3395 - 0.3829i    -0.3395 + 0.3829i    0.3308

D =
3.9893 + 5.5601i     0                     0
0                     3.9893 - 5.5601i     0
0                     0                     0.0214
```

Eigenvalues for symmetric matrices. Theorem

Let $A \in \mathcal{M}(n)$ be symmetric. Then there is a set of n eigenvalue-eigenvector pairs $\{\lambda_i, v^{(i)}\}_{1\le i\le n}$ that satisfy

i. $\lambda_1, \lambda_2, \dots, \lambda_n$ are all roots of $f(\lambda)$ and $\lambda_i \in \mathbb{R}$.

ii. The column vectors v^i are mutually perpendicular and of length 1.

iii. For each $x = [x_1, x_2, \dots, x_n]^T$ there is a unique choice of constants $c_1, \dots, c_n$ for which

$$x = c_1 v^{(1)} + \cdots c_n v^{(n)}.$$

The constants are given by $c_i = \sum_{j=1}^n x_j v_j^{(i)} = x^T v^{(i)}, \quad 1 \le i \le n.$

iv. The matrix

$$U = [v^{(1)}, v^{(2)}, \dots, v^{(n)}]$$

satisfies

$$(a)\ U^T A U = D \equiv \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \lambda_n \end{bmatrix},$$

$$(b)\ UU^T = U^T U = I.$$

Eigenvalues for symmetric matrices. Example

$$A = \begin{bmatrix} 1.25 & 0.75 \\ 0.75 & 1.25 \end{bmatrix}, \quad \boldsymbol{\lambda_1 = 2}, v^{(1)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \boldsymbol{\lambda_2 = 0.5}, v^{(2)} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}.$$

To make them length 1:

$$\boldsymbol{\lambda_1 = 2}, v^{(1)} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}, \quad \boldsymbol{\lambda_2 = 0.5}, v^{(2)} = \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

The matrix U is then

$$U = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}.$$

Check that $U^T U = I$ and $U^T A U = \begin{bmatrix} 2 & 0 \\ 0 & 0.5 \end{bmatrix}$.

The nonsymmetric eigenvalue problem

With nonsymmetric matrices, there are many possibilities for the eigenvalues and eigenvectors.

- The roots of the characteristic polynomial $f(\lambda)$ may be complex numbers,
- and these lead to eigenvectors with complex numbers as components.
- For multiple roots of $f(\lambda)$, it may not be possible to express an arbitrary vector x as a combination of eigenvectors,
- and there is no longer a simple formula for the coefficients.

The nonsymmetric eigenvalue problem. Example - complex eigenvalues

For $A = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$,
the characteristic polynomial is

$$f(\lambda) = det \begin{bmatrix} \lambda & -1 \\ 1 & \lambda \end{bmatrix} = \lambda^2 + 1$$

with complex roots

$$\lambda_1 = i \equiv \sqrt{-1}, \qquad \lambda_2 = -i$$

and the corresponding eigenvectors are

$$v^{(1)} = \begin{bmatrix} i \\ -1 \end{bmatrix}, v^{(2)} = \begin{bmatrix} i \\ 1 \end{bmatrix}.$$

The nonsymmetric eigenvalue problem. Ex. - multiple eigenvalues, few eigenvectors

Let $A = \begin{bmatrix} a & 1 & 0 \\ 0 & a & 1 \\ 0 & 0 & a \end{bmatrix}$ with a constant.

Check that $\lambda = a$ is the eigenvalue of A with multiplicity 3, and any associated eigenvector must be of the form

$$v = c \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

for some $c \neq 0$.
Thus, up to a constant we have only one eigenvector

$$v = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

for three equal eigenvalues $\lambda_1 = \lambda_2 = \lambda_3 = a$.

Similar Matrices

Definition

B is similar to A ($A \sim B$) if $\exists$ nonsingular (invertible) P:

$$B = P^{-1}AP$$

Assume A and B are similar. Then

1. $det(A - \lambda I) = det(B - \lambda I)$ (equal characteristic polynomials)
2. $det(A) = det(B)$
3. $Trace(A) = Trace(B)$

- $\det(B - \lambda I) = \det(P^{-1}AP - \lambda I) = \det(P^{-1}(A - \lambda I)P) = \det(P^{-1}) \det(A - \lambda I) \det(P) = \frac{1}{\det(P)} \det(A - \lambda I) \det(P) = \det(A - \lambda I)$

  From (8.30), for similar matrices we have that the coefficients of the characteristic polynomials are the same, hence
- $Trace(A) \equiv (\lambda_1 + \ldots + \lambda_n) = Trace(B)$
- $\det(A) = \lambda_1 \ldots \lambda_n = \det(B)$

Similar Matrices

A, B similar $\Rightarrow$ they have the same eigenvalues: x is an eigenvector for λ for $A \Rightarrow P^{-1}x$ is eigenvector for λ for B.

$$Ax = \lambda x \Leftrightarrow \underbrace{P^{-1}AP}_{B}\underbrace{P^{-1}x}_{y} = \lambda \underbrace{P^{-1}x}_{y}$$

A, B similar: A and B represent the same linear transformation of the vector space, with respect to different bases.

If A, B are similar and B is *diagonal* or *triangular*, then the eigenvalues are the elements on $diag(B)$.

Example

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

The eigenvalue of A satisfy

$$\lambda_1 + \lambda_2 = 5, \quad \lambda_1\lambda_2 = -2$$

System of ODEs

A is diagonalizable (similar to a diagonal matrix containing its eigenvalues):

$$\exists T \in \mathbb{R}^{m\times m} \text{ invertible s.t. } T^{-1}AT = \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_m \end{pmatrix}.$$

If $A\mathbf{x}_i = \lambda_i \mathbf{x}_i, i = 1, \ldots, m$ then $T = [\mathbf{x}_1 \ldots \mathbf{x}_m]$ (invertible since its columns span $\mathbb{R}^m$).

Not all matrices are diagonalizable

All 2×2 matrices are similar to one of the three types:

$$A_1 = \begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix}, A_2 = \begin{pmatrix} a & 1 \\ 0 & a \end{pmatrix}, A_3 = \begin{pmatrix} a & -b \\ b & a \end{pmatrix}$$

System of ODEs

$$\begin{cases} \mathbf{x}' = A\mathbf{x}, & A \in \mathbb{R}^{m\times m} \\ \mathbf{x}(0) = \mathbf{x}_0 \end{cases}$$

If A is diagonalizable $(A \sim \Lambda)$, then with the change of variables $\mathbf{w} = T^{-1}\mathbf{x}$ the ODE becomes:

$$\mathbf{w}' = T^{-1}\mathbf{x}' = T^{-1}A\mathbf{x} = T^{-1}A\underbrace{TT^{-1}}_{I}\mathbf{x} = T^{-1}AT\mathbf{w} = \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_m \end{pmatrix}\mathbf{w}$$

$\Leftrightarrow w_i' = \lambda_i w_i,$ with $w_i = i^{th}$ component of $\mathbf{w}$.

Canonical Forms

Given A, we use similarity transformations to get B, “simpler” than A.
Canonical forms:

(i) The Schur normal form
(ii) The principal axes form
(iii) The singular value decomposition
(iv) The Jordan form

(i) The Schur normal form

Theorem

$A \in \mathbb{C}^{n\times n}$. $\exists U$ unitary s.t. $U^*AU = T$, with T upper triangular.

Proof: Induction on (size of matrix) n

- $n = 1$ trivial
- assume it is true $\forall$ matrices in $\mathbb{C}^{(n-1)\times(n-1)}$.

Let $\lambda_1 \in$eig(A), u_1 eigenvector for λ, $u_1^T u_1 = 1$ and pick $\{u_1, \dots, u_n\}$ an orthonormal basis for $\mathbb{C}^n$. Define

$$U_1 = [u_1, V], V = [u_2, \dots, u_n] \in \mathbb{C}^{n\times(n-1)}$$

Note: $U_1^* U_1 = U_1 U_1^* = I$ since $u_1, \dots, u_n$ are orthonormal.

$AU_1 = (Au_1, AV)$

$$\Rightarrow U_1^* A U_1 = \begin{pmatrix} u_1^* \\ V^* \end{pmatrix} (Au_1, AV) = \begin{pmatrix} u_1^* A u_1 & u_1^* AV \\ V^* A u_1 & V^* AV \end{pmatrix}$$

$$= \begin{pmatrix} \lambda_1 & u_1^* AV \\ \underbrace{\lambda_1 V^* u_1}_{=0} & V^* AV \end{pmatrix} = \begin{pmatrix} \lambda_1 & u_1^* AV \\ 0 & V^* AV \end{pmatrix}, V^* AV \in \mathbb{C}^{(n-1)\times(n-1)}.$$

(i) The Schur normal form

Induction step for V^*AV: $\exists W \in \mathbb{C}^{(n-1)\times(n-1)}$ unitary s.t.

$$\hat{T} = W^*(V^*AV)W,$$

with $\hat{T}$ upper triangular of order $n-1$.

The matrix

$$U_2 = \begin{pmatrix} 1 & 0 \\ 0 & W \end{pmatrix}$$

is unitary ($U_2^*U_2 = U_2U_2^* = I$) and satisfies:

$$U_2^*(U_1^*AU_1)U_2 = \begin{pmatrix} \lambda_1 & u_1^*AV \\ 0 & \hat{T} \end{pmatrix}$$

Then U_1U_2 is an unitary matrix that satisfies the induction. ■

The characteristic polynomial for a triangular matrix T:

$$\det(T-\lambda I) = \det \begin{pmatrix} t_{11}-\lambda & & t_{1n} \\ & \ddots & \\ 0 & & t_{nn}-\lambda \end{pmatrix} = (t_{11}-\lambda)\dots(t_{nn}-\lambda)$$

hence the diagonal elements t_{ii} are eigenvalues of T.

(i) The Schur normal form

Example

$$A = \begin{bmatrix} 3 & -2 \\ 8 & 3 \end{bmatrix}$$

$det(A - \lambda I) = \lambda^2 - 6\lambda + 25 = 0,\ \lambda_{1,2} = 3 \pm 4i$ and the corresponding eigenvectors are $\boldsymbol{v_1} = \begin{bmatrix} i \\ 2 \end{bmatrix}$ and $\boldsymbol{v_2} = \begin{bmatrix} -i \\ 2 \end{bmatrix}$.

After the Gram-Schmidt orthogonalization:

$$\boldsymbol{u_1} = \boldsymbol{v_1}, \quad \boldsymbol{u_2} = \boldsymbol{v_2} - [\boldsymbol{v_2^* u_1}/\boldsymbol{u_1^* u_1}]\boldsymbol{u_1} = \begin{bmatrix} -2 \\ -i \end{bmatrix}$$

and normalizing we have the unitary matrix

$$U = \frac{1}{\sqrt{5}} \begin{bmatrix} i & -2 \\ 2 & -i \end{bmatrix}$$

and the Schur form

$$U^* A U = \begin{bmatrix} 3+4i & -6 \\ 0 & 3-4i \end{bmatrix}$$

(ii)The principal axes form

Theorem

$A \in \mathbb{C}^{n\times n}$ Hermitian ($A^* = A$). $\exists U$ unitary s.t.

$$U^*AU = \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{pmatrix}, \quad \lambda_i \in \mathbb{R}.$$

If $A \in \mathbb{R}^{n\times n}$ and symmetric ($A^T = A$), then U is a real orthogonal matrix.

Proof: From Schur normal form, $\exists U$ unitary and T upper triangular s.t.

$$U^*AU = T \Rightarrow T^* = (U^*AU)^* = U^*A^*U \overset{A=A^*}{=\!=} T. \qquad \blacksquare$$

The columns U are orthogonal eigenvectors of A :

$$U^*AU = \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{pmatrix} \Leftrightarrow [Au_1, \ldots, Au_n] = [u_1\lambda_1, \ldots, u_n\lambda_n]$$

(ii)The principal axes form

Remark: **For a hermitian matrix all eigenvalues are real ($T^* = T$).**

Application

Assume that we know $U = [u_1, \ldots, u_n]$.
Then the basis $\{u_1, \ldots, u_n\}$ for $\mathbb{C}^n$ can be used in place of the original basis:

$$\forall y \in \mathbb{C}^n, y = \sum_{i=1}^{n} \underbrace{y_i}_{\in \mathbb{C}^n} u_i \Rightarrow Ay = \sum_{i=1}^{n} y_i A u_i = \sum_{i=1}^{n} \lambda_i y_i u_i.$$

(iii)Singular Value Decomposition (SVD)

Canonical form for general rectangular matrices $A \in \mathbb{C}^{n\times m}$

Theorem

$\exists$ unitary matrices $U \in \mathbb{C}^{m\times m}, V \in \mathbb{C}^{n\times n}$ such that

$$V^*AU = \begin{pmatrix} \mu_1 & 0 & \dots & & & & \\ 0 & \mu_2 & 0 & \dots & & & \\ \vdots & & \ddots & & & & \\ & & & & \mu_r & & \\ & & & & & 0 & \\ & & & & & & \ddots \end{pmatrix}_{n\times m} , \qquad (8.31)$$

with $\mu_i \in \mathbb{R}^n, \mu_1 \geq \mu_2 \geq \mu_{r-1} \geq \mu_r > 0$.

- μ_i's - singular values of A.
- columns of U - the right singular values.
- columns of V - the left singular values.

Remark: The SVD is not unique!

(iii)Singular Value Decomposition (SVD)

Proof: Since A^*A is hermitian $((A^*A)^* = A^*A)$, from the previous result $\exists U$ unitary s.t.

$$\underbrace{U^*A^*}_{W^*}\underbrace{AU}_{W} = \begin{pmatrix} \lambda_1 & 0 & \dots & & & & \\ 0 & \lambda_2 & 0 & \dots & & & \\ \vdots & & \ddots & & & & \\ & & & & \lambda_r & & \\ & & & & & 0 & \\ & & & & & & \ddots \end{pmatrix}, \lambda_i \neq 0. \qquad (8.32)$$

To show $\lambda_i > 0$, let x be an eigenvector of corresponding to λ_i:

$$A^*Ax = \lambda_i x \Rightarrow \underbrace{x^*A^*}_{=(Ax)^*} Ax = \lambda_i \underbrace{x^*x}_{\|x\|_2} \Rightarrow \lambda_i = \frac{\|Ax\|_2^2}{\|x\|_2^2} = \mu_i^2 > 0.$$

From (8.32): $W^*W = [w_1^*, \dots, w_n^*]^T [w_1, \dots, w_n] = (w_i^* w_j)_{i,j}$

(iii)Singular Value Decomposition (SVD)

Proof (continued): (8.32) $\Rightarrow \begin{cases} w_i^* w_j = 0, & i \neq j, \\ \|w_i\|_2^2 = \lambda_i, & i \leq r \\ \|w_i\|_2^2 = 0, & i \geq r+1. \end{cases}$

The columns $w_{r+1}, \ldots, w_n$ are zero vectors for $i \neq j \leq r, w_i \perp w_j$ in $\mathbb{C}^n$.

The columns $w_1, \ldots, w_r$ form a set of orthogonal vectors in $\mathbb{C}^n$.

The set of vectors $\{v_1, \ldots, v_r\}, v_i = \frac{w_i}{\mu_i}$, form an orthonormal set of vectors in $\mathbb{C}^n$.

Complete $\{v_1, \ldots, v_r\}$ by $\{v_{r+1}, \ldots, v_n\}$ s.t. $\{v_1, \ldots, v_n\}$ form an orthonormal basis in $\mathbb{C}^n$.

The matrix $V = [v_1, \ldots, v_n]$ is **unitary** and satisfies (8.31):

$$V \begin{pmatrix} \mu_1 & 0 & \cdots & & & \\ 0 & \mu_2 & 0 & \cdots & & \\ \vdots & & \ddots & & & \\ & & & \mu_r & & \\ & & & & 0 & \\ & & & & & \ddots \end{pmatrix} = (\underbrace{\mu_1 v_1}_{w_1}, \ldots, \underbrace{\mu_r v_r}_{w_r}, 0, \ldots, 0) = W = AU. \quad \blacksquare$$

(iii) Singular Value Decomposition (SVD)

Since U, V unitary $\Rightarrow$ rows and columns are orthonormal sets; (8.31):

$$\begin{cases} Au_1 = \mu_1 v_1 \\ Au_2 = \mu_2 v_2 \\ \vdots \\ Au_n = \mu_r v_n \end{cases} \qquad \mu_1 \geq \ldots \geq \mu_n \geq 0.$$

A transforms the coordinate system $\{u_1 \ldots, u_m\}$ to $\{\mu_1 v_1, \ldots, \mu_n v_n\}$ where $\{v_1, \ldots, v_n\}$ is another coordinate system.

Proposition

$A \in \mathbb{C}^{n\times m}$. The eigenvalues of A^*A are nonnegative.

Proof: $\|v\| = 1$ eigenvector of A^*A: $A^*Av = \lambda v \Rightarrow$
$0 \leq \|Av\| = v^*A^*Av = \lambda v^*v = \lambda \quad \blacksquare$

(iii) Singular Value Decomposition (SVD)

A^*A is $m \times m$ and unitary $\Rightarrow$ its eigenvectors are orthogonal and its eigenvalues real, nonnegative: $\mu_1^2, \dots, \mu_m^2$, and the orthonormal set of eigenvectors is $\{u_1, \dots, u_m\}$. Find $\{v_i\}_{1 \le i \le n}$:

- if $\mu_i \neq 0$, define v_i by the equation $\mu_i v_i = A u_i$
- if $\mu_i = 0$, choose v_i arbitrary unit vector s.t. orthogonal to $v_1, \dots, v_{i-1}$.

The columns of $U = [u_1, \dots, u_m]$, the **right singular vectors**, are the set of orthonormal eigenvectors of A^*A. The columns of $V = [v_1, \dots, v_n]$, the **left singular vectors**, are the set of orthonormal eigenvectors of AA^*.

Example 1: Find the singular values and singular vectors of

$$A = \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix}.$$

(iii) Singular Value Decomposition (SVD)

The eigenvalues of $A^*A = \begin{bmatrix} 0 & 0 \\ 0 & 2 \end{bmatrix}$, in decreasing size, are

$$\mu_1^2 = 2, u_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad \mu_2^2 = 0, u_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

The singular values are $\sqrt{2}$ and 0. Then v_1:

$$\sqrt{2}v_1 = Au_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \quad v_1 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix}$$

and

$$v_2 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

is chosen to be orthogonal to v_1. The SVD is:

$$\begin{bmatrix} \sqrt{2} & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

(iii) Singular Value Decomposition (SVD)

Example 2: Find the singular values and singular vectors of $A = \begin{bmatrix} 0 & -\frac{1}{2} \\ 3 & 0 \\ 0 & 0 \end{bmatrix}$.

The eigenvalues of $A^*A = \begin{bmatrix} 0 & 3 & 0 \\ -\frac{1}{2} & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & -\frac{1}{2} \\ 3 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 9 & 0 \\ 0 & \frac{1}{4} \end{bmatrix}$ are

$$\mu_1^2 = 9, u_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \qquad \mu_2^2 = \frac{1}{4}, u_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

The singular values are 3 and $\frac{1}{2}$. Then:

$$3v_1 = Au_1 = \begin{bmatrix} 0 \\ 3 \\ 0 \end{bmatrix}, v_1 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \text{ and } \frac{1}{2}v_2 = Au_2 = \begin{bmatrix} -\frac{1}{2} \\ 0 \\ 0 \end{bmatrix}, v_2 = \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix}$$

The SVD is: $\begin{bmatrix} 3 & 0 \\ 0 & \frac{1}{2} \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & -\frac{1}{2} \\ 3 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.

(iii) Singular Value Decomposition (SVD)

Symmetric matrices: To find the SVD for a symmetric $A \in \mathbb{R}^{m\times m}$, it suffices to just find the eigenvalues, arrange them in decreasing magnitude

$$|\lambda_1| \geq \ldots \geq |\lambda_m| \quad \Rightarrow \quad \mu_1 \geq \ldots \geq \mu_m$$

and eigenvectors

$$v_i = \begin{cases} +u_i, & \text{if } \lambda_i \geq 0 \\ -u_i, & \text{if } \lambda_i < 0 \end{cases}$$

Example 3: Find the singular values and singular vectors of $A = \begin{bmatrix} 0 & 1 \\ 1 & \frac{3}{2} \end{bmatrix}$

The eigenvalues/eigenvectors are:

$$\lambda_1 = 2, u_1 = \begin{bmatrix} \frac{1}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} \end{bmatrix} \qquad \lambda_2 = -\frac{1}{2}, u_2 = \begin{bmatrix} \frac{2}{\sqrt{5}} \\ -\frac{1}{\sqrt{5}} \end{bmatrix}$$

The SVD is: $$\begin{bmatrix} 2 & 0 \\ 0 & \frac{1}{2} \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \\ -\frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & \frac{3}{2} \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} & -\frac{1}{\sqrt{5}} \end{bmatrix}$$

Properties of SVD

The rank(A)= no. l.i. rows (or equivalently columns).

1. The rank of A is r, the no. of nonzero entries in $\Lambda, (V^*AU = \Lambda)$.
   Proof: Since V^* and U are invertible, rank(A) = rank(S).
2. If $A \in \mathbb{C}^{n\times n}, \det(A) = \mu_1 \cdots \mu_n$.
   Proof: V^*, U unitary $\Rightarrow \det(U), \det(V^*) = \pm 1$.
3. **Low-rank approximation of the SVD:** $A \in \mathbb{C}^{m\times n}, r =$ rank(A),

$$A = \sum_{i=1}^{r} \mu_i v_i u_i^*, \quad v_i, u_i - i^{th}\text{columns of } V, U. \qquad (8.33)$$

Proof: (Sum of rank-one matrices)

$$A = V \begin{bmatrix} \mu_1 & 0 & \dots & & & \\ 0 & \mu_2 & 0 & \dots & & \\ \vdots & & \ddots & & & \\ & & & \mu_r & & \\ & & & & 0 & \\ & & & & & \ddots \end{bmatrix} U^* = V \left(\begin{bmatrix} \mu_1 & & \\ & & \\ & & \end{bmatrix} + \dots + \begin{bmatrix} & & \\ & & \\ & & \mu_r \end{bmatrix} \right) U^*$$

The best least squares approximation of A of rank $p \leq r$: retain the first p terms of (8.33).

Properties of SVD

Example 3': Find the rank-one approximation of $A = \begin{bmatrix} 0 & 1 \\ 1 & \frac{3}{2} \end{bmatrix}$.

$$\begin{aligned}
\begin{bmatrix} 0 & 1 \\ 1 & \frac{3}{2} \end{bmatrix} &= \begin{bmatrix} \frac{1}{\sqrt{5}} & -\frac{2}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} & -\frac{1}{\sqrt{5}} \end{bmatrix} \\
&= \begin{bmatrix} \frac{1}{\sqrt{5}} & -\frac{2}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \end{bmatrix} \left(\begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & \frac{1}{2} \end{bmatrix} \right) \begin{bmatrix} \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} & -\frac{1}{\sqrt{5}} \end{bmatrix} \\
&= 2 \begin{bmatrix} \frac{1}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \end{bmatrix} + \frac{1}{2} \begin{bmatrix} -\frac{2}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} \end{bmatrix} \begin{bmatrix} \frac{2}{\sqrt{5}} & -\frac{1}{\sqrt{5}} \end{bmatrix} \\
&= \begin{bmatrix} \frac{2}{5} & \frac{4}{5} \\ \frac{4}{5} & \frac{8}{5} \end{bmatrix} + \underbrace{\begin{bmatrix} -\frac{2}{5} & \frac{1}{5} \\ \frac{1}{5} & -\frac{1}{10} \end{bmatrix}}_{\text{small corrections}}.
\end{aligned}$$

The best rank-one approximation of A is the first rank-one matrix $\begin{bmatrix} \frac{2}{5} & \frac{4}{5} \\ \frac{4}{5} & \frac{8}{5} \end{bmatrix}$.

This is the main idea behind dimension reduction and compression applications of the SVD.

Dimension reduction: project data onto a lower dimension

$a_1, \ldots, a_n$ vectors of dimension $m, m \ll n$
Goal: replace $a_1, \ldots, a_n$ with n vectors of dimension $p < m$ while minimizing the error.

$$A := [a_1, \ldots, a_n] \overset{SVD}{=} V \Lambda U^*$$

then use the rank-p approximation

$$A \approx A_p = \sum_{i=1}^{p} \mu_i v_i u_i^*$$

to project $\{a_1, \ldots, a_n\}$ onto the p-dimensional space spanned by the (left singular vectors) columns $v_1, \ldots, v_p$ of V:

$$a_j = Ae_j \approx A_p e_j \qquad \text{(columns of } A_p\text{)}$$

Dimension reduction

Example: Find the best one-dimensional subspace fitting the data vectors $\begin{bmatrix}3\\2\end{bmatrix}, \begin{bmatrix}2\\4\end{bmatrix}, \begin{bmatrix}-2\\-1\end{bmatrix}, \begin{bmatrix}-3\\-5\end{bmatrix}$.

$$A = \begin{bmatrix}3 & 2 & -2 & -3\\ 2 & 4 & -1 & -5\end{bmatrix}$$

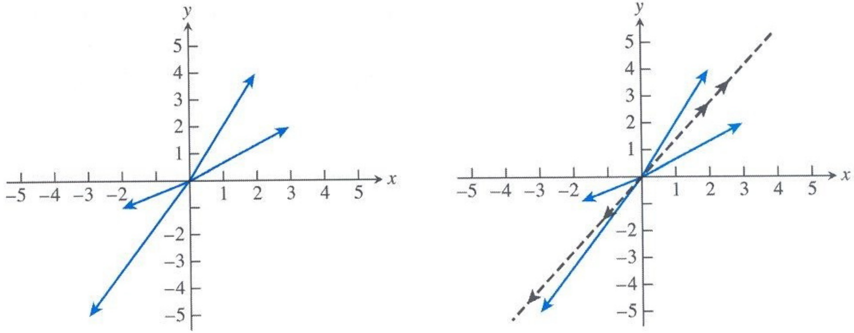
$$= \begin{bmatrix}0.5886 & -0.8084\\ 0.8084 & 0.5886\end{bmatrix}\begin{bmatrix}8.209 & 0 & 0 & 0\\ 0 & 1.8512 & 0 & 0\end{bmatrix}\begin{bmatrix}0.4085 & 0.5327 & -0.2398 & -0.7014\\ -0.6741 & 0.3985 & 0.5554 & -0.2798\\ 0.5743 & -0.1892 & 0.7924 & -0.0801\\ 0.2212 & 0.7223 & 0.0780 & 0.6507\end{bmatrix}$$

Reduce to $p = 1$ subspace: set $\mu_2 = 0$ and reconstruct the matrix:

$$A_1 = V\begin{bmatrix}8.209 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\end{bmatrix}U = \begin{bmatrix}1.9912 & 2.5964 & -1.1689 & -3.4188\\ 2.7346 & 43.5657 & -1.6052 & -4.6951\end{bmatrix}.$$

Columns of A_1: the four projected vectors corresponding to the original four data vectors

Dimension reduction



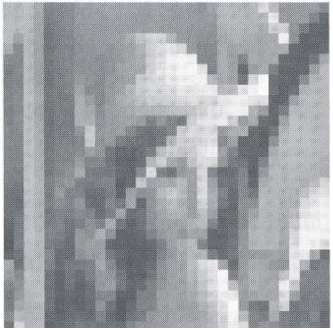
Compression

(8.33) can be used to compress the information in a matrix.
Lossy compression of A: throw away terms at the end of the sum with smaller μ_i (if A is $n \times n$, each term in the sum requires $2n+1$ numbers to store).
Example: if $n = 8$, matrix$\equiv 64$ numbers, but the 1^{st} term in (8.33) is only 17 numbers, so if $\mu_1 \gg \mu_i$'s $\Rightarrow 75\%$ saving in space.
Example: **Image Compression**
Grayscale image $=$ 256×256 array of pixels, grayness of each pixel $= 1$ byte, 8 bits representing 0(black) to 255(white). Picture: 256×256 array of integers, holding $(256)^2 = 2^{16} = 64K$ bytes of information.

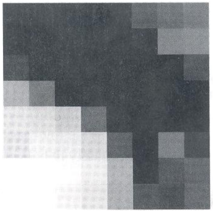
```
>> xin = imread('picture.jpg');
>> x = double(xin);
>>% imagesc(x) % displays x as an image
```

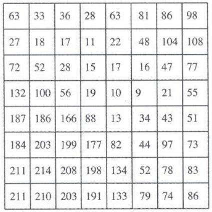
Crude method of compression: each 8×8 pixel block is replaced by its average pixel value $\Rightarrow$ considerable compression, only $(32)^2 = 2^{10}$ blocks, each represented by 1 integer, but the quality is poor.

Goal: compress less harshly, replacing each 8×8 block with a few integers that better carry the information in the original image.

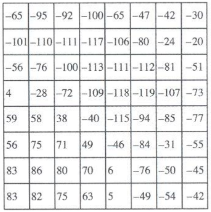


Look first at a single 8×8 block of pixels. Matrix of 1 byte integers of grayscale intensities of the 64 pixels and the matrix of intensities approximately centered around zero (subtract 256/2=128).



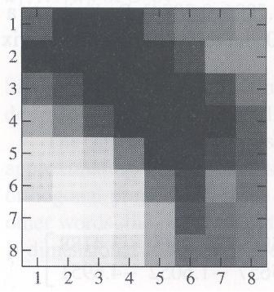


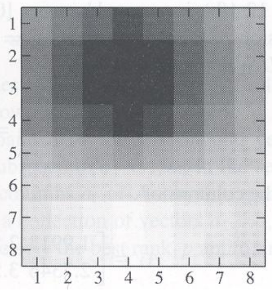
63	33	36	28	63	81	86	98
27	18	17	11	22	48	104	108
72	52	28	15	17	16	47	77
132	100	56	19	10	9	21	55
187	186	166	88	13	34	43	51
184	203	199	177	82	44	97	73
211	214	208	198	134	52	78	83
211	210	203	191	133	79	74	86

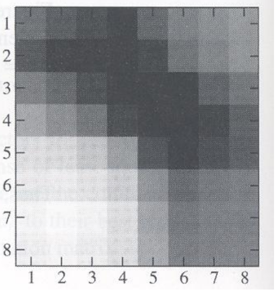


−65	−95	−92	−100	−65	−47	−42	−30
−101	−110	−111	−117	−106	−80	−24	−20
−56	−76	−100	−113	−111	−112	−81	−51
4	−28	−72	−109	−118	−119	−107	−73
59	58	38	−40	−115	−94	−85	−77
56	75	71	49	−46	−84	−31	−55
83	86	80	70	6	−76	−50	−45
83	82	75	63	5	−49	−54	−42

$$A = \begin{bmatrix} -65 & -95 & -92 & -100 & -65 & -47 & -42 & -30 \\ -101 & -110 & -111 & -117 & -106 & -80 & -24 & -20 \\ -56 & -76 & -100 & -113 & -111 & -112 & -81 & -51 \\ 4 & -28 & -72 & -109 & -118 & -119 & -107 & -73 \\ 59 & 58 & 38 & -40 & -115 & -94 & -85 & -77 \\ 56 & 75 & 71 & 49 & -46 & -84 & -31 & -55 \\ 83 & 86 & 80 & 70 & 6 & -76 & -50 & -45 \\ 83 & 82 & 75 & 63 & 5 & -49 & -54 & -42 \end{bmatrix}, \text{ Singular values } = \begin{bmatrix} 485.52 \\ 364.33 \\ 64.55 \\ 60.91 \\ 18.98 \\ 10.24 \\ 7.31 \\ 2.40 \end{bmatrix}$$





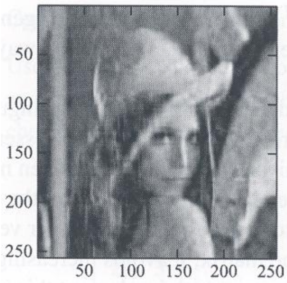


The original block

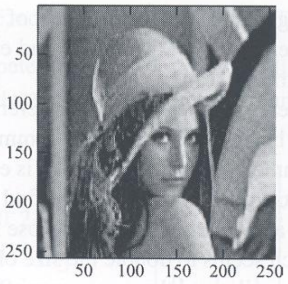
Compressed 4:1 (1^{st} term in (8.33))

Compressed 2:1 (2 terms in (8.33))

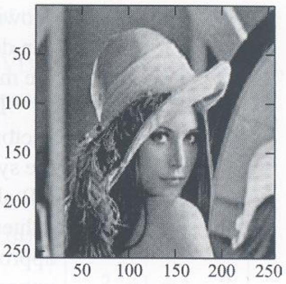
Apply (8.33) to the entire matrix. The 256 singular values $\in [2 \times 10^{-3}, 9637]$.



$p = 8$ SV



$p = 16$ SV



$p = 32$ SV

Computing the SVD

⋆ If A is real, symmetric: SVD reduces to the eigenvalue computation. U with unit eigenvectors as columns, $\Lambda = \text{diag}(\text{absolute values of eigenvalues})$, V the same as U, but with the sign of columns switched if the eigenvalue is negative.

$$A = V\Lambda U^*$$

⋆ For general $A \in \mathbb{C}^{m\times n}$ there are 2 approaches:

1. form A^*A and find its eigenvalues $\Rightarrow$ columns u_i of U and by normalizing the vectors $Au_i = \mu_i v_i$ we get both the singular values and the columns of V.
   Not recommended: if cond(A) is large $\Rightarrow$ cond$A^TA(\approx \text{cond}A)^2$ becomes very large and digits of accuracy may be lost.
2. Avoid forming the matrix product: consider $B = \begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix}$, a symmetric $(m+n)\times(m+n)$ matrix $\Rightarrow$ has real nonnegative eigenvalues and a basis of eigenvectors:
   $$\begin{bmatrix} A^Tw \\ Av \end{bmatrix} = \begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix}\begin{bmatrix} v \\ w \end{bmatrix} = \lambda\begin{bmatrix} v \\ w \end{bmatrix} \Rightarrow Av = \lambda w, A^TAv = \lambda A^Tw = \lambda^2 v.$$

(iv) The Jordan form

Jordan block

$$J_m(\lambda) = \begin{pmatrix} \lambda & 1 & 0 & \dots \\ 0 & \lambda & \ddots & \\ \vdots & \ddots & \ddots & 1 \\ 0 & \dots & 0 & \lambda \end{pmatrix}_{m\times m} .$$

λ is the unique eigenvalue of $J_m(\lambda)$ of alg.m. n and geom.m. 1.

Remark: $J_m(\lambda)$ is **nilpotent**: $J^m(0)^m = 0$.

Theorem

$A \in \mathbb{C}^n$. $\exists P$ nonsingular s.t.

$$P^{-1}AP = \begin{pmatrix} J_{n_1}(\lambda_1) & & & 0 \\ & J_{n_2}(\lambda_2) & & \\ & & \ddots & \\ 0 & & & J_{n_r}(\lambda_r) \end{pmatrix},$$

with the eigenvalues of A: $\lambda_1, \dots, \lambda_n$ not necessarily distinct.

(iv) The Jordan form

If A is hermitian: $n_1 = \ldots = n_r = 1$.
We can write the Jordan canonical form

$$P^{-1}AP = \mathrm{diag}(\lambda_1, \ldots, \lambda_r) + N$$

with N containing the rest of nonzero terms, nilpotent:

$$N^q = 0, \quad q = \max\{n_1, \ldots, n_r\} \leq n.$$

Norms

Definition

Let V be a vector space. $\|\cdot\| : V \to \mathbb{R}^+$ if

(i) $\|u\| = 0 \Leftrightarrow u = 0, \qquad u \in V$

(ii) $\|\alpha u\| = |\alpha|\|u\|, \qquad u \in V, \alpha$ scalar

(iii) $\|u + v\| \leq \|u\| + \|v\|, \qquad u, v \in V$ ("triangular inequality")

Consequence: $\Big|\|u\| - \|v\|\Big| \leq \|u - v\| \quad u, v \in V.$

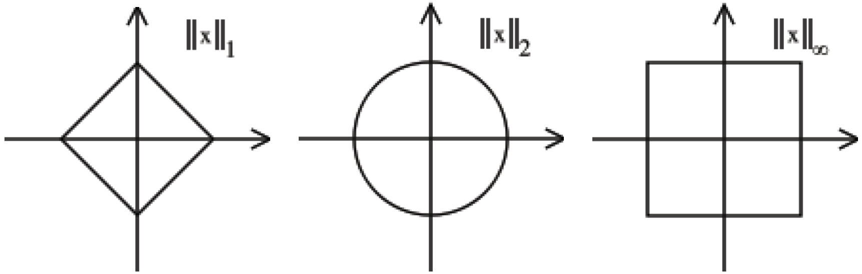
Examples: 1) Let $V = \mathbb{R}^n$, or $\mathbb{C}^n$.

p-norm, $1 \leq p < \infty$

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{\frac{1}{p}}$$

∞-norm

$$\|x\|_\infty = \max_{1<i<n} |x_i|, \qquad \left(\lim_{p\to\infty} \left(\sum_{i=1}^{n} |x_i|^p\right)^{\frac{1}{p}} = \|x\|_\infty\right)$$


$\|x\|_1$
$\|x\|_2$
$\|x\|_\infty$

2) Let $V = L^p(\Omega)$.

$$\|f\|_{L^p(\Omega)} = \left(\int_\Omega |f(x)|^p \, dx\right)^{\frac{1}{p}}, \qquad \|f\|_{L^\infty(\Omega)} = \sup_{x\in\Omega} |f(x)|.$$

Theorem

In finite dimensional spaces all norms are equivalent: $\forall N_1, N_2$ *norms* $\exists c_1, c_2 > 0$ *s.t.*

$$c_1 N_2(x) \le N_1(x) \le c_2 N_2(x), \quad \forall x \in V. \tag{8.34}$$

Corrollary

Convergence is equivalent in every norms $(x^{(n)} \xrightarrow{n\to\infty} x)$:

$$c_1 N_2(x - x^{(n)}) \le N_1(x - x^{(n)}) \le c_2 N_2(x - x^{(n)})$$

Proof: Let $N_2 = \|\cdot\|$. Then (8.34) is equivalent to boundedness of N_1

$$c_1 \leq N_1(y) \leq c_2, \quad \forall y \in V, \|y\|_\infty = 1.$$

The upper bound:

$$N_1(x) = N\left(\sum_{i=1}^n x_i e^i\right) \leq \sum_{i=1}^n |x_i| N_1(e^i) \leq \underbrace{c_2}_{\sum_{i=1}^n N_1(e^i)} \|x\|_\infty.$$

and continuity of N_1:

$$|N_1(x) - N_1(y)| \leq N_1(x-y) \leq c_2 \|x-y\|_\infty.$$

Hence N_1 reaches its minimum on the unit sphere $S = \{y \in V; \|y\|_\infty = 1\}$, closed and bounded set in V:

$$\min_{y\in S} N_1(y) = c_1 \neq 0. \qquad \blacksquare$$

Matrix norms

$m \times n$ matrices can be viewed as vectors of mn dimension $\Rightarrow$ matrix norm has to be first a vector norm:

(i) $\|A\| = 0 \Leftrightarrow A = 0$

(ii) $\|\alpha A\| = |\alpha|\|u\|, \qquad A \in \mathscr{M} \in \mathbb{C}^{n\times n}, \alpha$ scalar

(iii) $\|A+B\| \le \|A\| + \|B\|, \qquad A, B \in \mathscr{M}$ Matrices can be multiplied with matrices and vectors

(iv) $\|AB\| \le \|A\|\|B\|, \qquad A, B \in \mathscr{M}$

"Compatibility" between matrix norm and vector norm:

$$\|Ax\|_{\text{vector norm}} \le \|A\|\|x\|_{\text{vector norm}}, \qquad x \in V, A \in \mathscr{M} \tag{8.35}$$

Definition: Start with the vector norm and maintain "compatibility"

$$\|A\| := \sup_{x\neq 0} \frac{\|Ax\|}{\|x\|} = \sup_{x\neq 0} \left\|\frac{Ax}{\|x\|}\right\| = \sup_{x\neq 0} \left\|A\left(\frac{x}{\|x\|}\right)\right\| = \sup_{y, \|y\|=1} \|Ay\|$$

It is a norm and satisfies (8.35).

$$\|AB\| \leq \|A\|\|B\|$$

Proof: $\|AB\| := \sup\limits_{\|x\|=1} \|ABx\|$ and pick $y, \|y\|=1$ s.t. $\|AB\| = \|ABy\|$.

$$\|A(By)\| \leq \|A\|\|By\| \leq \|A\|\|B\|\|y\| = \|A\|\|B\|. \qquad \blacksquare$$

Example: max row sum

$$\|A\|_\infty := \sup_{\|y\|_\infty=1} \|Ay\|_\infty := \sup_{\|y\|_\infty=1} \max_{1\leq i\leq n} \left|\sum a_{ij}y_j\right|$$

Start with the ∞-vector norm: $\|x\|_\infty = \max_{1\leq i\leq n} |x_i|, \quad x \in \mathbb{R}^n$, then

$$A\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^n a_{1j}x_j \\ \vdots \\ \sum_{j=1}^n a_{nj}x_j \end{bmatrix} \Rightarrow \begin{aligned} \|Ax\|_\infty &= \max_{1\leq i\leq n} \left|\sum_{j=1}^n a_{ij}x_j\right| \\ &\leq \underbrace{\max_{1\leq i\leq n} \sum_{j=1}^n |a_{ij}|}_{:=\|A\|_\infty} \underbrace{|x_j|}_{\leq \|x\|_\infty} \end{aligned}$$

Max row sum norm

Proposition

$$\|A\|_\infty = \max_{1\le i\le n} \sum_{j=1}^n |a_{ij}|$$

Proof: **1.** "$\le$": $\exists x \in \mathbb{C}^n$ s.t. $\|A\|_\infty = \|Ax\|_\infty, \|x\|_\infty = 1.$

$$\Rightarrow \|A\|_\infty = \|Ax\|_\infty = \max_{1\le i\le n} \sum_{j=1}^n |a_{ij}x_j| \le \max_{1\le i\le n} \sum_{j=1}^n |a_{ij}| \underbrace{|x_j|}_{\le 1}$$

2. "$\ge$" Let i_0 the index for which: $\max_{1\le i\le n} \sum_{j=1}^n |a_{ij}| = \sum_{j=1}^n |a_{i_0 j}|$

and construct $x \in \mathbb{C}^n, x_j = \begin{cases} \frac{\overline{a}_{i_0 j}}{|a_{i_0 j}|} & \text{if } a_{i_0 j} \ne 0 \\ 0 & \text{if } a_{i_0 j} = 0 \end{cases} \Rightarrow \|x\|_\infty = 1.$

Max row sum norm

With this x:

$$\|Ax\|_\infty = \max_{1\le i\le n}\left|\sum_{j=1}^n a_{ij}x_j\right| = \max\left\{\left|\sum_{j=1}^n a_{i_0j}x_j\right|, \max_{i\neq i_0}\left|\sum_{j=1}^n a_{ij}x_j\right|\right\}$$

$$= \max\left\{\left|\sum_{j=1}^n a_{i_0j}\frac{\overline{a}_{i_0j}}{|a_{i_0j}|}\right|, \max_{i\neq i_0}\left|\sum_{j=1}^n a_{ij}x_j\right|\right\}$$

$$= \max\left\{\sum_{j=1}^n |a_{i_0j}|, \max_{i\neq i_0}\underbrace{\left|\sum_{j=1}^n a_{ij}x_j\right|}_{\le\sum_{j=1}^n |a_{i_0j}|}\right\} = \sum_{j=1}^n |a_{i_0j}|$$

$$\Rightarrow \qquad \textstyle\sum_{j=1}^n |a_{i_0j}| = \|Ax\|_\infty \le \|A\|_\infty \underbrace{\|x\|_\infty}_{\le 1} \qquad \blacksquare$$

Max column sum norm

Example: max column sum

$$\|A\|_1 := \sup_{\|x\|_1=1} \|Ax\|_1 := \sup_{\|x\|_1=1} \left\| \begin{bmatrix} \sum_{j=1}^{n} a_{1j}x_j \\ \vdots \\ \sum_{j=1}^{n} a_{nj}x_j \end{bmatrix} \right\|_1 = \sup_{\|x\|_1=1} \sum_{i=1}^{n} \left| \sum_{j=1}^{n} a_{ij}x_j \right|$$

Proposition

$$\|A\|_1 = \max_{1\le j\le n} \sum_{i=1}^{n} |a_{ij}|$$

Spectrum, spectral radius

Definition

1. spectrum of A: $\sigma(A) = \{\lambda : \lambda \text{ eigenvalue of } A\}$
2. spectral radius of A: $\rho(A) = \max_{\lambda\in\sigma(A)} |\lambda|$

Lemma

$$\|A\|_2 = \sqrt{\rho(A^*A)}$$

Proof: 1) A^*A hermitian $\Rightarrow \lambda \in \sigma(A), \lambda \in \mathbb{R}, \lambda \geq 0$. Let $\{u_1, \ldots, u_n\}$ an orthonormal basis in $\mathbb{C}^n$ s.t. $A^*Au_i = \lambda_i u_i, x \in \mathbb{C}^n$: $x = \sum_{i=1}^n \underbrace{x_i}_{\in\,\mathbb{C}^n} u_i$.

$$\|Ax\|_2^2 = (Ax)^*Ax = x^*A^*Ax = x^*\left(\sum_{i=1}^n x_i A^*Au_i\right) = x^*\left(\sum_{i=1}^n x_i\lambda_i u_i\right)$$
$$= \sum_{i=1}^n x_i\lambda_i \underbrace{x^*u_i}_{\overline{x_i}} = \sum_{i=1}^n \lambda_i |x_i|^2 \leq \max_{1\leq i\leq n} |\lambda_i|^2 \left(\sum_{i=1}^n |x_i|^2\right) \leq \rho(A^*A)\|x\|_2^2.$$

2) $\lambda =$the max. of the eigenvalues of A^*A, u, $\|u\|_2 = 1$ the corresponding eigenvector:

$$\|Au\|_2^2 = u^*A^*Au = \lambda u^*u = \lambda\|u\|_2^2 \Rightarrow \sqrt{\lambda} \leq \|A\|_2. \qquad \blacksquare$$

Lemma

$\rho(A) \leq \|A\|$ *for any induced matrix norm.*

<u>Proof</u>: Let $\lambda \in \sigma(A)$, u an associated eigenvector: $Au = \lambda u$, $\|u\| = 1$.

$$\|A\| = \max_{\|x\|=1} \|Ax\| \geq \|Au\| = \|\lambda u\| = |\lambda|. \qquad \blacksquare$$

Lemma

$\forall \varepsilon > 0 \ \exists$ *a matrix induced norm such that* $\|A\| \leq \rho(A) + \varepsilon$.

Consequence

$\forall \varepsilon > 0 \ \exists \| \cdot \|$ such that $\rho(A) \leq \|A\| \leq \rho(A) + \varepsilon$.

Recall: for $\{x_n\}_n \subset \mathbb{C}$

$$x_n \to x \qquad \Leftrightarrow \qquad \lim_{n\to\infty} \|x_n - x\| = 0$$

Definition

$A \in \mathbb{C}^{n\times n}$ is convergent if $\lim\limits_{n\to\infty} A^n = 0$.

Theorem

The following are equivalent:

(i) A is convergent.

(ii) $\lim_{n\to\infty} \|A^n\| = 0$ for some $\|\cdot\|$.

(iii) $\rho(A) < 1$.

Proof: By the continuity of $\|\cdot\|$ we have $(i) \Leftrightarrow (ii)$. Now $(ii) \Rightarrow (iii)$:

$$\underbrace{\rho(A^n)}_{\substack{=(\rho(A))^n \\ \lambda\in\sigma(A)\Leftrightarrow\lambda\in\sigma(A^n) \\ Ax=\lambda x \Leftrightarrow A^n x = \lambda^n x}} \le \|\underbrace{A^n}_{}\| \implies \rho(A) < 1$$

$$\downarrow_{n\to\infty}$$

$$0$$

$(iii) \Rightarrow (ii)$: Assuming $\rho(A) < 1$, we have $\varepsilon := \frac{1}{2}(1 - \rho(A)) > 0$. Then by the previous lemma, $\exists \|\cdot\|$ such that

$$\|A\| \leq \rho(A) + \varepsilon = \frac{1}{2} \underbrace{\rho(A)}_{<1} + \frac{1}{2} < 1 \Rightarrow \lim_{n\to\infty} \underbrace{\|A\|^n}_{\geq \|A^n\|} = 0. \quad \blacksquare$$

Theorem

(a) the series $\sum_{k=0}^{\infty} A^k$ is convergent $\Leftrightarrow$ A is convergent.

(b) A is convergent $\Rightarrow$ $(I - A)$ is invertible and $(I - A)^{-1} = \sum_{k=0}^{\infty} A^k$.

Proof: (a) $' \Rightarrow '$: the partial sum $B_n = \sum_{k=0}^{n} A^k$ is convergent, hence Cauchy:

$$0 = \lim_{n\to\infty} \|B_{n+1} - B_n\| = \lim_{n\to\infty} \|A^{n+1}\|.$$

(b) '$\Rightarrow$': $\rho(A) < 1 \Rightarrow 1 \notin \sigma(A) \Rightarrow (I - A)$ invertible.

(Otherwise $\exists x \neq 0 : (I - 1A)x = 0$).

$$\underbrace{(I - A)\sum_{k=0}^{n} A^k}_{\substack{\downarrow \\ (I-A)\sum^{\infty} A^k}} = \sum_{k=0}^{n} A^k - \sum_{k=0}^{n} A^{k+1} = I - A^{n+1} \xrightarrow{n\to\infty} I. \qquad \blacksquare$$

Lemma

If $\|A\| < 1$ for some $\|\cdot\|$, then $I - A$ is invertible and
$$\frac{1}{1+\|A\|} \le \|(I-A)^{-1}\| \le \frac{1}{1-\|A\|}.$$

Proof: $\rho(A) \le \|A\| < 1 \Rightarrow 1 \notin \sigma(A) \Rightarrow \exists (I - A)^{-1}$.

Recall $\|I\| = \sup_{\|x\|=1} \|Ix\| = \sup_{\|x\|=1} \|x\| = 1$. Then

$$(\star)\ 1 = \|I\| = \|(I-A)(I-A)^{-1}\| \le (\|I\| + \|A\|)\|(I-A)^{-1}\|$$

$$I = (I-A)(I-A)^{-1} = (I-A)^{-1} - A(I-A)^{-1}$$

$$\Rightarrow \|(I-A)^{-1}\| = \|I + A(I-A)^{-1}\| \le \|I\| + \|A\|\|(I-A)^{-1}\| \Rightarrow \underbrace{(1-\|A\|)}_{>0}\|(I-A)^{-1}\| \le 1$$

Theorem

Let $A, B \in \mathscr{M}_n$, A is nonsingular and

$$\|A - B\| < \frac{1}{\|A^{-1}\|}.$$

Then B is also nonsingular and

$$\|B^{-1}\| \leq \frac{\|A^{-1}\|}{1-\|A^{-1}\|\|A-B\|},$$
$$\|A^{-1} - B^{-1}\| \leq \frac{\|A^{-1}\|^2\|A-B\|}{1-\|A^{-1}\|\|A-B\|}.$$

Proof: Since $B = A - (A - B) = A[I - A^{-1}(A - B)]$, by Lemma $I - A^{-1}(A - B)$ is nonsingular $\Rightarrow B$ is nonsingular, as product of nonsingular matrices:

$$B^{-1} = [I - A^{-1}(A - B)]^{-1}A^{-1}.$$

The first estimate follows from Lemma, while for second we take norms in

$$A^{-1} - B^{-1} = A^{-1}(B - A)B^{-1}$$

and appply the 1st estimate. ■

Properties of Eigenvalues

Eigenvalues have many interesting and important mathematical properties. These include the following.

1. If A is diagonal, upper triangular or lower triangular, then the eigenvalues are on the diagonal of A.

Proof.

Let A be upper triangular. Then

$$\det\left[A-\lambda I\right] = \det\begin{bmatrix} a_{11}-\lambda & a_{12} & \dots & a_{1n} \\ & a_{22}-\lambda & \dots & a_{2n} \\ & & \ddots & \vdots \\ & & & a_{nn}-\lambda \end{bmatrix}$$
$$= (a_{11}-\lambda)(a_{22}-\lambda)\cdot\ldots\cdot(a_{nn}-\lambda) = p_n(\lambda).$$

The roots of p_n are obvious! □

Properties of Eigenvalues

2. Suppose A is $N \times N$ and $\lambda_1, \ldots, \lambda_k$ are <u>distinct</u> eigenvalues of A. Then,

$$\det(A - \lambda I) = (\lambda_1 - \lambda)^{p_1}(\lambda_2 - \lambda)^{p_2} \cdot \ldots \cdot (\lambda_k - \lambda)^{p_k}$$

1. where $p_1 + p_2 + \ldots + p_k = n$.
2. Each λ_j has at least one eigenvector Φ_j and possibly as many as p_k linearly independent eigenvectors.
3. If each λ_j has p_j linearly independent eigenvectors then all the eigenvectors form a basis for $\mathbb{R}^N$.
4. Anything between (2) and (3) can occur.

Properties of Eigenvalues

3. If A is **symmetric** (and real) $(A = A^t)$, then
   1. all the eigenvalues and eigenvectors are <u>real</u>.
   2. There exists N orthonormal eigenvectors $\Phi_1, \dots, \Phi_N$ of A:
$$\langle \Phi_i, \Phi_j \rangle = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j. \end{cases}$$
   3. If C is the $N \times N$ matrix with eigenvector Φ_j in the j^{th} column then
$$C^{-1} = C^t \qquad \text{and} \qquad C^{-1}AC = \begin{pmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_N \end{pmatrix}.$$

Properties of Eigenvalues

4. If an eigenvector Φ is known,
   the corresponding eigenvalue is given by
   the **Rayleigh quotient**

$$\lambda = \frac{\Phi^* A \Phi}{\Phi^* \Phi}, \qquad \text{where } \Phi^* = \overline{\Phi}^t.$$

Proof.

If $A\Phi = \lambda\Phi$, we have

$$\Phi^* A \Phi = \lambda \Phi^* \Phi$$

from which the formula for λ follows. □

Properties of Eigenvalues

5. If $\|\cdot\|$ is any matrix norm (induced by the vector norm $\|\cdot\|$),

$$|\lambda| \leq \|A\|.$$

Proof.

Since

$$\lambda\Phi = A\Phi$$

we have

$$|\lambda|\|\Phi\| = \|\lambda\Phi\| = \|A\Phi\| \leq \|A\|\|\Phi\|.$$ □

Finally, we note that: *the eigenvalues of A are complicated,* **nonlinear functions** *of the entries in A.*

Thus, the eigenvalues of $A + B$ can have no correlation with those of A and B. In particular,

$$\lambda(A + B) \neq \lambda(A) + \lambda(B).$$

Properties of Eigenvalues

Eigenvalues are very important yet they are complicated, **nonlinear functions** of the entries of A.
Thus, results that allow us to look at the entries of A and get information about where they live (known as *eigenvalue localization* results) are useful results indeed.
We have seen two such already:

- If $A = A^t$ then $\lambda(A)$ is real.
- $|\lambda| \leq \|A\|$ for any norm $\|\cdot\|$.

The classic such result is Gershgorin's theorem.

Gershgorin's circle theorem

Theorem (The Gershgorin Circle Theorem)

Let $A = (a_{ij}), i, j = 1, \dots, N$. Define the row and column sums which exclude the diagonal entry.

$$r_k = \sum_{j=1, j\neq k}^{N} |a_{kj}|, \qquad c_k = \sum_{j=1, j\neq k}^{N} |a_{jk}|.$$

Define the closed disks in $\mathbb{C}$:

$$R_k = \{z \in \mathbb{C};\ |z - a_{kk}| \leq r_k\}, \qquad C_k = \{z \in \mathbb{C};\ |z - a_{kk}| \leq c_k\}.$$

Then, if λ is an eigenvalue of A

(i) $\lambda \in R_k$ *for some* k.

(ii) $\lambda \in C_k$ *for some* k.

(iii) *If Ω is a union of precisely k disks that is disjoint from all other disks then Ω must contain k eigenvalues of A.*

Gershgorin's circle theorem

Example

$$A_{3\times 3} = \begin{bmatrix} 1 & 2 & -1 \\ 2 & 7 & 0 \\ -1 & 0 & 5 \end{bmatrix}, \quad eig(A) = \{0.2030, 5.1580, 7.6390\}.$$

We calculate

$$\begin{array}{lll} r_1 = 2+1 = 3, & r_2 = 2+0 = 2, & r_3 = 1+0 = 1. \\ c_1 = 2+1 = 3, & c_2 = 2+0 = 2, & c_k = 1+0 = 1. \end{array}$$

The Eigenvalues must belong to the above 3 circles.
Since $A = A^t$, they must also be real. Thus

$$-2 \leq \lambda \leq 8.$$

Gershgorin's circle theorem

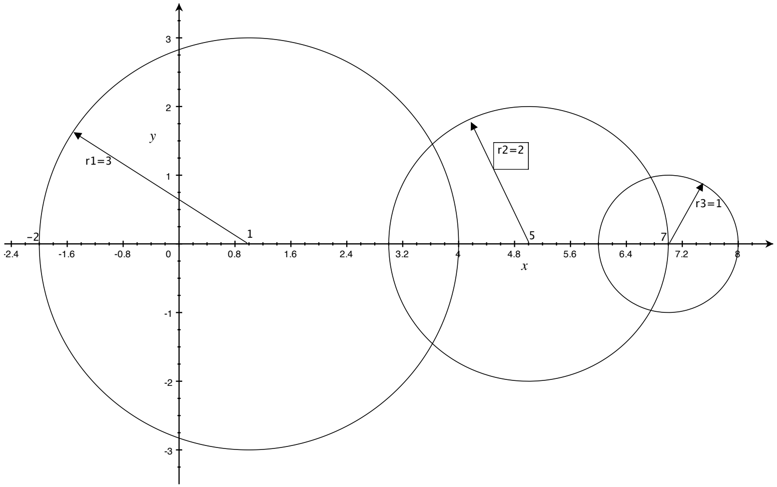


Figure: Gershgorin's Circles

Gershgorin's circle theorem

Gershgorin's circle theorem

Let define the circles

$$Z_i = \{z \in \mathbb{C} : |z - a_{ii}| \le r_i\}, i = 1, \dots, n$$

of radii

$$r_i = \sum_{j=1, j\neq i}^{n} |a_{ij}|, i = 1, \dots, n.$$

The eigenvalues of A are located in the union of circles Z_i:

$$\lambda \in \sigma(A) \quad \Rightarrow \quad \lambda \in \cup_{i=1}^{n} Z_i$$

Proof: Let $A\boldsymbol{x} = \lambda \boldsymbol{x}, \boldsymbol{x} \neq \mathbf{0}$ and $k \in \{1, \dots, n\}$ such that $\|\boldsymbol{x}\|_\infty = |x_k| > 0$. Then the kth line of $A\boldsymbol{x} = \lambda\boldsymbol{x}$ gives

$$|\lambda - a_{kk}||x_k| \le \sum_{j=1, j\neq k}^{n} |a_{kj}||x_j| \le \sum_{j=1, j\neq k}^{n} |a_{kj}||x_{k}| \qquad \blacksquare$$

Gershgorin's circle theorem

Example

$$\begin{pmatrix} 1 & 2 & 3 \\ 3 & 4 & 9 \\ 1 & 1 & 1 \end{pmatrix}, \qquad \begin{aligned} Z_1 &= \{z : |z-1| \le 5\} \\ Z_2 &= \{z : |z-4| \le 12\} \\ Z_3 &= \{z : |z-1| \le 2\} \end{aligned}$$

Other evaluating bounds:

$$|\lambda| \le \rho(A) = \max_{\lambda(A)\in\sigma(A)} |\lambda| \le \|A\| \qquad \Rightarrow |\lambda| \le \min\{\underbrace{\max_i \sum_{j=1}^n |a_{ij}|}_{\|\cdot\|_\infty}, \underbrace{\max_j \sum_{i=1}^n |a_{ij}|}_{\|\cdot\|_1}\}$$

$$|\lambda| \le \min\left\{\max\{6,16,3\}, \max\{5,7,13\}\right\} = 13.$$

Claim:

$$\sum_{i=1}^n |\lambda_i|^2 \le \sum_{i,j=1}^n |a_{ij}|^2.$$

Example: $|\lambda_1|^2 + \lambda_2|^2 + \lambda_3|^2 \le 1+4+9+9+16+81+1+1+1.$

Eigenvalue sensitivity

$$\begin{cases} Ax = \lambda x \\ (A+E)\hat{x} = \hat{\lambda}\hat{x} \end{cases}, \qquad |\lambda - \hat{\lambda}|?$$

Bauer-Fike Theorem

Assume A is diagonalizable $(P^{-1}AP = D)$. If λ is an eigenvalue of $A+E$, then

$$\min_{1 \le i \le n} |\lambda_i - \lambda| \le \|P\|\|P^{-1}\|\|E\|$$

where $\sigma(A) = \{\lambda_1, \dots, \lambda_n\}$.

Proof: Assume $\lambda \neq \lambda_i \Rightarrow (\lambda I - D)$ is invertible.

$$(A+E)x = \lambda x$$
$$Ex = (\lambda I - A)x = (\lambda I - PDP^{-1})x = P(\lambda I - D)P^{-1}x$$
$$P^{-1}Ex = (\lambda I - D)\underbrace{P^{-1}x}_{=y}$$
$$P^{-1}EPy = (\lambda I - D)y \quad \Rightarrow \quad (\lambda I - D)^{-1}P^{-1}EPy = y$$
$$\|y\| = \|(\lambda I - D)^{-1}P^{-1}EPy\| \le \|(\lambda I - D)^{-1}\|\|P^{-1}\|\|P\|\|E\|\|y\|$$

Bauer-Fike theorem

$x \neq 0 \Rightarrow y \neq 0 \Rightarrow \|y\| \neq 0.$
Assume we are using a matrix norm for which

$$\mathscr{D} = \max_{1\le i\le n} |\lambda_i|$$

$$1 \le \|(\lambda I - D)^{-1}\| \|P^{-1}\| \|P\| \|E\| \quad \text{and}$$

$$\|(\lambda I - D)^{-1}\| = \max_{1\le i\le n} \frac{1}{|\lambda - \lambda_i|} = \frac{1}{\min_i |\lambda - \lambda_i|}$$

$$\Rightarrow \min_i |\lambda - \lambda_i| \le \|P^{-1}\| \|P\| \|E\|. \qquad \blacksquare$$

Example

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 0 & .999 & 1 \\ 0 & 0 & 2 \end{pmatrix}, \quad E = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 10^{-5} & 0 & 2 \end{pmatrix}.$$

cond$(P) \approx 10^3$,
eig$(A) = \{1.0000, 0.9990, 2.0000\}$,
eig$(A+E) = \{2.0000, 0.9995 + 0.0044i, 0.9995 - 0.0044i\}$

If A is symmetric and real, then P can be chosen orthogonal $P^{-1} = P^T$, $\|P\|_2 = 1$. Hence

$$\min_i |\lambda - \lambda_i| \leq \|P^{-1}EP\|_2 \leq \|E\|_2 = \sqrt{\rho(E^T E)}$$

i.e., small changes in the matrix $\Rightarrow$ small changes in the eigenvalues.

Example in the nonsymmetric eigenvalue problem

$$A = \begin{pmatrix} 101 & -90 \\ 110 & -98 \end{pmatrix}, \quad A + E = \begin{pmatrix} 100.99 & -90.001 \\ 110 & -98 \end{pmatrix}$$

$$\sigma(A) = \{1, 2\}, \sigma(A + E) \approx \{1.298, 1.701\}$$

which is a significant change in eigenvalues for a very small change in the matrix.

Sensitivity of individual eigenvalues

$$\lambda_i \in \sigma(A),\ P^{-1}AP = \operatorname{diag}(\lambda_1, \ldots, \lambda_n), \qquad A = PDP^{-1}$$
$$\Rightarrow P^{-1}APe_i = \lambda_i e_i, APe_i = \lambda_i Pe_i,$$
$Au_i = \lambda_i u_i$, **right eigenvector:** $u_i = \dfrac{Pe_i}{\|Pe_i\|_2}$.
$$\Rightarrow P^*A^*(P^{-1})^*e_i = \overline{\lambda_i}e_i, A^*(P^{-1})^*e_i = \overline{\lambda_i}(P^*)^{-1}e_i,$$
$A^*v_i = \overline{\lambda_i}v_i$, or $v_i^*A = \lambda_i v_i^*$, **left eigenvector:** $v_i = \dfrac{(P^{-1})^*e_i}{\|(P^{-1})^*e_i\|_2}$.

Definition

The sensitivity of λ_i is $\frac{1}{s_i}$ with $s_i = v_i^*u_i$.

$$\frac{1}{s_i} \leq \operatorname{cond}_2(P).$$

Proof:

$$s_i = v_i^*u_i = \frac{Pe_i}{\|Pe_i\|_2}\frac{(P^{-1})^*e_i}{\|(P^{-1})^*e_i\|_2} = \frac{e_i^*P^{-1}Pe_i}{\|Pe_i\|_2\|(P^{-1})^*e_i\|_2} = \frac{1}{\|Pe_i\|_2\|(P^{-1})^*e_i\|_2}. \blacksquare$$

Sensitivity of individual eigenvalues

Application to a particular perturbed eigenvalue problem:

$$(A + \varepsilon B)x = \lambda(\varepsilon)x,$$

with $\varepsilon > 0, \lambda_k \in \sigma(A)$. Then

$$\lambda_k(\varepsilon) = \lambda_k + \frac{\varepsilon}{s_k} v_k^* B u_k + O(\varepsilon^2).$$

Example

$$A = \begin{pmatrix} 101 & -90 \\ 110 & -98 \end{pmatrix}, \quad P^{-1}AP = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$$

$$P = \begin{pmatrix} \frac{9}{\sqrt{181}} & \frac{-10}{\sqrt{221}} \\ \frac{10}{\sqrt{181}} & \frac{-11}{\sqrt{221}} \end{pmatrix}, \quad P^{-1} = \begin{pmatrix} -11\sqrt{181} & 10\sqrt{181} \\ -10\sqrt{221} & 9\sqrt{221} \end{pmatrix}$$

$$s_1 = v_1^* u_1 = \frac{1}{\sqrt{221 \cdot 181}} \approx .005$$

$$\Rightarrow |\lambda_1(\varepsilon) - \lambda_1| \leq \frac{\varepsilon}{s_1} \|B\|_2 + O(\varepsilon^2) \approx 200\varepsilon \|B\|_2 + O(\varepsilon)$$

The power method

The **power method** is an example of an iterative process that under right circumstances produces as sequence converging to an eigenvalue of a given matrix.
Suppose that $\boldsymbol{A} \in \mathcal{M}(n)$ and its eigenvalues satisfy

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots |\lambda_n|$$

Each eigenvalue has a nonzero eigenvector $\boldsymbol{u}^{(i)}$ and

$$\boldsymbol{A}\boldsymbol{u}^{(i)} = \lambda_i \boldsymbol{u}^{(i)}, \qquad i = 1 : n$$

and assume that $\{\boldsymbol{u}^{(1)}, \boldsymbol{u}^{(2)}, \ldots, \boldsymbol{u}^{(n)}\}$ are linearly independent (form a basis for $\mathbb{C}^n$).

Goal: compute the **dominant** eigenvalue (*single* eigenvalue) of maximum modulus and an associated eigenvector.

The power method

With a starting vector $\boldsymbol{x}^{(0)} \in \mathbb{C}^n$

$$\boldsymbol{x}^{(0)} = c_1\boldsymbol{u}^{(1)} + c_2\boldsymbol{u}^{(2)} + \cdots + c_n\boldsymbol{u}^{(n)}$$

with $c_1 \neq 0$ or simply

$$\boldsymbol{x}^{(0)} = \boldsymbol{u}^{(1)} + \boldsymbol{u}^{(2)} + \cdots + \boldsymbol{u}^{(n)}$$

we produce the sequence

$$\begin{aligned}
\boldsymbol{x}^{(1)} &= \boldsymbol{A}\boldsymbol{x}^{(0)} \\
\boldsymbol{x}^{(2)} &= \boldsymbol{A}\boldsymbol{x}^{(1)} = \boldsymbol{A}^{\boldsymbol{2}}\boldsymbol{x}^{(0)} \\
\boldsymbol{x}^{(3)} &= \boldsymbol{A}\boldsymbol{x}^{(2)} = \boldsymbol{A}^{\boldsymbol{3}}\boldsymbol{x}^{(0)} \\
&\vdots \\
\boldsymbol{x}^{(k)} &= \boldsymbol{A}\boldsymbol{x}^{(k-1)} = \boldsymbol{A}^{\boldsymbol{k}}\boldsymbol{x}^{(0)} \\
&\vdots
\end{aligned}$$

The power method

Using the form of $\boldsymbol{x}^{(0)}$ we have

$$\begin{aligned}\boldsymbol{x}^{(k)} &= \boldsymbol{A}^{\boldsymbol{k}}\boldsymbol{x}^{(0)} \\ &= \boldsymbol{A}^{\boldsymbol{k}}\boldsymbol{u}^{(1)} + \boldsymbol{A}^{\boldsymbol{k}}\boldsymbol{u}^{(2)} + \boldsymbol{A}^{\boldsymbol{k}}\boldsymbol{u}^{(3)} + \cdots + \boldsymbol{A}^{\boldsymbol{k}}\boldsymbol{u}^{(n)} \\ &= \lambda_1^k\boldsymbol{u}^{(1)} + \lambda_2^k\boldsymbol{u}^{(2)} + \lambda_3^k\boldsymbol{u}^{(3)} + \cdots + \lambda_n^k\boldsymbol{u}^{(n)}\end{aligned}$$

and equivalently

$$\boldsymbol{x}^{(k)} = \lambda_1^k\left[\boldsymbol{u}^{(1)} + \underbrace{\underbrace{\left(\frac{\lambda_2}{\lambda_1}\right)^k}_{\to 0}\boldsymbol{u}^{(2)} + \underbrace{\left(\frac{\lambda_3}{\lambda_1}\right)^k}_{\to 0}\boldsymbol{u}^{(3)} + \cdots + \underbrace{\left(\frac{\lambda_n}{\lambda_1}\right)^k}_{\to 0}\boldsymbol{u}^{(n)}}_{=\varepsilon^{(k)}\to 0}\right]$$

The power method

Hence we have

$$\boldsymbol{x}^{(k)} = \lambda_1^k \left[\boldsymbol{u}^{(1)} + \boldsymbol{\varepsilon}^{(k)}\right]$$

and for any linear functional φ

$$\varphi(\boldsymbol{x}^{(k)}) = \lambda_1^k \left[\varphi(\boldsymbol{u}^{(1)}) + \varphi(\boldsymbol{\varepsilon}^{(k)})\right]$$

We can compute the **dominant** eigenvalue λ_1 as the limit of

$$r_k = \frac{\varphi(\boldsymbol{x}^{(k+1)})}{\varphi(\boldsymbol{x}^{(k)})} = \lambda_1 \left[\frac{\varphi(\boldsymbol{u}^{(1)}) + \varphi(\boldsymbol{\varepsilon}^{(k+1)})}{\varphi(\boldsymbol{u}^{(1)}) + \varphi(\boldsymbol{\varepsilon}^{(k)})}\right] \to \lambda_1$$

The power method: Example

Let $A = \begin{bmatrix} 1 & 3 \\ 2 & 2 \end{bmatrix}$ with $\lambda_1 = 4, x_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \lambda_2 = -1, x_2 = \begin{bmatrix} -3 \\ 2 \end{bmatrix}$.

Multiplying A to a random vector $z^{(0)} = \begin{bmatrix} -5 \\ 5 \end{bmatrix}$

$$z^{(1)} = Az^{(0)} = \begin{bmatrix} 1 & 3 \\ 2 & 2 \end{bmatrix}\begin{bmatrix} -5 \\ 5 \end{bmatrix} = \begin{bmatrix} 10 \\ 0 \end{bmatrix}$$

$$z^{(2)} = Az^{(1)} = A^2 z^{(0)} = \begin{bmatrix} 1 & 3 \\ 2 & 2 \end{bmatrix}\begin{bmatrix} 10 \\ 0 \end{bmatrix} = \begin{bmatrix} 10 \\ 20 \end{bmatrix}$$

$$z^{(3)} = Az^{(2)} = A^3 z^{(0)} = \begin{bmatrix} 1 & 3 \\ 2 & 2 \end{bmatrix}\begin{bmatrix} 10 \\ 20 \end{bmatrix} = \begin{bmatrix} 70 \\ 60 \end{bmatrix}$$

$$z^{(4)} = Az^{(3)} = A^4 z^{(0)} = \begin{bmatrix} 1 & 3 \\ 2 & 2 \end{bmatrix}\begin{bmatrix} 70 \\ 60 \end{bmatrix} = \begin{bmatrix} 250 \\ 260 \end{bmatrix} = 260\begin{bmatrix} \frac{250}{260} \\ 1 \end{bmatrix} \approx 260\begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Review the calculation for

$$z^{(0)} = 1\begin{bmatrix} 1 \\ 1 \end{bmatrix} + 2\begin{bmatrix} -3 \\ 2 \end{bmatrix}$$

$$z^{(1)} = Az^{(0)} = 4\begin{bmatrix} 1 \\ 1 \end{bmatrix} - 2\begin{bmatrix} -3 \\ 2 \end{bmatrix}$$

$$z^{(2)} = A^2 z^{(0)} = 4^2\begin{bmatrix} 1 \\ 1 \end{bmatrix} + 2\begin{bmatrix} -3 \\ 2 \end{bmatrix}$$

$$z^{(3)} = A^3 z^{(0)} = 4^3\begin{bmatrix} 1 \\ 1 \end{bmatrix} - 2\begin{bmatrix} -3 \\ 2 \end{bmatrix}$$

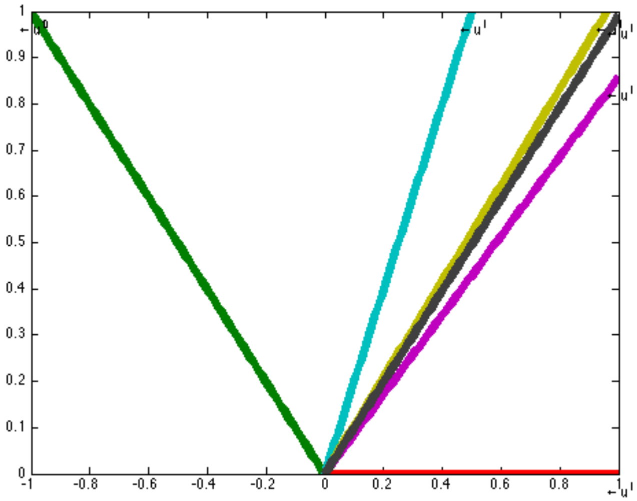
$$z^{(4)} = A^4 z^{(0)} = 4^4\begin{bmatrix} 1 \\ 1 \end{bmatrix} + 2\begin{bmatrix} -3 \\ 2 \end{bmatrix} = 256\begin{bmatrix} 1 \\ 1 \end{bmatrix} + 2\begin{bmatrix} -3 \\ 2 \end{bmatrix}$$

To keep the numbers from getting out of hand, it is necessary normalize at each step.

The power method: Example

```
for $i = 1 : i_{max}$
    $u = z/\|z\|_\infty$
    $z = Au$
    $r = (u^T * A * u)/(u^T * u)$
end
```

The power method: Example



The power method - linear convergence rate

Assume A has a diagonal Jordan canonical form:

$$P^{-1}AP = D = \text{diag}\left(\lambda_1, \ldots, \lambda_n\right)$$

and there is a single dominant eigenvalue

$$|\lambda_1| > |\lambda_2| \geq \ldots \geq |\lambda_n|.$$

We seek λ_1, x_1.

Recall that the columns of P, $x_1, \ldots, x_n$ form a basis of $\mathbb{C}^n$ and

$$Ax_i = \lambda_i x_i, \quad i = 1, \ldots, n$$

Let $z^{(0)}$ be a initial guess for x_1:

$$z^{(0)} = \sum_{j=1}^{n} \alpha_j x_j.$$

Assuming $\alpha_1 \neq 0$, apply A repeatedly:

$$A^k z^{(0)} = \sum_{j=1}^{n} \alpha_j A^k x_j = \sum_{j=1}^{n} \alpha_j \lambda_j^k x_j = \lambda_1^k \left(\alpha_1 x_1 + \sum_{j=2}^{n} \alpha_j \left(\frac{\lambda_j}{\lambda_1} \right)^k x_j \right) \approx \alpha_1 \lambda_1^k x_1$$

for k large (the convergence rate of $\frac{A^k z^{(0)}}{\lambda_1^k}$ to $\alpha_1 x_1$ is $r \leq \left|\frac{\lambda_2}{\lambda_1}\right|$, if $\alpha_2 \neq 0$).

The power method - linear convergence rate

Algorithm: initial guess $z^{(0)} \in \mathbb{R}^n$

```
for $k = 1, 2, 3, \dots$ do
    • $w^{(k)} = Az^{(k-1)}$
    • $z^{(k)} = \frac{w^{(k)}}{\|w^{(k)}\|_\infty}$, $\lambda_1^{(k)} = \frac{w_\ell^{(k)}}{z_\ell^{(k-1)}}$, $\ell$ the component used for computing an eigenvalue approximation.
```

Theorem

$$\lim_{k\to\infty} |\lambda_1 - \max(w_k)| = 0, \lim_{k\to\infty} \|z^{(k)} - cx_1\| = 0, \quad c \in \mathbb{R}, x_1 \text{ eigenvector.}$$

Moreover:

$$\lambda_1^{(k)} = \lambda_1 \left(1 + O(\left|\tfrac{\lambda_2}{\lambda_1}\right|^m)\right), \quad \left\|z^{(k)} - c_m \tfrac{x_1}{\|x_1\|_\infty}\right\|_\infty \le \gamma \left|\tfrac{\lambda_2}{\lambda_1}\right|^m, \quad k \ge 0, |c_m| = 1.$$

Proof: Induction on $k \Rightarrow z^{(k)} = \dfrac{A^k z^{(0)}}{\max(A^k z^{(0)})}$. Let x_1 be an eigenvector associated with λ_1 and complete by $(x_2, \ldots, x_n)$ s.t. $Ax_i = \lambda_i x_i$.

$$z^{(0)} = \alpha_1 x_1 + \ldots + \alpha_n x_n \Rightarrow A^k z^{(0)} = \alpha_1 A^k x_1 + \ldots + \alpha_n A^k x_n$$

$$= \alpha_1 \lambda^k x_1 + \ldots + \alpha_n \lambda_n^k x_n = \lambda_1^k \Big(\alpha_1 x_1 + \alpha_2 \big(\underbrace{\lambda_2/\lambda_1}_{<1}\big)^k x_2 + \ldots + \alpha_n (\lambda_n/\lambda_1)^k x_n\Big)$$

$$\Rightarrow \lim_{k\to\infty} z^{(k)} = \lim_{k\to\infty} \frac{A^k z^{(0)}}{\max(A^k z^{(0)})} = \lim_{k\to\infty} \frac{\lambda_1^k \alpha_1 x_1}{\max(\lambda_1^k \alpha_1 x_1)} = \frac{x_1}{\max(x_1)} = c x_1. \blacksquare$$

The steps deliver improved approximate eigenvectors: what is the best guess for the associate eigenvalue?

$$Ax = \lambda x,\ x \text{ is an approximate eigenvector, } \lambda \text{ unknown.}$$

Least squares: the answer is the solution of $x^T x \lambda = x^T A x$, i.e., the **Rayleigh quotient**

$$\lambda = \frac{x^T A x}{x^T x}.$$

With $\lambda_1^{(k)} = \frac{(Az^{(k)}, z^{(k)})}{(z^{(k)}, z^{(k)})}$ it can be shown that $\lambda_1^{(k)} = \lambda_1 \Big(1 + O(\left|\frac{\lambda_2}{\lambda_1}\right|^{2m})\Big)$,

so the **decay at each iteration is by a factor of** $\left|\frac{\lambda_2}{\lambda_1}\right|^2$.

The inverse iteration method

If Power Iteration is applied to the inverse of A, the smallest eigenvalue can be found.

Having σ as a good approximation of λ_1, we seek the eigenvector.

Let $eig(A) = \{\lambda_1, \lambda_2, \dots, \lambda_n\}$

- $eig(A^{-1}) = \{\lambda_1^{-1}, \lambda_2^{-1}, \dots, \lambda_n^{-1}\}$, assuming $\exists A^{-1}$. The eigenvectors are the same as those of A.
- $eig(A - \sigma I) = \{\lambda_1 - \sigma, \lambda_2 - \sigma, \dots, \lambda_n - \sigma\}$. The eigenvectors are the same as those of A.

Proof: 1. $Ax = \lambda x, x = \lambda A^{-1}x, A^{-1}x = \frac{1}{\lambda}x$, and the eigenvector is unchanged.

2. $Ax = \lambda x, (A - \sigma I)x = (\lambda - \sigma)x$. ■

To avoid explicit calculation of A^{-1}, rewrite the Power Iteration to A^{-1}:

$$w^{(k)} = A^{-1}z^{(k-1)} \quad \text{as} \quad Aw^{(k)} = z^{(k-1)}$$

which we solve for $w^{(k)}$ by Gaussian elimination.

The inverse iteration method

Algorithm: initial guess $z^{(0)} \in \mathbb{R}^n$

```
for $k = 1, 2, 3, \ldots$ do
    • solve $(A - \sigma I)w_k = z_{k-1}$
    • $z^{(k)} = \frac{w_k}{\|w_k\|_\infty}$
```

This is the power method applied to $(A - \sigma I)^{-1}$ *To find the eigenvalue of A closest to σ:*

1. *apply Inverse Power Iteration to get the smallest eigenvalue a of $A - \sigma I$ by Gaussian elimination.*
2. *then $\lambda = \frac{1}{a} + \sigma$ is the eigenvalue of A closest to σ.*

Heuristic result: Write $P(A - \sigma I) = LU$ and perform LU factorization with partial pivoting of $A - \sigma I$, obtaining a permutation matrix P.

$$z^{(0)} = P^{-1}Le, e = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}.$$

2 or 3 iterations of the inverse iteration $\Rightarrow$ a good approximation of x_1.

$$A\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{pmatrix}, \qquad \text{eig}(A) = \{0, -0.6235, 9.6235\}.$$

Take $\sigma = 9.1, z^{(0)} = P^{-1}L\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0.753 \\ -0.216 \end{pmatrix}$

$$\Rightarrow z^{(1)}, z^{(2)} = \begin{pmatrix} 0.531 \\ 0.763 \\ 1 \end{pmatrix}.$$

Rayleigh Quotient

Definition

$\forall x \neq 0, R = \frac{x^T A x}{x^T x}$ is the Rayleigh quotient.

Algorithm: initial guess $z^{(0)} \in \mathbb{R}^n$

```
for $k \geq 0$ do
    • $R_k = \dfrac{z^{(k)T} A z^{(k)}}{z^{(k)T} z^{(k)}}$
    • $(A - R_k I) w_{k+1} = z^{(k)}$
    • $z^{(k+1)} = \frac{w_{k+1}}{\max(w_{k+1})}$.
```

$\Rightarrow (z^{(N)}, R_N)$ is a good approximation to (cx_1, λ_1).

Computing intermediate eigenvalues: deflation

Definition

Let $w \in \mathbb{C}^n$, $\|w\|^2 = w^*w = 1$. Define the Hermitian unitary (orthogonal if $w \in \mathbb{R}^n$) matrix H by

$$H = I - 2ww^* \qquad (I - 2ww^T \text{ for } w \in \mathbb{R}^n).$$

Remark

$$H^* = I - \frac{2}{u^*u}(uu^*)^* = I - \frac{2}{u^*u}uu^* = H$$

and

$$H^*H = HH^* = H^2 = \left(I - \frac{2}{u^*u}uu^*\right)\left(I - \frac{2}{u^*u}uu^*\right)$$
$$= I - \frac{4}{u^*u}uu^* + \frac{4}{(u^*u)^2}u(u^*u)u^* = I.$$

Ordinary Differential Equations

Differential Equations are among the most important mathematical tools used in producing models in physical biological science and engineering.

An ODE is an equation that involves one or more derivatives of an unknown function.
A solution of a DE is a specific function that satisfies the equation.

Solving ODEs is about 'predicting the future'.

- We know:
  - initial state $\mathbf{x}(0)$
  - law: $\mathbf{x}'(t) = \mathbf{f}(t, \mathbf{x}(t))$
- Goal: to predict $\mathbf{x}(t)$ for $t > 0$.

Definition

Let $D \subset \mathbb{R} \times \mathbb{R}^n$, $f(t,x)$ continuous in D.
$x(t)$ is a solution on $[t_0, T]$ of the **initial value problem**

$$\begin{cases} x'(t) = f(t, x(t)) \\ x(t_0) = x_0 \end{cases} \tag{10.1}$$

if $\forall t \in [t_0, T]$

- $(t, x(t)) \in D$
- $x(t_0) = x_0$
- $\exists x'(t)$ and $x'(t) = f(t, x(t))$

Example

For simple ODEs, it is possible to find closed-form solutions.
Given a function g, the general solution of the *simplest* equation:

$$x'(t) = g(t)$$

is

$$x(t) = \int g(t)dt + C$$

with C an arbitrary integration constant.
The constant C, and hence a particular solution is obtained by specifying the value of x at some given point

$$x(t_0) = x_0.$$

The general solution of $x'(t) = \sin(t)$ is $x(t) = -\cos(t) + c$.
If we specify $x(\frac{\pi}{3}) = 2$ then $C = 2.5$ and the solution is

$$x(t) = 2.5 - cos(t).$$

Linear Equation

The more general equation

$$x'(t) = f(t, x(t))$$

is approached in a similar spirit, in the sense that there is a general solution dependent on a constant.

Example

The general first-order linear DE

$$x'(t) = a(t)x + b(t) \qquad t \in [t_0, T]$$

with $a(t), b(t) \in C[t_0, T]$, $D = \{(t, x) \in [t_0, T] \times \mathbb{R}\}$

Special case

$$x' = \lambda x + b(t), \qquad t \geq 0,$$

$$x(t) = e^{\lambda t} x_0 + \int_0^t e^{\lambda(t-s)} b(s) ds \qquad t \geq 0.$$

Method of integrating factor.

Separable Equation

Example

Nonlinear equation

$$x' = -x^2$$

has as solutions

$$x(t) \equiv 0$$

$$x(t) = \frac{1}{t+c}, c > 0 \quad (c\text{-determined from the initial condition})$$

- For a general RHS $f(t, x(t))$,
  1. it is usually *not possible* to solve the initial value problem $x'(t) = f(t, x(t))$ analytically.
     $$x' = e^{-tx^4}$$
  2. and numerical methods are the only possible way to compute solutions.
- Even when a DE can be solved analytically, the solution formula involves integration of general functions, that have to be evaluated numerically.

Example.

$$\begin{aligned} &x' = 2tx + 1, \qquad t > 0 \\ &x(0) = 1 \end{aligned}$$

with solution

$$x(t) = e^{t^2} \int_0^t e^{-t^2} dt + e^{t^2}$$

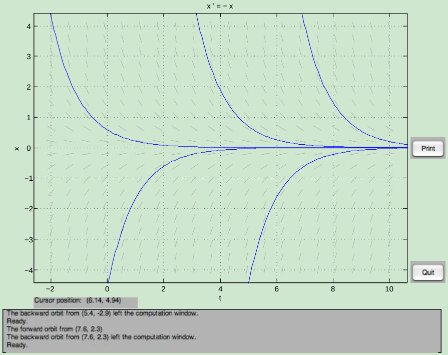
For such a solution, it is usually *more efficient* to use numerical methods from the outset to solve the differential equation.

Geometric insight: **direction field**

Direction field - induced in the tx-plane by the equation, the slope of a solution $x(t)$ that passes through (t_0, x_0) is $x'(t_0) = f(t_0, x_0)$.

Example

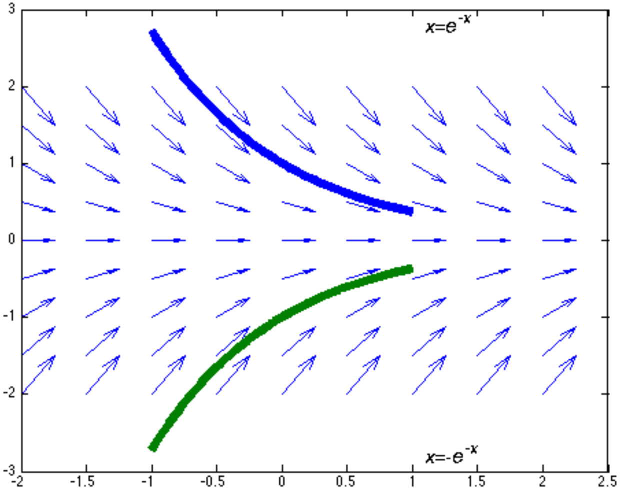
$x' = -x$, **level curves**: $f(t, x) = Const.$



```
Cursor position:  (6.14, 4.94)
```

```
The backward orbit from (5.4, -2.9) left the computation window.
Ready.
The forward orbit from (7.6, 2.3)
The backward orbit from (7.6, 2.3) left the computation window.
Ready.
```

Geometric insight: **direction field**



Geometric insight: **direction field**

```
[x,y] = meshgrid(-2:0.5:2,-2:0.5:2);
dx = ones(9); %Generates a matrix of 1's.
dy = -y;
quiver(x,y,dx,dy);

hold on
x = -1:0.01:1;
y1 = exp(-x);
y2 = -exp(-x);
plot(x,y1,x,y2,'LineWidth',6);
text(1.1,2.8,"\itx=e^{-x}","Fontsize",14)
text(1.1,-2.8,"\itx=-e^{-x}","Fontsize",14)
hold off
```

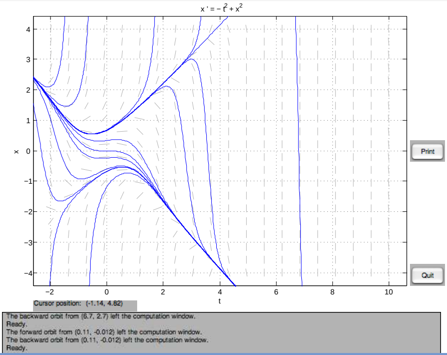
Geometric insight: **direction field**

Example

Qualitative behavior of

$$x' = t^2 + x^2.$$

The **level curves** $t^2 + x^2 = Const > 0$ are circles centered in $(0, 0)$ with radius $\sqrt{Const}$.



```
Cursor position:  (-1.14, 4.82)
```

```
The backward orbit from (6.7, 2.7) left the computation window.
Ready.
The forward orbit from (0.11, -0.012) left the computation window.
The backward orbit from (0.11, -0.012) left the computation window.
Ready.
```

Existence and Uniqueness

Theorem (Picard-Lindelöf)

Let $D \in \mathbb{R}^2$ a convex set, $f \in C(D)$ and $(t_0, x_0) \in \overset{\circ}{D}$ (interior point of D).
Assume f is **Lipschitz continuous** *in the second argument*

$$|f(t, x_1) - f(t, x_2)| \le K|x_1 - x_2|, \qquad \forall (t, x_1), (t, x_2) \in D$$

for some $K \ge 0$.
Then $\exists!$ (unique) local solution $x(t)$ to the IVP (10.1) on $I = [t_0 - \alpha, t_0 + \alpha]$.

Recall: D is convex if
$\forall (t_1, x_1), (t_2, x_2) \in D, \forall \lambda \in [0, 1] \Rightarrow \lambda(t_1, x_1) + (1 - \lambda)(t_2, x_2) \in D$,
i.e., if the the points are in D, then the seqment is in D as well.

Example

Consider $x' = 1 + \sin(tx)$ with $D = [0,1] \times \mathbb{R}$. The Lipschitz constant

$$K = \max_D \left| \frac{\partial f(t,x)}{\partial x} \right| \equiv \max_D |t \cos(tx)| = 1.$$

$\forall (t_0, x_0)$ with $t_0 \in (0,1)$, $\exists$ a solution $x(t)$ to the IVP on $I = [t_0 - \alpha, t_0 + \alpha]$.

Example

IVP: $\quad x' = \frac{2t}{a^2} x^2, \quad x(0) = 1$, $a > 0$.

The solution $x(t) = \frac{a^2}{a^2 - t^2}, \quad t \in (-a, a)$. If a is small, existence only on a small interval. The Lipschitz constant

$$\frac{\partial f(t,x)}{\partial x} = \frac{4tx}{a^2}$$

which is finite if D is bounded.

Stability of the solution

The perturbed problem

$$\begin{cases} x'(t) = f(t,x) + \delta(t) \\ x(t_0) = x_0 + \varepsilon \end{cases}$$

where $\delta(t) \in C[t_0, T]$.

Theorem (IVP is well-posed, stable)

$\exists!$ $x(t,\delta,\varepsilon)$ on $[t_0-\alpha, t_0+\alpha]$, $\alpha > 0$, uniformly for all $\varepsilon, \delta(t)$ s.t.

$$|\varepsilon| \le \varepsilon_0, \qquad \|\delta\|_\infty \le \varepsilon_0$$

for ε_0 sufficiently small. Moreover

$$\max_{|t-t_0|\le\alpha} |x(t) - x(t,\delta,\varepsilon)| \le \kappa(|\varepsilon| + \alpha\|\delta\|_\infty)$$

with $\kappa = 1/(1-\alpha K)$, K the Lipschitz constant of f.

Stable but ill-conditioned w.r.t numerical computation

$$\begin{cases} x'(t,\varepsilon) = f(t, x(t,\varepsilon)), & [t_0-\alpha, t_0+\alpha] \\ x(t_0,\varepsilon) = x_0 + \varepsilon \end{cases}$$

Subtract from IVP for $x(t)$ to obtain for $z(t,\varepsilon) = x(t,\varepsilon) - x(t)$:

$$\begin{cases} z'(t,\varepsilon) = f(t, x(t,\varepsilon)) - f(t, x(t)) \approx \frac{\partial f(t,x(t))}{\partial x} z(t,\varepsilon) \\ z(t_0,\varepsilon) = \varepsilon. \end{cases}$$

The approximate (*linear*) DE has the solution

$$z(t,\varepsilon) = \varepsilon e^{\int_{t_0}^{t} \frac{\partial f(s,x(s))}{\partial x} ds}.$$

Stable but ill-conditioned w.r.t. numerical computation

- If $\frac{\partial f(s,x(s))}{\partial x} \leq 0, |t_0 - s| \leq \alpha$ then $z(t,\varepsilon)$ is bounded by ε as t increases $\Rightarrow$ IVP is *well-conditioned*. (*stiff DE* when $\frac{\partial f(s,x(s))}{\partial x}$ is large in magnitude)
- Opposite behavior: $x' = \lambda x + b(t), \quad x(0) = x_0$ with $\lambda > 0$. Then $\partial f/\partial x = \lambda$ and $z(t,\varepsilon) = \varepsilon e^{\lambda t}$, increasing as t increases (*ill-cond.*).

Example (ill-conditioned problem)

The equation

$$x' = 100x - 101e^{-t}, \qquad x(0) = 1$$

has the solution

$$x(t) = e^{-t}$$

and the perturbed problem

$$x' = 100x - 101e^{-t}, \qquad x(0) = 1 + \varepsilon$$

has the solution

$$x(t,\varepsilon) = e^{-t} + \varepsilon e^{100t}.$$

For a simple linear IVP:

$$\begin{cases} x' = \lambda x, & \lambda \in \mathbb{R} \\ x(t_0) = x_0 \end{cases}$$

we have $\frac{\partial f}{\partial x} = \lambda$, and

- if $\lambda \leq 0$, then ODE is stable
- if $\lambda \geq 0$, then ODE is **un**stable:

$$z(t, \varepsilon) = \varepsilon e^{\lambda(t-t_0)} \to \infty \text{ as } t \to \infty$$

Systems of DEs

All previous results apply for a system of m first-order equations:

$$\begin{cases} x_1' = f_1(t, x_1, \ldots, x_m), & x_1(t_0) = x_{1,0} \\ \vdots & \\ x_m' = f_m(t, x_1, \ldots, x_m), & x_m(t_0) = x_{m,0} \end{cases}$$

or

$$\mathbf{x}' = \mathbf{f}(t, \mathbf{x}), \qquad \mathbf{x}(t_0) = \mathbf{x}_0,$$

where

$$\mathbf{x}(t) = \begin{bmatrix} x_1(t) \\ \vdots \\ x_m(t) \end{bmatrix} \quad \mathbf{f}(t, \mathbf{x}) = \begin{bmatrix} f_1(t, \mathbf{x}) \\ \vdots \\ f_m(t, \mathbf{x}) \end{bmatrix} \quad \mathbf{x}_0(t) = \begin{bmatrix} x_{1,0} \\ \vdots \\ x_{m,0} \end{bmatrix}.$$

Systems of DEs

If $\mathbf{f}$ is differentiable, then we can compute the **Jacobian** matrix:

$$\mathbf{f}_{\mathbf{x}} = \left(\frac{\partial f_i}{\partial x_j}\right)_{1\leq i,j\leq m}.$$

Then the Lipschitz constant in the existence theorem is given by $\|\mathbf{f}_{\mathbf{x}}\|$, where $\|\cdot\|$ is a matrix norm (induced by a vector norm).

Recall

If $\|\cdot$ is a vector norm, then

$$\|A\| = \sup_x \frac{\|Ax\|}{\|x\|}.$$

Systems of DEs

Example

Let $A \in \mathbb{R}^{m\times m}$ and the first-order constant coefficients system

$$\begin{cases} \mathbf{x}' = A\mathbf{x} & (A \text{ does not vary w.r.t. } t) \\ \mathbf{x}(t_0) = \mathbf{x}_0 \end{cases} \tag{10.2}$$

Solution: $\mathbf{x}(t) = e^{At}\mathbf{x}_0$.

Definition

λ is an **eigenvalue** of A if there exists $\mathbf{x} \neq 0$ such that $A\mathbf{x} = \lambda\mathbf{x}$.

- The IVP (10.2) is **stable** if **all eigenvalues** satisfy
  - $Re(\lambda) < 0$ or
  - $Re(\lambda) = 0$ and λ is simple (multiplicity 1)
- The IVP (10.2) is **asymptotically stable** if **all eigenvalues** satisfy $Re(\lambda) < 0$.

Systems of DEs

Example

$$\begin{cases} x_1' = x_2 \\ x_2' = x_1 \end{cases} \Rightarrow \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}' = \underbrace{\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}}_{A} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

Question: is this ODE stable?

The eigenvalues are $+1, -1 \Rightarrow$ **Unstable**.

Higher order equations

The IVP

$$\begin{cases} x^{(m)} = f(t, x, x', \ldots, x^{(m-1)}) \\ x(t_0) = x_0, \quad \ldots \quad , x^{(m-1)}(t_0) = x_0^{(m-1)} \end{cases}$$

can be converted to a first-order system with the unknowns

$$\boldsymbol{x_1 = x, x_2 = x', \ldots, x_m = x^{(m-1)}.}$$

These functions satisfy the system

$$\begin{cases} x_1' = x_2 & x_1(t_0) = x_0 \\ x_2' = x_3 & x_2(t_0) = x_0' \\ \vdots & \vdots \\ x_{m-1}' = x_m & \\ x_m' = f(t, x_1, \ldots, x_m) & x_m(t_0) = x_0^{(m-1)}. \end{cases}$$

Linear second-order equation

Example

$$x'' = a_1(t)x' + a_0(t)x + g(t), \quad x(t_0) = \alpha, \quad x'(t_0) = \beta$$

becomes

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}' = \underbrace{\begin{bmatrix} 0 & 1 \\ a_0(t) & a_1(t) \end{bmatrix}}_{A(t)} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ g(t) \end{bmatrix}}_{\mathbf{G}(t)}, \qquad \begin{bmatrix} x_1(t_0) \\ x_2(t_0) \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}.$$

In vector form

$$\mathbf{x}' = A(t)\mathbf{x} + \mathbf{G}(t), \qquad \mathbf{x}(t_0) = \mathbf{x}_0 = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}.$$

Taylor Series Methods

This numerical methods are based on the representation of a DE locally by a few terms of its Taylor series.

$$x(t+h) = x(t) + hx'(t) + \frac{1}{2!}h^2x''(t) + \frac{1}{3!}h^3x'''(t) + \frac{1}{4!}h^4x^{(4)}(t) + \cdots + \frac{1}{m!}h^mx^{(m)}(t) + \frac{1}{(m+1)!}h^{m+1}x^{(m+1)}(c)$$

Taylor series method of order m

$$y_0 = x_0$$
$$y_{i+1} = y_i + hf(t_i, y_i) + \frac{h^2}{2}f'(t_i, y_i) + \cdots + \frac{h^m}{m!}f^{(m-1)}(t_i, y_i)$$

where prime notation refers to the total derivative of $f(t, y(t))$ w.r.t t:

$$\begin{aligned} f'(t,y) &= f_t(t,y) + f_y(t,y)y'(t) \\ &= f_t(t,y) + f_y(t,y)f(t,y). \end{aligned}$$

Taylor Series Methods

To find the local truncation error of the Taylor method, set y_i to x_i and compare to the Taylor expansion

$$x_{i+1} - y_{i+1} = \frac{h^{m+1}}{(m+1)!} x^{(m+1)}(c)$$

The Taylor method of order m has a local truncation error h^{m+1}

1. The first-order Taylor Method is

$$y_{i+1} = y_i + hf(t_i, y_i),$$

   which is Euler's method.

2. The second order Taylor Method is

$$y_{i+1} = y_i + hf(t_i, y_i) + \frac{1}{2}h^2\Big(f_t(t_i, y_i) + f_y(t_i, y_i)f(t_i, y_i)\Big)$$

Taylor Series Methods

Example

Determine the second-order Taylor Method for the following

$$x' = tx + t^3$$
$$x(0) = x_0$$

Since $f(t,x) = tx + t^3$, it follows that

$$\begin{aligned} f'(t,x) &= f_t + f_x f \\ &= x + 3t^2 + t(tx + t^3) \end{aligned}$$

and the method gives

$$y_{i+1} = y_i + h(t_i y_i + t_i^3) + \frac{1}{2}h^2\Big(y_i + 3t_i^2 + t_i(t_i y_i + t_i^3)\Big)$$

Forward Euler Method

$$\begin{cases} x' = f(t,x) \\ x(t_0) = x_0. \end{cases}$$

Let $t_0 < t_1 < \cdots < t_N = T$ a partition of $[t_0, T]$

We construct

$$x_1, x_2, \cdots$$

such that

$$x_i \simeq x(t_i)$$

Let $h_{i+1} = t_{i+1} - t_i$

The method is:

$$\begin{cases} x_0 & \textbf{(given)} \\ x_{n+1} = x_n + h_{n+1} f(t_n, x_n) \end{cases}$$

Since the RHS does not depend on x_{n+1}, the FE method is explicit

Algorithm: Euler

For simplicity assume $h_i = h, \forall i$

- Input: h, mesh size
  N, number of steps
- Output: x_n, approximation of $x(t_n)$
- Initialize:

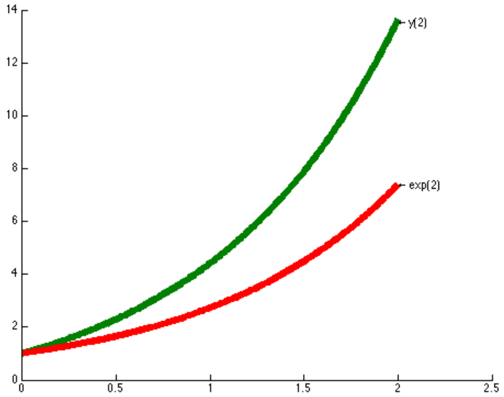
$$x_{old} = x_0$$
$$t_{old} = t_0$$

- Loop over number of steps

```
*$x_{new} = x_{old} + h \cdot f(t_{old}, x_{old})$
Is $t_{old} > t_{final}$?
If not
    $t_{old} \Leftarrow t_{old} + h$
    $x_{old} \Leftarrow x_{new}$
Go to *
```

Example: $x' = -x + 2\cos t$

Example: $x'(t) = x, x(0) = 1.$



How accurate are the answers?
Are higher-order Taylor series methods ever needed?

Taylor series method of higher order

Consider the IVP

$$\begin{cases} x' = 1 + x^2 + t^3 \\ x(1) = -4 \end{cases}$$

The computed value with forward Euler method is $x(2) \approx 4.23585$.

Let differentiate the DE:

$$\begin{aligned} x' &= 1 + x^2 + t^3 \\ x'' &= 2xx' + 3t^2 \\ x''' &= 2xx'' + 2(x')^2 + 6t \\ x^{(4)} &= 2xx''' + 6x'x'' + 6 \end{aligned}$$

If we know t and $x(t)$ numerically, then these formulae give $x'(t), x''(t), x'''(t), x^{(4)}(t)$ and use them in the Taylor series.

```
% Taylor series Method of higher order for Solving Initial Value Problems
% Input:  interval [a,b], initial value x0, step size h
% Output:  time steps t, solution y

% Example usage:  x=taylor([1 2],2,0.01);

function [x,t] = taylor(int,x0,h)
a = int(1); b = int(2);
x = x0;
n = round((int(2)-int(1))/h);
t = a;
for k = 1 :  n
xprime = 1 + x.^2 + t^3;
xsecond = 2*x.*xprime + 3*t^2;
xtertius = 2*x*xsecond + 2* xprime*xprime + 6*t;
xquatro = 2*x*xtertius + 6* xprime*xsecond + 6;
x = x + h *(xprime + .5*h*(xsecond + (1/3)*h*(xtertius +
.25*h*xquatro)));
t = a + k*h;
end
```

Taylor series method of higher order

Forward Euler Method	$x(2) \approx 4.235841$
Taylor Series Method of Order 4	$x(2) \approx 4.3712096$
Correct value	$x(2) = 4.371221866$

Derivation of F.E.

1. Euler's idea: tracking slope field, approximate the graph by the line tangent at (t,x(t))
2. Taylor's series ($x(t)$ = true solution),

$$x(t+h) = x(t) + h\underbrace{x'(t)}_{f(t,x(t))} + \underbrace{\frac{h^2}{2}x''(\xi)}_{h\cdot\text{truncation error}} \Rightarrow x(t+h) \approx x(t) + hf(t,x(t))$$

3. Numerical differentiation

$$\underbrace{x'(t)}_{\frac{x(t+h)-x(t)}{h}} = f(t,x(t)) \Rightarrow \frac{x_{n+1}-x_n}{h} = f(t_n,x_n)$$

4. Numerical integration: $\int_{t_n}^{t_{n+1}} \Big/ \; x'(t) = f(t,x(t))$

$$x(t_{n+1}) - x(t_n) = \int_{t_n}^{t_{n+1}} f(t,x(t))dt \approx f(t_n,x(t_n))(t_{n+1}-t_n)$$
$$\Rightarrow x_{n+1} - x_n = hf(t_n,x_n)$$

Error Analysis

Definition

- **global error**: $x(t_n) - x_n$ at the node t_n
- truncation (consistency) error

$$\tau_n = \frac{x(t_{n+1}) - x(t_n)}{h_{n+1}} - f(t_n, x(t_n)) = O(h_{n+1})$$

Truncation error - introduced at each step of the Euler method

Is the error in approximating the differential operator by a **finite difference operator:**

$$\begin{cases} x'(t_n) = f(t_n, x_n) \\ x'(t_n) \simeq \frac{x(t_n + h_{n+1}) - x(t_n)}{h_{n+1}} \end{cases}$$

From the Taylor expansion:

$$x(t_{n+1}) = x(t_n) + h_{n+1}x'(t_n) + \frac{h_{n+1}^2}{2}x''(t_n) + O(h_{n+1}^3)$$
$$\Rightarrow \tau_n = \frac{h_{n+1}}{2}x''(t_n) + O(h_{n+1}^2) = O(h_{n+1}) \to 0 \text{ as } h_{n+1} \to 0$$

We say **the method is consistent** (goes to 0) **of order 1**

Example

The problem

$$x'(t) = 2t, x(0) = 0$$

has the solution $x(t) = t^2$ and Euler's method is

$$x_{n+1} = x_n + 2ht_n, \quad x_0 = 0$$

with $x_n = t_{n-1}t_n$. The error

$$x(t_n) - x_n = t_n^2 - t_n t_{n-1} = ht_n$$

is proportional to h.

Theorem

Theorem

Assume f is uniformly Lipschitz

$$|f(t_1,x_1) - f(t_1,x_2)| \leq K|x_1 - x_2|, \quad \forall x_1, x_2 \in \mathbb{R}, \forall t \in [t_0, T]$$

and the exact solution $x \in C^2[0,T]$. Then $\{x_n\}_n$ obtained by F.E.'s method satisfies

$$\max_{t_n \in [t_0,T]} |\underbrace{x(t_n) - x_n}_{e_n}| \leq e^{(T-t_0)K}|e_0| + \frac{e^{(T-t_0)K} - 1}{K}\tau(h) \qquad (10.3)$$

where $\tau(h) = \frac{h}{2}\|x''\|_\infty$ and $e_0 = x(t_0) - x_0$.

If $x(t_0) - x_0 \leq Ch$ as $h \to 0$, then there exists $C_1 \geq$ such that

$$\max_{t_n \in [t_0,T]} |x(t_n) - x_n| \leq C_1 h.$$

Proof

Let $e_n = x(t_n) - x_n$ and define $\tau_n = \frac{h}{2}x''(\xi_n)$, the truncation error. Assume $h_n = h, \forall n$. Then

$x(t_{n+1}) = x(t_n) + hf(t_n, x(t_n)) + h\tau_n$ **(Taylor's expansion)**

$x_{n+1} = x_n + hf(t_n, x_n)$ **(Forward Euler)**

Subtracting:

$$e_{n+1} = e_n + h\Big(f(t_n, x(t_n)) - f(t_n, x_n)\Big) + h\tau_n$$

$$|e_{n+1}| \le |e_n| + hK|x(t_n) - x_n| + h|\tau_n| \le (1 + hK)|e_n| + h|\tau_n|$$

or

$$\begin{aligned} |e_n| &\le (1 + hK)|e_{n-1}| + h|\tau_{n-1}| \\ &\le (1 + hK)\Big((1 + hK)|e_{n-2}| + h|\tau_{n-2}|\Big) + h|\tau_{n-1}| \\ &\le (1 + hK)^2|e_{n-2}| + \frac{h^2}{2}\|x''\|_\infty\Big(1 + (1 + hK)\Big) \le \cdots \end{aligned}$$

$$
\begin{aligned}
|e_n| &\le (1+hK)^3|e_{n-3}| + \frac{h^2}{2}\|x''\|_\infty\Big(1 + (1+hK) + (1+hK)^2\Big) \\
&\le (1+hK)^n|e_0| + \frac{h^2}{2}\|x''\|_\infty\Big(1 + (1+hK) + \ldots + (1+hK)^{n-1}\Big) \\
&= (1+hK)^n|e_0| + \frac{(1+hK)^n - 1}{K}h\tau(h).
\end{aligned}
$$

Since $(1+hK)^n \le e^{nhK} \le e^{(T-t_0)K}$ we conclude the proof. ■

We used $1+z \le e^z$ from Taylor, which implies $(1+z)^n \le e^{nz}$

Remark

$e_n = O(h)$. *The method is convergent.*

Theorem

Theorem

Under the same assumptions, and if in addition $\frac{\partial f}{\partial x} \leq 0$ *for* $t \in [t_0, T], x \in \mathbb{R}$, *then*

$$|e_n| \leq |e_0| + \frac{h}{2}(t_n - t_0)\|x''\|_\infty \qquad \forall n = 1, \ldots, N \quad \textit{for } h \textit{ suf. small.}$$

Proof. As previously:

$$e_{n+1} = e_n + h\underbrace{\left(f(t_n, x(t_n)) - f(t_n, x_n)\right)}_{= \frac{\partial f}{\partial x}(t_n, x_n) \cdot e_n \text{ by MVT}} + \frac{h^2}{2}x''(\xi_n)$$

$$= e_n\Bigg(\underbrace{1 + h\frac{\partial f}{\partial x}(t_n, x_n)}_{\leq 1 \text{ for } h \text{ small enough}} \Bigg) + \frac{h^2}{2}x''(\xi_n)$$

$$|1 + h\frac{\partial f}{\partial x}(t_n, x_n)| \leq 1,$$

here we use $\dfrac{\partial f}{\partial x} \leq 0,$ $\qquad h \leq \dfrac{2}{-\frac{\partial f}{\partial x}}.$

The rest is similar

$$|e_{n+1}| \leq |e_n| + \frac{h^2}{2}\|x''\|_\infty$$

$$|e_n| \leq |e_{n-1}| + \frac{h^2}{2}\|x''\|_\infty \leq |e_{n-2}| + \frac{h^2}{2}\|x''\|_\infty + \frac{h^2}{2}\|x''\|_\infty$$

$$\vdots$$

$$|e_n| \leq |e_0| + n\frac{h^2}{2}\|x''\|_\infty,$$ but $nh = t_n - t_0.$ ■

Roundoff Error in finite precision

Assume that at each time step, the resulting local rounding error is ρ_n. Denote $\tilde{x_n}$ the finite precision number obtained on the computer:

$$\tilde{x}_{n+1} = \tilde{x}_n + hf(t_n, \tilde{x}_n) + \rho_n$$

and repeat the analysis to bound the error $\epsilon_n = x(t_n) - \tilde{x}_n$.

$$\epsilon_{n+1} = \epsilon_n + h(\underbrace{f(t_n, x(t_n)) - f(t_n, \tilde{x}_n)}_{\text{use Lipschitz condition}}) + \underbrace{\frac{h^2}{2}x''(\xi_n) - \rho_n}_{hA_n}$$

Denote $A_n = \frac{h}{2}x''(\xi_n) - \frac{\rho_n}{h}$, then $|A_n| \le \frac{h}{2}\|x''\|_\infty + \frac{1}{h}\|\rho\|_\infty := A$.

$$|\epsilon_{n+1}| = |\epsilon_n|(1 + hK) + hA.$$

Roundoff Error in finite precision

$$|\epsilon_{n+1}| = |\epsilon_n|(1 + hK) + hA$$

while in exact arithmetics we had

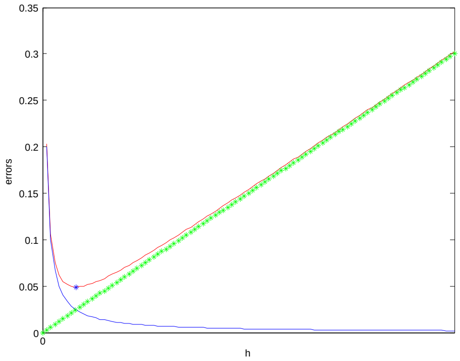
$$|e_{n+1}| = |e_n|(1 + hK) + h\frac{h}{2}\|x''\|_\infty.$$

Repeating the proof:

$$|\epsilon_n| \le \frac{e^{(T-t_0)K} - 1}{K}\left(\underbrace{\frac{h}{2}\|x''\|_\infty}_{\text{exact arithmetic}} + \underbrace{\frac{\|\rho\|_\infty}{h}}_{\text{rounding error}} \right)$$

The rounding error is negligible w.r.t. exact arithmetic error.

Roundoff Error in finite precision



Optimal h:

$$\frac{h}{2}\|x''\|_\infty = \frac{\|\rho\|_\infty}{h} \Rightarrow h_{opt} = \sqrt{\frac{2\|\rho\|_\infty}{\|x''\|_\infty}}$$

very small. (ρ - the machine precision)

Improving the accuracy of the FE solution

Recall:

$$x_{n+1} = x_n + hf(t_n, x_n) \qquad \textbf{(Forward Euler)}$$

$e_n = O(h)$, 1^{st} order method that is explicit, cheap.

Theorem

Assume the solution $x \in C^3(t_0, T)$, its derivatives are bounded and $\frac{\partial^2 f}{\partial y^2}$ is continuous and bounded. Then

$$e_n = x(t_n) - x_n = hu(t_n) + O(h^2) \quad \forall n$$

where u is the solution to the IVP

$$\begin{cases} u'(t) = \frac{\partial f}{\partial x}(t, x(t))u(t) + \frac{1}{2}x''(t) \\ u(t_0) = u_0 \end{cases}$$

and u_0 is such that $e_0 = x(t_0) - x_0 = u_0 h$.

Improving the accuracy of the FE solution

<u>Proof</u>: We show that $e_n - hu(t_n) = O(h^2)$.
Taylor expansion and F.E. to get an approximation of e_n:

$$x(t_{n+1}) = x(t_n) + hx'(t_n) + \frac{h^2}{2}x''(t_n) + \frac{h^3}{6}x'''(\xi_n)$$

$$x_{n+1} = x_n + hf(t_n, x_n)$$

$$e_{n+1} = e_n + h\Big(f(t_n, x(t_n)) - f(t_n, x_n)\Big) + \frac{h^2}{2}x''(t_n) + \frac{h^3}{6}x'''(\xi_n)$$

Taylor expansion for f:

$$f(t_n, x_n) = f(t_n, x(t_n)) + \underbrace{(x_n - x(t_n))}_{-e_n}\frac{\partial f}{\partial x}(t_n, x(t_n)) + \frac{(x_n - x(t_n))^2}{2}\frac{\partial^2 f}{\partial x^2}(t_n, \xi_n$$

$\Rightarrow$

$$e_{n+1} = e_n\left(1 + h\frac{\partial f}{\partial x}(t_n, x(t_n)) - h\frac{e_n^2}{2}\frac{\partial^2 f}{\partial x^2}(t_n, \xi_n)\right) + \frac{h^2}{2}x''(t_n) + \frac{h^3}{6}x'''(\xi_n) \tag{10.4}$$

Improving the accuracy of the FE solution

Proof (continued): Taylor expansion for u:

$$\begin{aligned} u(t_{n+1}) &= u(t_n) + hu'(t_n) + \frac{h^2}{2}u''(\delta_n) \\ &= u(t_n)+h\left(\frac{\partial f}{\partial x}(t_n, x(t_n))u(t_n)+\frac{1}{2}x''(t_n)\right)+\frac{h^2}{2}u''(\delta_n) \end{aligned}$$

$\Rightarrow$

$$u(t_{n+1}) = u(t_n)\left(1 + h\frac{\partial f}{\partial x}(t_n, x(t_n))\right) + \frac{h}{2}x''(t_n) + \frac{h^2}{2}u''(\delta_n) \tag{10.5}$$

From (10.4)-(10.5):

$$e_{n+1} - hu(t_{n+1}) = (e_n - hu(t_n))\left(1+h\frac{\partial f}{\partial x}(t_n, x(t_n))\right)+h\Big(\underbrace{O(h^2)+O(e_n^2)}_{O(h^2)}\Big)$$

since $e_n = O(h)$

Improving the accuracy of the FE solution

Proof (continued): Assume $\|\frac{\partial f}{\partial x}\|_\infty$:

$$|e_{n+1} - hu(t_{n+1})| \leq |e_n - hu(t_n)|\left(1 + \ h\left\|\frac{\partial f}{\partial x}\right\|_\infty\right) + hO(h^2),$$

and recursively:

$$|e_n\text{-}hu(t_n)| \leq \left(1+h\left\|\frac{\partial f}{\partial x}\right\|_\infty\right)^n \underbrace{|e_0 - h\,\underbrace{u(t_0)}_{=u_0}\,|}_{=0} + \frac{(1+h\left\|\frac{\partial f}{\partial x}\right\|_\infty)^n - 1}{\left\|\frac{\partial f}{\partial x}\right\|_\infty} O(h^2),$$

$$\leq \frac{e^{(T-t_0)\left\|\frac{\partial f}{\partial x}\right\|_\infty} - 1}{\left\|\frac{\partial f}{\partial x}\right\|_\infty} O(h^2), \quad \text{(since } (1+z)^n \leq e^{nz} \text{ as before).}$$

■

Improving the accuracy of the FE solution

Forward Euler: $x_0, x_1, \ldots$: error$= O(h)$.

Can we have a sequence that is more accurate: error $= O(h^2)$?

Richardson Extrapolation

Consider a mesh of size h: $t_{n+1} - t_n = h$ and a refined mesh of size $\tilde{t}_{n+1} - \tilde{t}_n = h/2$:

$$\begin{array}{cccccc} t_0 & & t_1 & & t_2 & \cdots \\ \tilde{t}_0 & \tilde{t}_1 & \tilde{t}_2 & \tilde{t}_3 & \tilde{t}_4 & \cdots \end{array} \qquad \text{(i.e. } x(\tilde{t}_m) = x(\tilde{t}_{2n}) = x(t_n))$$

regular mesh: $x(t_n) = x_n + hu(t_n) + O(h^2)$

refined mesh: $\underbrace{x(t_n)}_{x(\tilde{t}_{2n})} = \tilde{x}_{2n} + \frac{h}{2}u(\underbrace{t_n}_{\tilde{t}_{2n}}) + \underbrace{O(\left(\frac{h}{2}\right)^2)}_{O(h^2)} \qquad / \cdot (-2)$

Improving the accuracy of the FE solution

Richardson Extrapolation Formula

$$-x(t_n) = x_n - 2\tilde{x}_{2n} + O(h^2)$$

or

$$x(t_n) = 2\tilde{x}_{2n} - x_n + O(h^2)$$

Has a higher order of convergence than FE, but requires the computation of x_n and $\tilde{x}_{2n}$

Richardson Error Estimate

$$x(t_n) - x_n = 2\Big(\tilde{x}_{2n} - x_n\Big) + O(h^2)$$

an asymptotic estimate of error:

$$x(t_n) - x_n \simeq 2\Big(\tilde{x}_{2n} - x_n\Big)$$

Absolute stability

Definition

For a given numerical method, the **region of absolute stability** is the region of the complex plane $\{z \in \mathbb{C}\}$ s.t. applying the method to $x' = \lambda x$ with $z = \lambda h$ $(h \in \mathbb{R}, \lambda \in \mathbb{C})$ yields a solution x_n s.t. $|x_{n+1}| \leq |x_n|$.

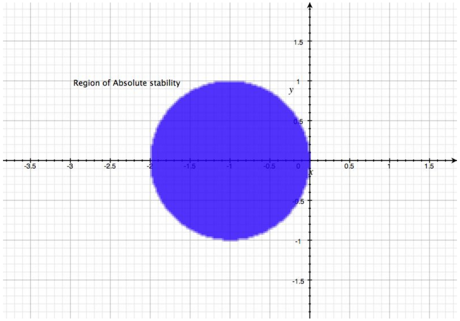
Exact solution: $x(t) = x(0)e^{\lambda t}, \quad \lambda \leq 0, \lambda \in \mathbb{R} \Rightarrow |x(t_{n+1})| \leq |x(t_n)|$
Apply the definition to Forward Euler:

$$x_{n+1} = x_n + h(\lambda x_n) = x_n(1 + \lambda h)$$

and we want $|x_{n+1}| \leq |x_n|$.
Hence we need $|1 + \underbrace{\lambda h}_{z}| \leq 1$, $|1 + z| \leq 1$.

Absolute stability

Assume $\lambda \in \mathbb{R}$ and $\lambda < 0$. Then
$|1+\lambda h| \leq 1, -1 \leq 1+\lambda h \leq 1 \Rightarrow -1 \leq 1+\lambda h \Leftrightarrow -\lambda h \leq 2 \Rightarrow h \leq \frac{2}{-\lambda}$
for stiff problems h has to be very small.



System of ODEs

$$\begin{cases} \mathbf{x}' = A\mathbf{x}, & A \in \mathbb{R}^{m\times m} \\ \mathbf{x}(0) = \mathbf{x}_0 \end{cases}$$

Assume A is diagonalizable: $\exists T \in \mathbb{R}^{m\times m}$ invertible s.t.

$T^{-1}AT = \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_m \end{pmatrix}$. With the change of variables $\mathbf{w} = T^{-1}\mathbf{x}$ the ODE becomes:

$$\mathbf{w}' = T^{-1}\mathbf{x}' = T^{-1}A\mathbf{x} = T^{-1}A\underbrace{TT^{-1}}_{I}\mathbf{x} = T^{-1}AT\mathbf{w}$$

$$\mathbf{w}' = \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_m \end{pmatrix}\mathbf{w} \Leftrightarrow w_i' = \lambda_i w_i, \quad \text{with } w_i = i^{th} \text{ component of } \mathbf{w}.$$

System of ODEs

So, absolute stability region for $\mathbf{w}$:

$$|1 + h\lambda_i| \leq 1 \qquad \Rightarrow \text{gives a constraint on } h.$$

Do we have $\|\mathbf{x}_{n+1}\| \leq \|\mathbf{x}_n\|$? We know $\|\mathbf{w}_{n+1}\| \leq \|\mathbf{w}_n\|$, and then we get back to $\mathbf{x}_n$ by

$$\|\mathbf{x}_{n+1}\| = \|T\mathbf{w_{n+1}}\| \leq \underbrace{\|T\|}_{\text{matrix induced norm}} \|\mathbf{w_{n+1}}\| \underbrace{\leq}_{\text{Absolute Stability}} \|T\|\|\mathbf{w_n}\|$$

$$= \|T\|\|T^{-1}\mathbf{x_n}\| \leq \underbrace{\|T\|\|T^{-1}\|}_{\text{condition \# of the matrix} T} \|\mathbf{x_n}\|$$

Stability Bound:

$$\|\mathbf{x}_{n+1}\| \leq \text{cond}(T)\|\mathbf{x}_n\|$$

The Midpoint Method: derivation

$$x' = f(t,x)$$

Forward Euler:

$$x'(t_n) \simeq \frac{x(t_{n+1}) - x(t_n)}{h} \Longrightarrow \text{ error } O(h).$$

Let use a higher order *finite difference*:

$$x'(t_n) = \frac{x(t_{n+1}) - x(t_{n-1})}{2h} \Longrightarrow \text{ error } O(h^2).$$

Indeed

$$\begin{aligned} x(t_{n+1}) &= x(t_n) + hx'(t_n) + \frac{h^2}{2}x''(t_n) + \frac{h^3}{6}x'''(\xi_n) \\ x(t_{n-1}) &= x(t_n) - hx'(t_n) + \frac{h^2}{2}x''(t_n) - \frac{h^3}{6}x'''(\zeta_n) \quad \Big| \cdot (-1) \end{aligned}$$

$$x(t_{n+1}) - x(t_{n-1}) = 2h\color{blue}{x'(t_n)}\color{black} + \frac{h^3}{6}\Big(x'''(\xi_n) + x'''(\zeta_n)\Big) \tag{10.6}$$

The Midpoint Method: derivation

Hence

$$x'(t_n) = \frac{x(t_{n+1}) - x(t_{n-1})}{2h} - \underbrace{\frac{h^2}{12}\Big(x'''(\xi_n) + x'''(\zeta_n)\Big)}_{O(h^2)} \qquad (10.7)$$

$$\Rightarrow \frac{x(t_{n+1}) - x(t_{n-1})}{2h} = f(t_n, x(t_n)), \quad \text{or}$$

$$\Rightarrow x(t_{n+1}) = x(t_{n-1}) + 2hf(t_n, x(t_n))$$

Midpoint Method

$$\begin{cases} x_{n+1} = x_{n-1} + 2hf(t_n, x_n) & n \geq 1 \\ x_0, x_1 \text{ - needed} \end{cases}$$

1. **explicit two-step method**
2. **order of convergence two**

The Midpoint Method: derivation

If we only have x_0, then x_1 can be obtained using Forward Euler on a finer mesh $\tilde{h} = \frac{h}{4}$...
Or use Taylor:

$$x_1 = x_0 + hf(t_0, x_0) + \frac{h^2}{2}\Big(\frac{\partial f}{\partial t}(t_0, x_0) + f(t_0, x_0)\frac{\partial f}{\partial t}(t_0, x_0)\Big)$$

Local truncation error

$$\tau_n = \frac{x(t_{n+1}) - x(t_{n-1})}{h} - 2f(t_n, x(t_n)) \underbrace{=}_{(10.6)} \frac{h^2}{6}\Big(x'''(\xi_n) + x'''(\zeta_n)\Big)$$

$\tau_n \underset{h\to 0}{\longrightarrow} 0$: The method is consistent, the order of consistency is 2.

The Midpoint Method: Error Analysis

Theorem

$$\max_{0\leq n\leq N} |x(t_n)-x_n| \leq e^{2K(T-t_0)}\eta(h)+\frac{e^{2K(T-t_0)}-1}{2K}\frac{h^2}{3}\|x^{(3)}\|_\infty \tag{10.8}$$

where $\eta(h) = \max\{|\underbrace{x(t_0)-x_0}_{\textit{usually } =0}|, |\underbrace{x(t_1)-x_1}_{\textit{wanted } O(h^2)}|\}.$

If $x(t_0)=x_0$ we still need $x(t_1)-x_1=O(h^2)$ to have global order of convergence $O(h^2)$: use single step Euler's method:

$$x_1 = x_0 + hf(t_0,x_0), \qquad x_0 = x(t_0)$$

$$x(t_1)-x_1 = \frac{h^2}{2}x''(\xi) \qquad t_0\leq\xi\leq t_1$$

(10.8)

$t_0\leq t_n\leq T$

The Midpoint Method: Error Analysis

Proof:

$$x(t_{n+1}) = x(t_{n-1}) + 2hx'(t_n) + h\tau_n \qquad \Leftarrow \textbf{(10.6): Taylor expansion}$$

$$x_{n+1} = x_{n-1} + 2hf(t_n, x_n) \qquad \Leftarrow \textbf{Midpoint Method}$$

$$e_{n+1} = e_{n-1} + 2h\Big(\underbrace{f(t_n, x(t_n)) - f(t_n, x_n)}_{\frac{\partial f}{\partial x}(t_n,\xi_n)e_n}\Big) + h\tau_n$$

$$|e_{n+1}| \le |e_{n-1}| + 2h\left\|\frac{\partial f}{\partial x}\right\|_\infty |e_n| + h\|\tau\|_\infty, \text{ where } \|\tau\|_\infty = \max_n |\tau_n|,$$

$$\tau_n = \frac{h^2}{6}\Big(x'''(\xi_n) + x'''(\zeta_n)\Big) \Rightarrow \|\tau\|_\infty \le \frac{h^2}{3}\|x'''\|_\infty.$$

Let introduce the sequence $\{a_n\}_n$:

$$\begin{cases} a_0 = \max(|e_0|, |e_1|) \\ a_{n+1} = \left(1 + 2h\left\|\frac{\partial f}{\partial x}\right\|_\infty\right)a_n + h\|\tau\|_\infty, \quad \forall n \ge 0. \end{cases}$$

The Midpoint Method: Error Analysis

Proof (continued): We show by induction $|e_n| \leq |a_n|$.

1. First step: $|e_0| \leq |a_0|$
2. Assuming $e_i \leq a_i, \forall i \leq n$, we have

$$|e_{n+1}| \leq \underbrace{a_{n-1}}_{\leq a_n} + 2h \left\| \frac{\partial f}{\partial x} \right\|_\infty |a_n| + h\|\tau\|_\infty$$

$$\leq a_n \left(1 + 2h \left\| \frac{\partial f}{\partial x} \right\|_\infty \right) + h\|\tau\|_\infty \equiv a_{n+1}.$$

The Midpoint Method: Error Analysis

Proof (continued):

Applying the definition of a_n recursively:

$$\begin{aligned} a_n &= \left(1+2h\left\|\frac{\partial f}{\partial x}\right\|_\infty\right)a_{n-1} + h\|\tau\|_\infty \\ &= \left(1+2h\left\|\frac{\partial f}{\partial x}\right\|_\infty\right)^2 a_{n-2} + h\|\tau\|_\infty\left(1+2h\left\|\frac{\partial f}{\partial x}\right\|_\infty\right) = \cdots \\ &= \left(1+2h\left\|\frac{\partial f}{\partial x}\right\|_\infty\right)^n a_0 + h\|\tau\|_\infty\left[1+\left(1+2h\left\|\frac{\partial f}{\partial x}\right\|_\infty\right)+\cdots\right. \\ &\qquad \left.\cdots+\left(1+2h\left\|\frac{\partial f}{\partial x}\right\|_\infty\right)^{n-1}\right] \\ &\le e^{2hn\|\frac{\partial f}{\partial x}\|_\infty} a_0 + h\|\tau\|_\infty \frac{\left(1+2h\left\|\frac{\partial f}{\partial x}\right\|_\infty\right)^n - 1}{2h\|\frac{\partial f}{\partial x}\|_\infty}. \end{aligned}$$

■

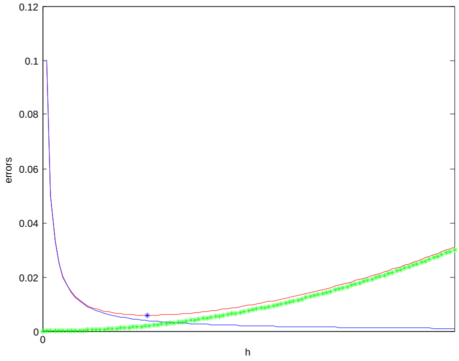
Remark: Only difference in the proof is the sequence $\{a_n\}_n$ because of the 2-step method.

Influence of finite arithmetic

Let ρ_n be the roundoff error at a given step. A similar proof as for the Forward Euler:

$$\max_{0\leq n\leq N}|e_n| \leq e^{2\|\frac{\partial f}{\partial x}\|_\infty (T-t_0)}\max(|e_0|,|e_1|) + \frac{e^{2\|\frac{\partial f}{\partial x}\|_\infty (T-t_0)}-1}{2\left\|\frac{\partial f}{\partial x}\right\|_\infty}\left(\underbrace{\frac{h^2}{3}\|x'''\|_\infty}_{\text{dominant term}} + \underbrace{\frac{\|\rho\|_\infty}{h}}_{\text{finite arithmetic error}}\right)$$

Influence of finite arithmetic



Asymptotic Error Formula

Assume $x \in C^4(t_0, T)$, then $e_n = x(t_n) - x_n = hu(t_n) + O(h^3)$ where u s the solution to IVP

$$\begin{cases} u'(t) = \frac{\partial f}{\partial x}(t, x(t))u(t) + \frac{1}{6}x'''(t) \\ u(t_0) = u_0 \end{cases}$$

Using Richardson extrapolation (2 sequences obtained on 2 meshes - one a refinement of the other) we can obtain a more accurate numerical solution.

- mesh "h": $x(t_n) = x_n + h^2 u(t_n) + O(h^3)$
- mesh "$\frac{h}{2}$": $x(t_n) = \tilde{x}_m + \left(\frac{h}{2}\right)^2 u(t_n) + O(h^3) | \cdot (-4)$:

$$-3x(t_n) = x_n - 4\tilde{x}_m + O(h^3) \;\Rightarrow\; x(t_n) = \frac{-x_n + 4\tilde{x}_m}{3} + O(h^3)$$

Error here is $O(h^3)$ instead of $O(h^2)$.

The Midpoint Method is weakly stable

$$\begin{cases} x' = \lambda x \\ x(0) = 1 \end{cases} \Rightarrow \underbrace{|x(t_{n+1})| \leq |x(t_n)|}_{\text{for this we need } Re(\lambda) \leq 0} \text{ and expect } |x_{n+1}| \leq |x_n|.$$

Apply the Midpoint Method to this ODE:

$$x_{n+1} = x_{n-1} + 2h\lambda x_n$$

This is a finite difference equation. We look for solutions of a particular form: $x_n = r^n$.

$$r^{n+1} - r^{n-1} - 2h\lambda r^n = 0, \qquad r^2 - 2h\lambda r - 1 = 0$$

$$r_0 = h\lambda + \sqrt{1 + h^2\lambda^2}, \quad r_1 = h\lambda - \sqrt{1 + h^2\lambda^2}$$

Weak stability of the midpoint method

General solution: $x_n = \beta_0 r_0^n + \beta_1 r_1^n, \quad n \geq 0$

where β_0, β_1 are given by the initial conditions:

$$\begin{array}{ll} n=0: & x_0 = \beta_0 + \beta_1 \\ n=1: & x_1 = \beta_0 r_0 + \beta_1 r_1 \end{array} \quad \Rightarrow \beta_0 = \frac{x_1 - r_1 x_0}{r_0 - r_1}, \beta_1 = \frac{x_0 r_0 - x_1}{r_0 - r_1},$$

Take $x_0 = 1, x_1 = e^{\lambda h}$ (recall $x(t) = e^{\lambda t}$) to get:

$$\beta_0 = \frac{e^{\lambda h} - \lambda h\left(1 - \sqrt{1 + \frac{1}{\lambda^2 h^2}}\right)}{2\sqrt{\lambda^2 h^2 + 1}} = \frac{e^{\lambda h} - \lambda h}{2\sqrt{\lambda^2 h^2 + 1}} + \frac{1}{2} \xrightarrow[h \to 0]{} 1,$$

$$\beta_1 \to 0.$$

- If $0 < \lambda < \infty$: $r_0 > |r_1| > 0 \Rightarrow r_1^n$ increases less rapidly than r_0^n, the correct term. $\beta_0 r_0^n$ dominates.
- If $\lambda < 0, 0 < r_0 < 1, r_1 < -1 \Rightarrow \beta_1 r_1^n$ dominates as n increases, hence the numerical solution diverges from the true solution as $t_n \to \infty$. **Midpoint method is weakly stable**.

The original equation

$$x' = \lambda x$$

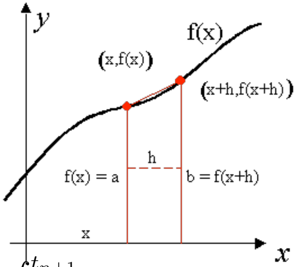
has a one-parameter family of solutions depending on $x(t_0)$, while the midpoint approximation

$$x_{n+1} = x_{n-1} + 2h\lambda x_n$$

has the two-parameter family of solutions $x_n = \beta_0 r_0^n + \beta_1 r_1^n$ depending on x_0 and x_1.
The parasitic solution $\beta_1 r_1^n$ is a creation of the numerical method.

The Trapezoidal Method: derivation

$$x' = f(t,x) \Rightarrow \int_{t_n}^{t_{n+1}} x'dt = \int_{t_n}^{t_{n+1}} f(t,x)dt$$



$$x(t_{n+1}) - x(t_n) = \int_{t_n}^{t_{n+1}} f(t,x)dt$$

$$\simeq \frac{t_{n+1}-t_n}{2}\Big(f(t_n, x(t_n)) - f(t_{n+1}, x(t_{n+1}))\Big)$$

$$\overset{x(t_n)\leftarrow x_n}{\Longrightarrow} \quad x_{n+1} - x_n = \frac{h_{n+1}}{2}\Big(f(t_n,x_n) - f(t_{n+1},x_{n+1})\Big)$$

The Trapezoidal Method: derivation

Trapezoidal Method

$$\begin{cases} x_{n+1} = x_n + \frac{h}{2}\Big(f(t_n, x_n) - f(t_{n+1}, \underbrace{x_{n+1}}_{???})\Big) & n \geq 1 \\ x_0 - \text{needed} \end{cases}$$

x_0 is given $\Rightarrow$ **one-step method**, but is **non-linear** in x_{n+1}, **implicit**.

If f is linear in x, i.e. $f(t,x) = \lambda x$, then we are back to a linear problem. But if not, i.e. $f(t,x) = \lambda x^2$, then....

The Trapezoidal Method: solving nonlinear equations

At each time step we need to solve for x_{n+1}:

$$\begin{aligned} & x_{n+1} - x_n - \frac{h}{2} f(t_n, x_n) - \frac{h}{2} f(t_{n+1}, x_{n+1}) = 0 \\ \Leftrightarrow\ & F(x_{n+1}) = 0 \end{aligned}$$

1. Functional Iteration (Picard)
2. Newton's Method

Solving nonlinear equations: Functional Iteration (Picard)

We start with a good guess for x_{n+1}: $x_{n+1}^{(0)}$
and construct a sequence $x_{n+1}^{(1)}, x_{n+1}^{(2)}, \ldots$ by

$$x_{n+1}^{(j+1)} = \underbrace{x_n}_{\text{fixed}} + \frac{h}{2} f(t_n, \underbrace{x_n}_{\text{fixed}}) + \frac{h}{2} f(t_{n+1}, \underbrace{x_{n+1}^{(j)}}_{\text{previous iterate}}) \quad : \text{corrector}$$

and want: $\lim_{j\to\infty} x_{n+1}^{(j)} = x_{n+1}$.

$$\begin{aligned} x_{n+1} - x_{n+1}^{(j+1)} &= x_n + \frac{h}{2} f(t_n, x_n) + \frac{h}{2} f(t_{n+1}, x_{n+1}) \\ &\quad - \left(x_n + \frac{h}{2} f(t_n, x_n) + \frac{h}{2} f(t_{n+1}, x_{n+1}^{(j)}) \right) \\ &= \frac{h}{2} \Big(\underbrace{f(t_{n+1}, x_{n+1}) - f(t_{n+1}, x_{n+1}^{(j)})}_{= \frac{\partial f}{\partial x}(t_{n+1}, \xi_{n+1})(x_{n+1} - x_{n+1}^{(j)})} \Big) \end{aligned}$$

Solving nonlinear equations: Functional Iteration (Picard)

$$\Longrightarrow \qquad |x_{n+1} - x_{n+1}^{(j+1)}| \leq \frac{h}{2} \left\| \frac{\partial f}{\partial x} \right\|_\infty |x_{n+1} - x_{n+1}^{(j)}| \tag{10.9}$$

So for the iteration to converge we need

$$\frac{h}{2} \left\| \frac{\partial f}{\partial x} \right\|_\infty < 1$$

which gives a constraint on h.

Solving nonlinear equations: Newton-Raphson Method

Produces iteratively a sequence of approximations to the zeros, with quadratic rate of convergence.
Start with $x_{n+1}^{(0)}$ a **good** initial guess of x_{n+1} (otherwise convergence may fail to occur)
and construct a sequence : $x_{n+1}^{(1)}, x_{n+1}^{(2)}, \dots$ by

$$x_{n+1}^{(j+1)} = x_{n+1}^{(j)} - \left(F'(x_{n+1}^{(j)})\right)^{-1} F(x_{n+1}^{(j)}) \qquad \text{(solving } F(x_{n+1}) = 0\text{)}$$

In our case

$$F(z) = z - x_n - \frac{h}{2} f(t_n, x_n) - \frac{h}{2} f(t_{n+1}, z)$$

hence

$$F'(z) = 1 - \frac{h}{2} \frac{\partial f}{\partial x}(t_{n+1}, z)$$

Solving nonlinear equations: Newton-Raphson Method

Take $x_{n+1}^{(0)} = x_n$ or use Picard iteration.

Algorithm

- Loop over the time steps n

  $x_{n+1} = x_{n+1}^{(\text{final } j)}$
- Stopping criteria for Newton Loop:

$$|x_{n+1}^{(j+1)} - x_{n+1}^{(j)}| \le \text{Tolerance}$$

Error Analysis

Theorem

The local truncation error for the trapezoidal method is $O(h^2)$.

local truncation error:

$$\tau_n := \frac{x(t_{n+1}) - x(t_n)}{h} - \frac{1}{2}\Big(f(t_n, x(t_n)) + f(t_{n+1}, x(t_{n+1}))\Big)$$

Proof: Doing Taylor expansions around $t_{n+1/2} = \frac{t_n + t_{n+1}}{2}$:

$$x(t_{n+1}) = x(t_{n+1/2}) + \tfrac{h}{2}x'(t_{n+1/2}) + \tfrac{h^2}{8}x''(t_{n+1/2}) + \tfrac{h^3}{48}x'''(t_{n+1/2}) + O(h^4)$$

$$x(t_n) = x(t_{n+1/2}) - \tfrac{h}{2}x'(t_{n+1/2}) + \tfrac{h^2}{8}x''(t_{n+1/2}) - \tfrac{h^3}{48}x'''(t_{n+1/2}) + O(h^4)$$

$$\frac{x(t_{n+1}) - x(t_n)}{h} = x'(t_{n+1/2}) + \frac{h^2}{24}x'''(t_{n+1/2}) + \underbrace{O(h^3)}_{\text{could be } O(h^4)} \tag{10.10}$$

Error Analysis

Proof (continued): Repeat the Taylor expansion for x' instead of x:

$$\begin{array}{rl} x'(t_{n+1}) = & x'(t_{n+1/2}) + \frac{h}{2}x''(t_{n+1/2}) + \frac{h^2}{8}x'''(t_{n+1/2}) + \frac{h^3}{48}x''''(t_{n+1/2}) + O(h^4) \\ x'(t_n) = & x'(t_{n+1/2}) - \frac{h}{2}x''(t_{n+1/2}) + \frac{h^2}{8}x'''(t_{n+1/2}) - \frac{h^3}{48}x''''(t_{n+1/2}) + O(h^4) \\ \hline \end{array}$$

$$x'(t_n) + x'(t_{n+1}) = 2x'(t_{n+1/2}) + \frac{h^2}{4}x'''(t_{n+1/2}) + O(h^4) \quad (10.11)$$

x is the exact solution, so:

$$f(t_n, x(t_n)) + f(t_{n+1}, x(t_{n+1})) = x'(t_n) + x'(t_{n+1}).$$

From (10.10), (10.11) and above:

Local truncation error

$$\tau_n = \frac{h^2}{24}x'''(t_{n+1/2}) - \frac{h^2}{8}x'''(t_{n+1/2}) + O(h^3) = O(h^2)$$

F.Euler: $O(h)$, Midpoint: $O(h^2)$, Trapezoidal: $O(h^2)$

Convergence Theorem

Theorem

$$\max_{0\le n\le N}|x(t_n)-x_n| \le e^{2K(T-t_0)}\underbrace{|x(t_0)-x_0|}_{=0 \text{ in most cases}}+\frac{e^{2K(T-t_0)}-1}{2K}\frac{h^2}{12}\|x'''\|_\infty$$

where $K = \|\frac{\partial f}{\partial x}\|_\infty$.

Proof: The local truncation error

$$\begin{aligned} h\tau_n &= x(t_{n+1})-x(t_n)-\frac{h}{2}\Big(f(t_n,x(t_n))+f(t_{n+1},x(t_{n+1})\Big)\\ &= \int_{t_n}^{t_{n+1}}\left[x'(t)-\Big(\frac{x'(t_{n+1})-x'(t_n)}{h}(t-t_n)+x'(t_n)\Big)\right]dt\\ &= -\frac{h^3}{12}x'''(\xi_n) \qquad \text{(see Chapter 3. Interpolation Theory)}\end{aligned}$$

Convergence Theorem

Proof (continued):

$$x(t_{n+1}) = x(t_n) + \frac{h}{2}\left(f(t_n, x(t_n) + f(t_{n+1}, x(t_{n+1}))\right) - \frac{h^3}{12}x'''(\xi_n)$$

$$x_{n+1} = x_n + \frac{h}{2}\left(f(t_n, x_n + f(t_{n+1}, x_{n+1})\right) \qquad | \cdot (-1)$$

$$e_{n+1} = e_n + \frac{h}{2}\Bigg(\underbrace{f(t_n, x(t_n) - f(t_n, x_n)}_{\frac{\partial f}{\partial x}(t_n,\zeta_n)e_n}$$

$$+ \underbrace{f(t_{n+1}, x(t_{n+1})) + f(t_{n+1}, x_{n+1})}_{\frac{\partial f}{\partial x}(t_{n+1},\chi_{n+1})e_{n+1}}\Bigg) - \frac{h^3}{12}x'''(\xi_n)$$

$$\Rightarrow |e_{n+1}| \le |e_n| + \frac{h}{2}K|e_n| + \frac{h}{2}K|e_{n+1}| + \frac{h^3}{12}\|x'''\|_\infty$$

Convergence Theorem

Proof (continued): Defining $f_n := \max\limits_{0\le i\le n} |e_i| \Rightarrow f_n \le f_{n+1}$ we get

$$\begin{aligned}|e_{n+1}| &\le f_n + \frac{hK}{2} f_{n+1} + \frac{hK}{2} f_{n+1} + \frac{h^3}{12}\|x'''\|_\infty = f_n + hK f_{n+1} + \frac{h^3}{12}\|x'''\|_\infty \\ &\Rightarrow f_{n+1} \le f_n + hK f_{n+1} + \frac{h^3}{12}\|x'''\|_\infty \\ &\Leftrightarrow (1-hK) f_{n+1} \le f_n + \frac{h^3}{12}\|x'''\|_\infty .\end{aligned}$$

I f we choose h small enough so that $hk \le \frac{1}{2} \Leftrightarrow \frac{1}{1-hK} \le 1 + 2hK$:

$$f_{n+1} \le (1+2hK) f_n + (1+2hK)\frac{h^3}{12}\|x'''\|_\infty$$

and the proof follows as before (see F.Euler and Midpoint Method). ■

Stability

Stability

- Trapezoidal M. - Unconditionally Stable,
- Midpoint M. - Weakly Stable,
- F. Euler M. - Stable w/a constraint

Apply the method to $\begin{cases} x' = \lambda x \\ x(0) = 1 \end{cases}$

$$x_{n+1} = x_n + \frac{h}{2}\Big(\lambda x_n + \lambda x_{n+1}\Big) \Leftrightarrow \left(1 - \frac{\lambda h}{2}\right) x_{n+1} = x_n \left(1 + \frac{\lambda h}{2}\right)$$

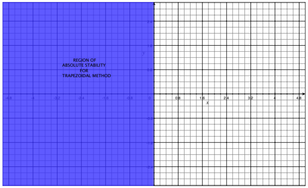
Since we want $|x_{n+1}| \le |x_n|$ it is sufficient that

$$\frac{\left|1 + \frac{h\lambda}{2}\right|}{\left|1 - \frac{h\lambda}{2}\right|} \le 1 \qquad \overset{z=\lambda h}{\Longleftrightarrow} \qquad \left|1 + \frac{z}{2}\right| \le \left|1 - \frac{z}{2}\right|, \quad (z = a + ib)$$

Stability

$$\left(1+\frac{a}{2}\right)^2+\left(\frac{b}{2}\right)^2 \le \left(1-\frac{a}{2}\right)^2+\left(\frac{b}{2}\right)^2,$$

$$\Leftrightarrow \left|1-\frac{a}{2}\right| \le \left|1+\frac{a}{2}\right| \Leftrightarrow a \le 0 \qquad \text{(independent of } b\text{)}$$



Definition

If the region of absolute stability contains the entire half-plane $(t \le 0)$, then the method is called $A-$stable.

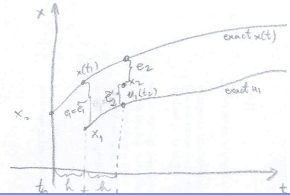
Local error vs. Global error

Definition

The local error at t_n is $\tilde{e}_n = u_{n-1}(t_n) - x_n$ where u_{n-1} is the exact solution of

$$\begin{cases} u'_{n-1} = f(t, u_{n-1}) \\ u_{n-1}(t_{n-1}) = x_{n-1}. \end{cases} \qquad \text{(error from \underline{one} step)}$$

$$\tilde{e}_1 = u_0(t_1) - x_1, \qquad \left.\begin{array}{l} u'_0 = f(t, u_0) \\ u(t_0) = x_0 \end{array}\right\} \Rightarrow u_0 = x \Rightarrow \underbrace{\tilde{e}_1}_{\text{local}} = \underbrace{e_1}_{\text{global}}$$



Local error vs. Global error

Proposition

We know that $e_n = O(h^2)$, but we can show that $\tilde{e}_n = O(h^3)$.

Proof: Doing a Taylor expansion of u_n at t_{n+1} we have:

$$u_n(t_{n+1}) = u_n(t_n) + \frac{h}{2}\Big(f(t_n, \overbrace{\underbrace{u_n(t_n)}_{\text{(by def.)}}}^{=x_n}) + f(t_n, u_n(t_{n+1})\Big) - \frac{h^3}{12}u'''(t_n) + O(h^4$$

$$x_{n+1} = x_n + \frac{h}{2}\Big(f(t_n, x_n) + f(t_{n+1}, x_{n+1})\Big)$$

$$\tilde{e}_{n+1} = \frac{h}{2}\frac{\partial f}{\partial x}(t_{n+1}, \xi_{n+1})\tilde{e}_{n+1} - \frac{h^3}{12}u_n'''(t_n) + O(h^4)$$

hence for sufficiently small h, $\tilde{e}_{n+1} = O(h^3)$:

$$u_n(t_{n+1}) - x_{n+1} = -\frac{h^3}{12}u_n'''(t_n) + O(h^4) \qquad (10.12)$$

i.e., the local error is the truncation error.

The computation of x_{n+1} to x_n contains a truncation error that is $O(h^3)$. To maintain it, $x_{n+1}^{(i)}$ should satisfy $|x_{n+1} - x_{n+1}^{(i)}| = O(h^3)$ or

$$|x_{n+1} - x_{n+1}^{(i)}| = O(h^4) \tag{10.13}$$

if we want the iteration error to be less significant.

⋆ If the initial guess $x_{n+1}^{(0)}$ is computed with the Euler method:

$$x_{n+1}^{(0)} = x_n + hf(t_n, x_n) \qquad : \text{ predictor}$$

$$u_n(t_{n+1}) = \underbrace{u_n(t_n)}_{x_n} + h\underbrace{u_n'(t_n)}_{f(t_n,x_n)} + \frac{h^2}{2}u''(\xi_n)$$

$$\Rightarrow u_n(t_{n+1}) - x_{n+1}^{(0)} = \frac{h^2}{2}u''(\xi_n) \tag{10.14}$$

$$\overset{(10.12),(10.14)}{\Longrightarrow} x_{n+1} - x_{n+1}^{(0)} = O(h^2), \tag{10.15}$$

hence to satisfy (10.13), the Lipschitz bound (10.9) implies that **two** iterates have to be computed: $x_{n+1}^{(2)} \equiv x_{n+1}$.

$\star\star$ If the initial guess $x_{n+1}^{(0)}$ is computed with the midpoint method:

$$x_{n+1}^{(0)} = x_{n-1} + 2hf(t_n, x_n)\text{: predictor}$$

$$u_n(t_{n+1}) = u_n(t_{n-1}) + 2h\,\underbrace{u_n'(t_n)}_{f(t_n,x_n)} + \frac{h^3}{3}u'''(\eta_n)$$

$$\Rightarrow u_n(t_{n+1}) - x_{n+1}^{(0)} = \underbrace{u_n(t_{n-1}) - x_{n-1}}_{\frac{h^3}{12}u_n'''(t_n)+O(h^4),\text{ as for }(10.12)} + \frac{h^3}{3}u'''(\eta_n)$$

$$+\frac{5h^3}{12}u'''(\eta_n) + O(h^4) \qquad (10.16)$$

$$\overset{(10.12),(10.16)}{\Longrightarrow} x_{n+1} - x_{n+1}^{(0)} = \frac{h^3}{2}u'''(\eta_n) + O(h^4), \qquad (10.17)$$

hence just **one** iteration is sufficient to satisfy $O(h^4)$ bound

Runge-Kutta methods: general form

Solving

$$x' = f(t, x)$$

using the Taylor series method requires offline calculations of

$$x'', x''', \cdots$$

Runge-Kutta methods provide codes to solve the equation involving only evaluations of f.

General form

$$\begin{cases} x_{n+1} = x_n + h\psi(h, t_n, x_n, f) \\ x_0 \text{ given} \end{cases}$$

Linear truncation error: $\tau_n = \frac{x(t_{n+1}) - x(t_n)}{h} - \psi(h, t_n, x(t_n), f)$

Runge-Kutta methods: general form

Example

- Forward Euler: $\psi(h, t_n, x_n, f) = f(t_n, x_n)$
- Explicit 2^{nd} order RK:

$$\psi(h, t_n, x_n, f) = \gamma_1 f(t_n, x_n) + \gamma_2 f(t_n + \alpha h, x_n + \beta h f(t_n, x_n)) \quad (10.18)$$

(10.18) gives an "average" derivative, $\alpha \in [0, 1]$.

How should we choose $\gamma_1, \gamma_2, \alpha, \beta$?

Convergence theorem

Theorem

Assume ψ is Lipschitz in x_n:

$$|\psi(h,t,\xi,f) - \psi(h,t,\eta,f)| \le K|\xi - \eta| \qquad \forall \xi, \eta.$$

Then

$$\max_{0\le n\le N} |x(t_n) - x_n| \le e^{K(t_n-t_0)}|x(t_0) - x_0| + \frac{e^{K(t_n-t_0)} - 1}{K}\|\tau\|_\infty.$$

Proof:

$$\begin{aligned} x(t_{n+1}) &= x(t_n) + h\psi(h,t_n,x(t_n),f) + h\tau_n \\ x_{n+1} &= x_n + h\psi(h,t_n,x_n,f) \\ \hline e_{n+1} &= e_n + h\Big(\psi(\ldots x(t_n)\ldots) - \psi(\ldots x_n \ldots)\Big) + h\tau_n \end{aligned}$$

hence

$$|e_{n+1}| \le |e_n|(1 + hK) + h\tau_n$$

as in the Forward Euler proof. ■

Taylor series for $f(x,y)$

Taylor series in two variables:

$$f(x+h,y+k) = \sum_{i=0}^{\infty} \frac{1}{i!}\left(h\frac{\partial}{\partial x} + k\frac{\partial}{\partial y}\right)^i f(x,y)$$

For example
$$\begin{cases} \left(h\frac{\partial}{\partial x} + k\frac{\partial}{\partial y}\right)^0 f(x,y) = f \\ \left(h\frac{\partial}{\partial x} + k\frac{\partial}{\partial y}\right)^1 f(x,y) = h\frac{\partial f}{\partial x} + k\frac{\partial f}{\partial y} \\ \left(h\frac{\partial}{\partial x} + k\frac{\partial}{\partial y}\right)^2 f(x,y) = h^2\frac{\partial^2 f}{\partial x^2} + 2hk\frac{\partial^2 f}{\partial x \partial y} + k^2\frac{\partial^2 f}{\partial y^2} \end{cases}$$

which yields

$$\begin{aligned} f(x+h,y+k) =& f + (hf_x + kf_y) \\ & + \frac{1}{2!}\left(h^2 f_{xx} + 2hkf_{xy} + k^2 f_{yy}\right) \\ & + \frac{1}{3!}\left(h^3 f_{xxx} + 3h^2kf_{xxy} + 3hk^2 f_{xyy} + k^3 f_{yyy}\right) \\ & \vdots \end{aligned}$$

Runge-Kutta method of order 2

RK2 (implicit, trapezoidal method);
it involves **two function evaluations**:

$$K_1 = hf(t,x)$$
$$K_2 = hf(t+\alpha h, x+\beta K_1)$$

and the formula that gives $x(t+h)$ is

$$\begin{aligned} x(t+h) &= x(t) + w_1 K_1 + w_2 K_2 \\ &= x(t) + \boldsymbol{w_1} \underbrace{hf(t,x)}_{K_1} + \boldsymbol{w_2} \underbrace{hf(t+\boldsymbol{\alpha} h, x + \boldsymbol{\beta} hf(t,x))}_{K_2} \end{aligned}$$

The goal is to *reproduce as many terms as possible* in the Taylor series

$$x(t+h) = x(t) + hx'(t) + \frac{1}{2!}h^2x''(t) + \frac{1}{3!}h^3x'''(t) + \cdots$$

Runge-Kutta method of order 2

One way is to have agreement through the term in $\boldsymbol{h}$

$$x(t+h) = x(t) + \boldsymbol{w_1} \underbrace{hf(t,x)}_{K_1} + \boldsymbol{w_2} \underbrace{hf(t+\boldsymbol{\alpha} h, x + \boldsymbol{\beta} hf(t,x))}_{K_2}$$

$$= x(t) + h \underbrace{x'(t)}_{=f(t,x)} + \frac{1}{2!}h^2 x''(t) + \frac{1}{3!}h^3 x'''(t) + \cdots$$

is to set

$$w_1 = 1 \quad \text{and} \quad w_2 = 0$$

which is Euler's method.

Runge-Kutta method of order 2

To have agreement through the term in $\boldsymbol{h^2}$

$$x(t+h) = x(t) + \boldsymbol{w_1} \underbrace{hf(t,x)}_{K_1} + \boldsymbol{w_2} \underbrace{hf(t+\boldsymbol{\alpha} h, x + \boldsymbol{\beta} hf(t,x))}_{K_2}$$

$$= x(t) + h \underbrace{x'(t)}_{=f} + \frac{1}{2!}h^2 \underbrace{x''(t)}_{=f_t + ff_x} + \mathcal{O}(h^3)$$

apply the two-variables Taylor series

$$f(t+\alpha h, x+\beta hf) = f + \alpha h f_t + \beta h f f_x + \underbrace{\frac{1}{2}\left(\alpha h \frac{\partial}{\partial t} + \beta h f \frac{\partial}{\partial x}\right)^2 f(\hat{x},\hat{y})}_{\text{to be neglected}}$$

Runge-Kutta method of order 2

Therefore we have agreement through the term in $\boldsymbol{h^2}$

$$\begin{aligned} x(t+h) &= x(t) + \boldsymbol{w_1} hf + \boldsymbol{w_2} h \cdot (f + \alpha h f_t + \beta h f f_x) \\ &= x(t) + hf + \frac{1}{2!}h^2 f_t + \frac{1}{2!}h^2 f f_x + \mathcal{O}(h^3) \end{aligned}$$

provided that

$$w_1 + w_2 = 1, \quad \alpha w_2 = \frac{1}{2}, \quad \beta w_2 = \frac{1}{2}$$

which holds for

$$\alpha = 1, \quad \beta = 1, \quad w_1 = \frac{1}{2}, \quad w_2 = \frac{1}{2}$$

Runge-Kutta method of order 2

The **second-order Runge-Kutta method:**

$$x(t+h) = x(t) + \frac{h}{2} f(t,x) + \frac{h}{2} f(t+h, x+hf(t,x))$$

or equivalently

$$\boldsymbol{x(t+h) = x(t) + \frac{1}{2}(K_1 + K_2)}$$

with

$$\begin{cases} \boldsymbol{K_1 = hf(t,x)} \\ \boldsymbol{K_2 = hf(t+h, x+K_1)} \end{cases}$$

The solution at $t+h$ is computed with **two** evaluations of f, with the error of order $\mathcal{O}(h^3)$.

Runge-Kutta method of order 2

We have agreement through up to $\boldsymbol{h^2}$ in

$$\begin{aligned} x(t+h) &= x(t) + \boldsymbol{w_1} hf + \boldsymbol{w_2} h \cdot (f + \alpha h f_t + \beta h f f_x) \\ &= x(t) + hf + \frac{1}{2!}h^2 f_t + \frac{1}{2!}h^2 f f_x + \mathcal{O}(h^3) \end{aligned}$$

if

$$w_1 + w_2 = 1, \quad \alpha w_2 = \frac{1}{2}, \quad \beta w_2 = \frac{1}{2}$$

which holds also for

$$\beta = \alpha, \quad w_1 = 1 - \frac{1}{2\alpha}, \quad w_2 = \frac{1}{2\alpha}$$

Runge-Kutta method of order 4

The **fourth-order Runge-Kutta method:**

$$x(t+h) = x(t) + \tfrac{1}{6}\left(K_1 + K_2 + K_3 + K_4\right)$$

where

$$\begin{cases} K_1 = hf(t, x) \\ K_2 = hf(t + \dfrac{1}{2}h, x + \dfrac{1}{2}K_1) \\ K_3 = hf(t + \dfrac{1}{2}h, x + \dfrac{1}{2}K_2) \\ K_4 = hf(t + h, x + K_3) \end{cases}$$

The solution at $t + h$ is computed with **four** evaluations of f, and the formula agrees with the Taylor expansion up to the term in h^4.

Review

1. The **second order Runge-Kutta method** is

$$x(t+h) = x(t) + \frac{1}{2}(K_1 + K_2)$$

with

$$\begin{cases} K_1 = hf(t,x) \\ K_2 = hf(t+h, x+K_1) \end{cases}$$

The method requires **two** evaluations of f. It equivalent to a Taylor series method of order 4 and it has truncation error of order $\mathcal{O}(h^3)$.

2. The **fourth order Runge-Kutta method**

$$x(t+h) = x(t) + \tfrac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4)$$

where

$$\begin{cases} K_1 = hf(t,x) \\ K_2 = hf(t+\frac{1}{2}h, x+\frac{1}{2}K_1) \\ K_3 = hf(t+\frac{1}{2}h, x+\frac{1}{2}K_2) \\ K_4 = hf(t+h, x+K_3) \end{cases}$$

It is a one of the most popular single-step methods for solving ODEs, it needs **four** evaluations of the function f per step. It equivalent to a Taylor series method of order 4 and it has truncation error of order $\mathcal{O}(h^5)$.

Variable step-size RK

- Reason for change the step size h during computations: a solution that moves between periods of **slow** and **fast** change.
- To use the fixed step size h small enough to track accurately the fast changes could imply in *slow solve for the rest of the solution*.

Idea: monitor the error estimate e_n (or relative error estimate $\frac{e_n}{|x_n|}$) produced by the current step s.t.

$$e_n \leq \texttt{error tolerance}.$$

The method:

1. if the `tolerance` is exceeded, reject the step and repeat step with $h = \frac{h}{2}$
2. if the error is met, accept the step and choose h appropriate for next step (for example, double it)

Key question: how to approximate the error made on each step (w/o large amount of extra computation)?

Error control

We want to estimate the local error (91)

$$x(t_n + 2h) \text{ relative to } u_n(t_n + 2h),$$

where u_n is the solution to $x' = f(t, x), t \geq t_n$, passing through (t_n, x_n).
It can be shown that

$$u_n(t_n + 2h) - x_h(t_n + 2h) \simeq \frac{1}{2^p - 1}\Big(\underbrace{x_h(t_n + 2h) - x_{2h}(t_n + 2h)}_{trunc}\Big)$$

We want $trunc$ to satisfy a local error control per unit stepsize:

$$.5\varepsilon h \leq |trunc| \leq 2\varepsilon h,$$

and when this is violated, we seek $\hat{h}$ s.t. $trunc = \varepsilon \hat{h}$ and continue with the solution process.

Cost in function evaluations

For the 4th order RK:
$$\left|\begin{array}{l} x_{n+1} = x_n + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4) \\ \quad K_1 = f(t_n, x_n) \\ \quad K_2 = f(t_n + \frac{h}{2}, x_n + \frac{h}{2}K_1) \\ \quad K_3 = f(t_n + \frac{h}{2}, x_n + \frac{h}{2}K_2) \\ \quad K_4 = f(t_n + h, x_n + hK_3) \end{array}\right.$$

to go from t_n to $t_n + 2h$ we need

- 4 evaluations for $x_h(t_n + h)$,
- 4 evaluations for $x_h(t_n + 2h)$, and
- 3 more to get $x_{2h}(t_n + 2h)$,

a total of 11.

Fairly expansive compared to (implicit) multistep methods, but a variable-stepsize RK is stable, reliable and easy to program.

Embedded RK Methods

The most widely way for obtaining an error estimate: run in parallel

- a higher order ODE solver and
- the ODE solver of interest.

The higher order method's estimate $\hat{x}_{n+1}$ for x_{n+1} is signifficantly more accurate than the original x_{n+1}, hence the difference

$$\delta_n = \left|x_{n+1} - \hat{x}_{n+1}\right|$$

is used as an error estimate for the current step.
Several "pairs" of RK methods, one of order p and another of order $p+1$ have been developed that share much of the needed calculations, so the cost of extra step-size control is kept low: **embedded RK pairs**.

Embedded RK Methods

The truncation error is computed comparing the result x_{n+1} with one from a higher order RK, $\hat{x}_{n+1}$, and used as an estimate in the lower order method.

$$\begin{array}{c|l} c & A \\ \hline & b \quad \longleftarrow \text{gives a method of order } p \\ \hline & \hat{b} \quad \longleftarrow \text{gives a method of order } p+1 \end{array}$$

$$\begin{aligned} &1 \le i \le s, \quad K_i = \cdots \\ &x_{n+1} = x_n + h \textstyle\sum_{i=1}^{s} b_i f(t_n + c_i h, K_i) \\ &\hat{x}_{n+1} = \hat{x}_n + h \textstyle\sum_{i=1}^{s} \hat{b}_i f(t_n + c_i h, K_i) \end{aligned}$$

Let $\delta_{n+1} = x_{n+1} - \hat{x}_{n+1}$, an estimate of the local error. With a fixed `tolerance` we want:

$$\frac{\texttt{tolerance}}{8h(T-t_0)} \le |\delta_{n+1}| \le \frac{\texttt{tolerance}}{8h(T-t_0)}$$

If this fails, then choose

$$h_{new} = \alpha h \left(\frac{\texttt{tolerance}}{|\delta_{n+1}|} \right)^{\frac{1}{p+1}}, \qquad (10.19)$$

where α is a "safety" parameter. (e.g., $\alpha = 8, .9$)

Embedded Methods: RK order2/order3 pair

The explicit Trapezoidal Method can be paired with a 3^{rd} order RK method to make an embedded pair suitable for step size control:

$$x_{n+1} = x_n + h\frac{V_1 + V_2}{2} \qquad \textit{(trapezoid step)}$$

$$\hat{x}_{n+1} = x_n + h\frac{V_1 + V_2 + 4V_3}{6} \qquad \textit{(3^{rd} order, 3-stages RK)}$$

$$\text{where } \begin{cases} V_1 = f(t_n, x_n) \\ V_2 = f(t_n + h, x_n + hV_1) \\ V_3 = f(t_n + \frac{h}{2}, x_n + \frac{h}{2}\frac{V_1+V_2}{2}) \end{cases}$$

The estimate for the error:

$$\delta_n \simeq |x_{n+1} - \hat{x}_{n+1}| = \left| h\frac{V_1 + V_2 - 2V_3}{3} \right|$$

is then used as an estimate for the local truncation error to control the step size h.

Embedded Methods: Bogacki-Shampine order2/order3 pair

MATLAB uses a different embedded pair in the `ode23` command.

$$\begin{aligned} V_1 &= f(t_n, x_n) \\ V_2 &= f(t_n + \tfrac{1}{2}h, x_n + \tfrac{1}{2}hV_1) \\ V_3 &= f(t_n + \tfrac{3}{4}h, x_n + \tfrac{3}{4}hV_2) \end{aligned}$$

$$\hat{x}_{n+1} = x_n + \frac{h}{9}(2V_1 + 3V_2 + 4V_3) \quad \textit{(3}^{rd}\textit{ order approximation)}$$

$$V_4 = f(t_n + h, \hat{x}_{n+1}) \qquad \text{(FSAL)}$$

$$x_{n+1} = x_n + \frac{h}{24}(7V_1 + 6V_2 + 8V_3 + 3V_4) \quad \textit{(2}^{rd}\textit{ order approx., despite 4 stages)}$$

The error estimate for step-size control:

$$\delta_n = |x_{n+1} - \hat{x}_{n+1}| = \frac{h}{72}| - 5V_1 + 6V_2 + 8V_3 - 9V_4|.$$

Remark: V_4 becomes V_1 is the step is accepted, hence there are no wasted stages (at least 3 stages are needed for 3^{rd} order RK method).

Embedded RK Methods: RK-Fehlberg 4/5

$$x_{n+1} = x_n + h\sum_{i=1}^{5} b_i V_i \quad \textit{(4th order approximation)}$$

$$\hat{x}_{n+1} = x_n + h\sum_{i=1}^{6} \hat{b}_i V_i \quad \textit{(5th order approximation)}$$

$$\left| \begin{array}{l} V_1 = f(t_n, x_n) \\ V_i = f(t + c_i h, x_n + h\sum_{j=1}^{5} a_{ij} V_j), \qquad i = 2, \ldots, 6. \end{array} \right.$$

$$\begin{array}{c|cccccc}
0 & 0 & 0 & 0 & 0 & 0 & \\
\frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & \\
\frac{3}{8} & \frac{3}{32} & \frac{9}{32} & & & & \\
\frac{12}{13} & \frac{1932}{2197} & -\frac{7200}{2197} & \frac{7296}{2197} & & & \\
1 & \frac{439}{216} & -8 & \frac{3680}{513} & -\frac{845}{4104} & & \\
\frac{1}{2} & -\frac{8}{27} & 2 & -\frac{3544}{2565} & \frac{1859}{4104} & -\frac{11}{40} & \\
\hline
 & \frac{25}{216} & 0 & \frac{1408}{2565} & \frac{2197}{4104} & -\frac{1}{5} & \\
\hline
 & \frac{16}{135} & 0 & \frac{6656}{12825} & \frac{28561}{56430} & -\frac{9}{50} & \frac{2}{55}
\end{array}$$

The error estimate for step-size control is

$$\delta_n = h\left| \frac{1}{360} V_1 - \frac{128}{4275} V_3 - \frac{2197}{75240} V_4 + \frac{1}{50} V_5 + \right.$$

Take the relative test $\frac{\delta_n}{|x_n|} < \texttt{Tolerance}$

and change h using (10.19) with $p = 4$.

Embedded Dormand-Prince order4/order5 method

MATLAB uses a different embedded pair in the ode45 command.

$$\begin{aligned}
V_1 &= f(t_n, x_n)\\
V_2 &= f(t_n + \tfrac{1}{5}h, x_n + \tfrac{1}{5}hV_1)\\
V_3 &= f(t_n + \tfrac{3}{10}h, x_n + \tfrac{3}{40}hV_1 + \tfrac{9}{40}hV_2)\\
V_4 &= f(t_n + \tfrac{4}{5}h, x_n + \tfrac{44}{45}hV_1 - \tfrac{56}{15}hV_2 + \tfrac{32}{9}hV_3)\\
V_5 &= f(t_n + \tfrac{8}{9}h, x_n + \tfrac{19372}{6561}hV_1 - \tfrac{25360}{2187}hV_2 + \tfrac{64448}{6561}hV_3 - \tfrac{212}{729}hV_4)\\
V_6 &= f(t_n + h, x_n + \tfrac{9017}{3168}hV_1 - \tfrac{355}{33}hV_2 + \tfrac{46732}{5247}hV_3 + \tfrac{49}{176}hV_4 - \tfrac{5103}{18656}hV_5)
\end{aligned}$$

$$\hat{x}_{n+1} = x_n + h\left(\frac{35}{384}V_1 + \frac{500}{1113}V_3 + \frac{125}{192}V_4 - \frac{2187}{6784}V_5 + \frac{11}{84}V_6\right)$$ *(5^{th} order approximation)*

$$V_7 = f(t_n + h, \hat{x}_{n+1})$$ (FSAL)

$$x_{n+1} = x_n + h\left(\frac{5179}{57600}V_1 + \frac{7571}{16695}V_3 + \frac{393}{640}V_4 - \frac{92097}{339200}V_5 + \frac{187}{2100}V_6 + \frac{1}{40}V_7\right)$$ *(4^{th} order*

The error estimate for step-size control:

$$\delta_n = |x_{n+1} - \hat{x}_{n+1}| = h\left|\frac{71}{57600}V_1 - \frac{71}{16695}V_3 + \frac{71}{1920}V_4 - \frac{17253}{339200}V_5 + \frac{22}{525}V_6 - \frac{1}{40}V_7\right|.$$

Remark:

1. V_4 becomes V_1 if the step is accepted, hence there are no wasted stages (at least 6 stages are needed for 5^{th} order RK method).
2. For the error control we don't need to compute x_{n+1} if we use local extrapolation (advance with $\hat{x}_{n+1}$).

Multistep methods

Definition: $p+1$ step method

Assume $t_{n+1} - t_n = h, \qquad \forall n$

$$x_{n+1} = \sum_{j=0}^{p} a_j x_{n-j} + h \sum_{j=-1}^{p} b_j \underbrace{f(t_{n-j}, x_{n-j})}_{:=f_{n-j}}, \quad n \geq p \qquad (10.20)$$

$a_0, \ldots, a_p, b_{-1}, \ldots, b_p \in \mathbb{R}, p \geq 0 \ (a_p \neq 0, b_p \neq 0)$.

Euler's method - one-step method:

$$p = 0, \qquad a_0 = 1, b_0 = 1, b_{-1} = 0$$

- **Explicit Methods**: $b_{-1} = 0$, x_{n+1} occurs only on the LHS of (10.20)
- **Implicit Methods**: $b_{-1} \neq 0$, x_{n+1} occurs on both sides of (10.20), solved by iteration.

Examples

1. **Midpoint method** (explicit two-step)

$$x_{n+1} = x_{n-1} + 2hf(t_n, x_n) \qquad n \geq 1$$

2. **The trapezoidal method** (implicit one-step)

$$x_{n+1} = x_n + \frac{h}{2}\Big(f(t_n, x_n) + f(t_{n+1}, x_{n+1})\Big)$$

For $x(t)$ differentiable define the Truncation error for integrating $x'(t)$:

$$T_n(x) = x(t_{n+1}) - \Big(\sum_{j=0}^{p} a_j x(t_{n-j}) + h \sum_{j=-1}^{p} b_j x'(t_{n-j}) \Big), \ n \geq p \tag{10.21}$$

and the function $\tau_n(x)$ by

$$\tau_n(x) = \frac{1}{h} T_n(x).$$

We want to choose a_j's and b_j's to have consistency:

Consistency error

To prove $x_n \to x(t_n)$, x_n solution of (10.20), it is necessary that

$$\tau(h) \equiv \max_{t_0 \leq t_n \leq T} |\tau_n(x)| \to 0 \quad \text{as } h \to 0. \tag{10.22}$$

Order of convergence of x_n to $x(t)$ is related to the speed of convergence in (10.22) :

$$\tau(h) = O(h^m) \tag{10.23}$$

Consistency condition

Theorem

Assume $m \in \mathbb{N}^*$ *and*

1. $x \in C^1[t_0, T]$. *The method (10.20) is* **consistent**, *i.e. (10.22) holds, iff*

$$\sum_{j=0}^{p} a_j = 1 \qquad -\sum_{j=0}^{p} j a_j + \sum_{j=-1}^{p} b_j = 1. \tag{10.24}$$

2. $x \in C^{m+1}[t_0, T]$. *The method (10.20) is of order* *(of convergence)* m *iff (10.24) holds and*

$$\sum_{j=0}^{p} (-j)^i a_j + \sum_{j=-1}^{p} (-j)^{i-1} b_j = 1 \quad i = 2, \ldots, m. \tag{10.25}$$

Adams Methods: multistep methods based on interpolation

$$x(t_{n+1}) - x(t_n) = \int_{t_n}^{t_{n+1}} x'(t)dt = \int_{t_n}^{t_{n+1}} f(t, x(t))dt$$

<u>Idea</u>: replace $f(t, x(t))$ by a polynomial $v_p(t)$ that interpolates f at a certain number of points t_i:

$$v_p(t_i) = f(t_i, x(t_i)).$$

<u>Polynomial Interpolation</u>

Let $t_0, \dots, t_p$ be $p+1$ points and let $z_0, \dots, z_p \in \mathbb{R}$.
$\exists!$ polynomial of $\deg \le p$ that interpolates the z_i's at t_i's given by:

$$v_p(t) = \sum_{j=0}^{p} z_j L_j(t), \qquad \underbrace{L_j(t) = \prod_{i=0, i\neq j}^{p} \frac{t - t_i}{t_j - t_i}}_{\text{Lagrange basis function}} .$$

Adams Methods: properties of Lagrange basis function

- L_j is a polynomial of degree p
- $L_j(t_i) = \begin{cases} 0, & i \neq j \\ 1, & i = j \end{cases}$
- If data z_i's come from a function z, then the interpolation error:

$$z(t) - v_p(t) = \frac{(t-t_0)(t-t_1)\cdots(t-t_p)}{(p+1)!} z^{(p+1)}(\xi), \qquad (10.26)$$

$$\xi \in [\min_i(t_i,t), \max_i(t_i,t)].$$

- Another form of v_p is given by Backward Differences (assume $t_{i+1} - t_i = h, \forall i$):

$$v_p(t) = z_p + \frac{t-t_p}{h}\nabla z_p + \ldots + \frac{(t-t_0)(t-t_1)\cdots(t-t_p)}{p!h^p}\nabla^p z_p,$$

where

$$\nabla z_i = z_i - z_{i-1}, \nabla^2 z_i = \nabla z_i - \nabla z_{i-1}, \cdots, \nabla^k z_i = \nabla^{k-1} z_i - \nabla^{k-1} z_{i-1}.$$

Adams-Bashforth methods: Interpolation nodes: $t_n, t_{n-1}, \ldots, t_{n-p}$

Example

Linear interpolation of f at t_n and t_{n-1}
$v_1(t) = \frac{(t_n-t)x'(t_{n-1})+(t-t_{n-1})x'(t_n)}{h} = \alpha + t\beta.$
Indeed: $v_1(t_n) = x'(t_n) = f(t_n, x(t_n)), v_1(t_{n-1}) = x'(t_{n-1})$, and

$$x(t_{n+1})-x(t_n) \simeq \int_{t_n}^{t_{n+1}} v_1(t)dt = \frac{x'(t_{n-1})}{h}\int_{t_n}^{t_{n+1}}(t_n-t)dt + \frac{x'(t_n)}{h}\int_{t_n}^{t_{n+1}}(t-t_{n-1})dt$$

$$= \frac{x'(t_{n-1})}{h}\frac{(t_n-t_{n+1})^2}{2}(-1) + \frac{x'(t_n)}{h}\left[\frac{(t_{n+1}-t_{n-1})^2}{2} - \frac{(t_n-t_{n-1})^2}{2}\right]$$

$$= -\frac{h}{2}x'(t_{n-1}) + \frac{3h}{2}x'(t_n) = -\frac{h}{2}f(t_{n-1}, x(t_{n-1})) + \frac{3h}{2}f(t_n, x(t_n)).$$

$\Longrightarrow x_{n+1} - x_n = -\frac{h}{2}f(t_{n-1}, x_{n-1}) + \frac{3h}{2}f(t_n, x_n)$, 2 step method

Rewritten as $x_{n+1} = \sum_0^p a_j x_{n-j} + h\sum_{-1}^p b_j f(t_{n-j}, x_{n-j})$, with
$a_0 = 1, a_1 = 0$ and $b_{-1} = 0, b_0 = 3/2, b_1 = -1/2$.

Adams-Bashforth

Define $\tilde{f}_i = f(t_i, x(t_i))$. The backward difference formula gives

$$v_p(t) = \tilde{f}_n + \frac{\overbrace{t - t_n}^{sh}}{h} \nabla \tilde{f}_n + \frac{(t-t_n)(\overbrace{t - t_{n-1}}^{(s+1)h})}{2h^2} \nabla^2 \tilde{f}_n + \dots + \frac{(t-t_n)\dots(t-t_{n-p+1})}{p!h^p} \nabla^p \tilde{f}_n.$$

Integrate using the change of variable $t = x_n + sh, dt = hds$:

$$\int_{t_n}^{t_{n+1}} v_p(t)dt = h \int_0^1 v_p(t_n + sh)ds \qquad \Rightarrow$$

and recalling that

$$v_p(t_n+sh) = \tilde{f}_n + s\nabla \tilde{f}_n + \frac{s(s+1)}{2} \nabla^2 \tilde{f}_n + \dots + \frac{s(s+1)\dots(s+p-1)}{p!} \nabla^p \tilde{f}_n$$

to get the formula

$$x_{n+1} = x_n + h \sum_{j=1}^{p} \frac{\nabla^j f_n}{j!} \int_0^1 s(s+1)\dots(s+p-1)ds + h \underbrace{f_n}_{f(t_n,x_n)} .$$

$a_0 = 1, a_j = 0, j = 1, \dots, p$; to get b_j's we need to expand the backward differences.

Adams-Bashforth

Example. Take $p = 2 \Rightarrow 3$ step method

$$x_{n+1} = x_n + h\Big(\nabla f_n \underbrace{\int_0^1 s ds}_{1/2} + \frac{1}{2}\nabla^2 f_n \underbrace{\int_0^1 s(s+1)ds}_{5/6}\Big) + hf_n$$

$$x_{n+1} = x_n + \frac{h}{2}(f_n - f_{n-1}) + \frac{5h}{12}\Big(\underbrace{\nabla f_n - \nabla f_{n-1}}_{f_n - f_{n-1} - f_{n-1} + f_{n-2}}\Big) + hf_n$$

$$= x_n + \tfrac{23}{12}hf_n - \tfrac{16}{12}hf_{n-1} + \tfrac{5}{12}hf_{n-2}$$

$$\Rightarrow b_0 = \tfrac{23}{12}, b_1 = -\tfrac{16}{12}, b_2 = \tfrac{5}{12}$$

p	b_0	b_1	b_2	b_3	LTE$\leq C_{p+1}h^{p+1}$
0	1				1/2
1	3/2	-1/2			5/12
2	23/12	-16/12	5/12		3/8 3rd order
3	55/24	-59/24	37/24	-9/24	251/720 4th order

pag. 387

Adams-Bashforth

The LTE is

$$\tau_n = \frac{x(t_{n+1}) - x(t_n)}{h} - \frac{1}{h}\int_{t_n}^{t_{n+1}} v_p(t)dt,$$

but we already know the interpolation error (10.26):

$$x'(t) - v_p(t) = \frac{(t-t_n)(t-t_{n+1})\dots(t-t_{n-p})}{(p+1)!}x^{(p+2)}(\xi_n).$$

Since

$$\tau_n = \frac{1}{h}\int_{t_n}^{t_{n+1}}(x'-v_p)dt = \frac{1}{h(p+1)!}\int_{t_n}^{t_{n+1}}\underbrace{(t-t_n)\dots(t-t_{n-p})}_{\geq 0 \text{ on } [t_n,t_{n+1}]}dt\, x^{(p+2)}(\xi_n)$$

$$\overset{t=t_n+sh}{=\!=\!=} h^{p+1}\frac{x^{(p+2)}(\xi_n)}{(p+1)!}\int_0^1 s(s+1)\dots(s+p)\,ds \Rightarrow$$

The Adams-Bashforth methods are consistent of order $p+1$.

(explicit: $b_{-1} = 0$)

Adams-Moulton methods

Implicit: the interpolation nodes $t_{n+1}, t_n, \dots, t_{n-p+1}$

$$v_p(t) = f_{n+1} + \frac{t - t_{n+1}}{h} \nabla f_{n+1} + \dots + \frac{(t - t_{n+1}) \dots (t - t_{n-p+2})}{p! h^p} \nabla^p f_{n+1}$$

Integrate $\int_{t_n}^{t_{n+1}} v_p$, use the change of var. $(t = t_n + sh) \Rightarrow$ the method:

$$x_{n+1} = x_n + h f_{n+1} + h \sum_{j=1}^{p} \frac{\nabla^j f_{n+1}}{j!} \int_0^1 (s-1)s(s+1)\dots(s+j-2)ds$$

p	b_{-1}	b_0	b_1	b_2
0	1			
1	1/2	1/2		
2	5/12	8/12	-1/12	2-step 3rd order
3	9/24	19/24	-5/24	1/24 3-step 4th order

LTE for AM: $\tau_n = h^{p+1} \frac{x^{(p+2)}(\xi_n)}{(p+1)!} \int_0^1 (s-1)s(s+1)\dots(s+p-1)ds$

Adams methods

The difference between the AB and AM: **the region of absolute stability are larger for AM than for AB**.

Absolute stability

Definition

For a given numerical method, the **region of absolute stability** is the region of the complex plane $\{z \in \mathbb{C}\}$ s.t. applying the method to $x' = \lambda x$ with $z = \lambda h$ $(h \in \mathbb{R}, \lambda \in \mathbb{C})$ yields a solution x_n s.t. $|x_{n+1}| \leq |x_n|$.

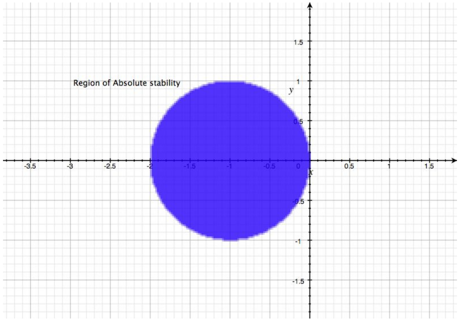
Exact solution: $x(t) = x(0)e^{\lambda t}, \quad \lambda \leq 0, \lambda \in \mathbb{R} \Rightarrow |x(t_{n+1})| \leq |x(t_n)|$
Apply the definition to Forward Euler:

$$x_{n+1} = x_n + h(\lambda x_n) = x_n(1 + \lambda h)$$

and we want $|x_{n+1}| \leq |x_n|$.
Hence we need $|1 + \underbrace{\lambda h}_{z}| \leq 1$, $|1 + z| \leq 1$.

Absolute stability

Assume $\lambda \in \mathbb{R}$ and $\lambda < 0$. Then
$|1 + \lambda h| \leq 1, -1 \leq 1 + \lambda h \leq 1 \Rightarrow -1 \leq 1 + \lambda h \Leftrightarrow -\lambda h \leq 2 \Rightarrow h \leq \frac{2}{-\lambda}$
for stiff problems h has to be very small.



Backward Euler method

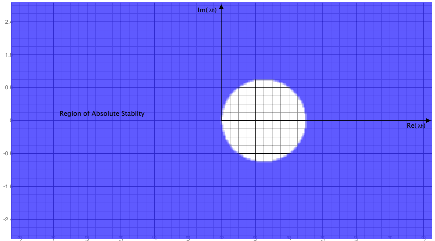
- Forward Euler
$$x_{n+1} = x_n + hf(t_n, x_n)$$
is explicit, $O(h)$, **not** A-stable.
- Backward Euler
$$\boldsymbol{x_{n+1} = x_n + hf(t_{n+1}, x_{n+1})} \qquad (10.27)$$
is implicit, $O(h)$, LTE: $\tau_n = -\frac{h}{2}x''(\xi_n)$, A-stable.

Backward Euler method is A-stable

Region of Absolute Stability: Apply (10.27) to $x' = \lambda x$:

$$x_{n+1} = x_n + \lambda h x_{n+1} \Leftrightarrow (1-\lambda h)x_{n+1} = x_n$$

$$|x_{n+1}| \le \frac{1}{|1-\lambda h|}|x_n| \Rightarrow \frac{1}{|1-\lambda h|} \le 1$$



Theorem (recall - convergence of single step RK methods)

Assume ψ is Lipschitz in x_n:

$$|\psi(h,t,\xi,f) - \psi(h,t,\eta,f)| \le K|\xi - \eta| \qquad \forall \xi, \eta.$$

Then

$$\max_{0\le n\le N} |x(t_n) - x_n| \le e^{K(t_n - t_0)}|x(t_0) - x_0| + \frac{e^{K(t_n-t_0)} - 1}{K}\|\tau\|_\infty.$$

Regions of Absolute Stability

Rewrite RK in vector form:

$$\mathbf{K} = \begin{pmatrix} K_1 \\ \vdots \\ K_s \end{pmatrix}, \mathbf{A} = \begin{pmatrix} a_{11} & \dots & a_{1s} \\ \vdots & & \vdots \\ a_{s1} & \dots & a_{ss} \end{pmatrix}, \mathbf{b} = \begin{pmatrix} b_1 \\ \vdots \\ b_s \end{pmatrix}, \mathbf{1} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}.$$

Apply the method to $x' = \lambda x, \forall 1 \leq i \leq s$

$$K_i = x_n + h \sum_{j=1}^{s} a_{ij} \lambda K_j \Rightarrow \mathbf{K} = x_n \mathbf{1} + \lambda h \mathbf{A}\mathbf{K}$$

$$x_{n+1} = h\lambda \sum_{j=1}^{s} b_j K_j \Rightarrow x_{n+1} = x_n + h\lambda \mathbf{b}^T \mathbf{K}$$

Regions of Absolute Stability

Solve for $\mathbf{K}$:

$$\mathbf{K} - \lambda h \mathbf{A}\mathbf{K} = x_n \mathbf{1}, \quad \mathbf{K} = (\mathbf{I} - \lambda h \mathbf{A})^{-1} x_n \mathbf{1}.$$

$$x_{n+1} = x_n + h\lambda \mathbf{b}^T (I - h\lambda \mathbf{A})^{-1} x_n \mathbf{1} = x_n \Big(1 + h\lambda \mathbf{b}^T (I - h\lambda \mathbf{A})^{-1} \mathbf{1}\Big)$$

Absolute stability (formal derivation): $|x_{n+1}| \le |x_n|$ hence we want $1 + h\lambda \mathbf{b}^T (I - h\lambda \mathbf{A})^{-1} \mathbf{1} \le 1$. Let $z = \lambda h$:

$$(I - z\mathbf{A})^{-1} = I + z\mathbf{A} + z^2\mathbf{A}^2 + z^3\mathbf{A}^3 + \ldots = \sum_{k=0}^{\infty} z^k \mathbf{A}^k,$$

$$\mathbf{b}^T (I - z\mathbf{A})^{-1} \mathbf{1} = \mathbf{b}^T \Big(\sum_{k=0}^{\infty} z^k \mathbf{A}^k\Big) \mathbf{1} = \sum_{k=0}^{\infty} z^k \mathbf{b}^T \mathbf{A}^k \mathbf{1},$$

$$\Rightarrow 1 + z\mathbf{b}^T (I - z\mathbf{A})^{-1} \mathbf{1} = 1 + \sum_{k=0}^{\infty} z^{k+1} \mathbf{b}^T \mathbf{A}^k \mathbf{1}$$

$$x_{n+1} = x_n \Big(1 + z\mathbf{b}^T\mathbf{1} + z^2\mathbf{b}^T\mathbf{A}\mathbf{1} + z^3\mathbf{b}^T\mathbf{A}^2\mathbf{1} + \ldots + z^p\mathbf{b}^T\mathbf{A}^{p-1}\mathbf{1} + \ldots\Big) \quad (\star)$$

Regions of Absolute Stability

The true solution of $x' = \lambda x$ is $x(t) = x(0)e^{\lambda x}$

$$\left.\begin{array}{l} x(t_{n+1}) = x(0)e^{\lambda t_{n+1}} \\ x(t_n) = x(0)e^{\lambda t_n} \end{array}\right\} \Rightarrow x(t_{n+1}) = x(t_n)e^{\lambda h} = x(t_n)e^{z}$$

$$x(t_{n+1}) = x(t_n)\left(1 + z + \frac{z^2}{2} + \ldots + \frac{z^p}{p!} + \ldots\right) \quad (\star\star)$$

The method is of order p if $\left\{\begin{array}{l} \mathbf{b}^T\mathbf{1} = 1 \\ \vdots \\ \mathbf{b}^T\mathbf{A}^{p-1}\mathbf{1} = \frac{1}{p!} \end{array}\right.$. Then

$$x_{n+1} = x_n\left(1 + z + \frac{z^2}{2} + \ldots + \frac{z^p}{p!} + \sum_{k=p+1}^{\infty} z^p\mathbf{b}^T\mathbf{A}^{k-1}\mathbf{1}\right). \quad (\bullet)$$

For RK explicit: $\mathbf{A}_{s\times s} = \begin{pmatrix} 0 & \ldots & 0 \\ & \ddots & \vdots \\ & & 0 \end{pmatrix}$, and $\mathbf{A}$ is nilpotent,

$$\mathbf{A}^k = 0, \forall k \geq s \Rightarrow \sum_{k=p+1}^{\infty} z^p\mathbf{b}^T\mathbf{A}^{k-1}\mathbf{1} = \sum_{k=p+1}^{s} z^p\mathbf{b}^T\mathbf{A}^{k-1}\mathbf{1}.$$

Regions of Absolute Stability

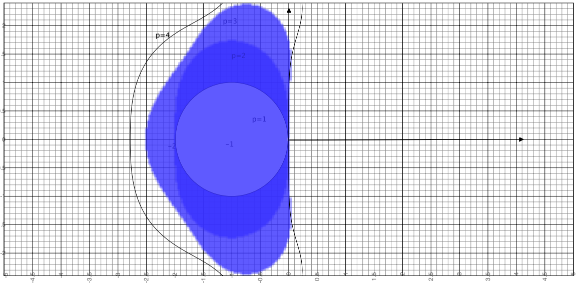
If we have an explicit RK-s stages with order p:

$$x_{n+1} = x_n\left(1 + z + \frac{z^2}{2} + \ldots + \frac{z^p}{p!}\right).$$

Region of absolute stability:

$$\left|1 + z + \ldots + \tfrac{z^p}{p!}\right| \leq 1.$$

For a fixed p, all explicit regions RK-s stage of order p have the **same** region of absolute stability. **Not** appropriate for stiff problems.



There are no A-stable explicit Runge-Kutta methods.

Predictor-Corrector Scheme: remove nonlinearity for implicit scheme

1. Prediction step: use explicit
2. Correction step: use modified implicit: x_{n+1} in RHS is replaced by the predicted value

Example: Adams Methods

1. predictor step: AB of order $p+1$

$$x_{n+1}^{P} = x_n + h\sum_{j=0}^{p} b_j^{AB} f(t_{n-j}, x_{n-j}) \qquad \text{explicit}$$

2. corrector step: AM of order $p+1$

$$x_{n+1}^{C} = x_n + h\sum_{j=0}^{p-1} b_j^{AM} f(t_{n-j}, x_{n-j}) + hb_{-1}^{AM} f(t_{n+1}, x_{n+1}^{P}) \quad \text{explicit}$$

3. NEW VALUE: $x_{n+1} = x_{n+1}^{C}$.

Predictor-Corrector Scheme: Local truncation error

$$x(t_{n+1}) \overset{AB}{=} x(t_n) + h\sum_{j=0}^{p} b_j^{AB} f(t_{n-j}, x(t_{n-j})) + h\tau_n^P$$

$$x(t_{n+1}) \overset{AM}{=} x(t_n) + h\sum_{j=0}^{p-1} b_j^{AM} f(t_{n-j}, x(t_{n-j})) + hb_{-1}^{AM} f(t_{n+1}, x(t_{n+1})) + h\tau_n^C$$

$$x(t_{n+1}) \overset{PC}{=} x(t_n) + h\sum_{j=0}^{p-1} b_j^{AM} f(t_{n-j}, x(t_{n-j})) + hb_{-1}^{AM} f(t_{n+1}, x(t_{n+1}) - h\tau_n^P) + h\tau_n$$

$$0 = hb_{-1}^{AM}\Big(\underbrace{f(t_{n+1}, x(t_{n+1})) - f(t_{n+1}, x(t_{n+1}) - h\tau_n^P)}_{\frac{\partial f}{\partial x}(t_{n+1},\xi_{n+1})h\tau_n^P}\Big) + h\tau_n^C - h\tau_n$$

$$\tau_n = \tau_n^C + b_{-1}^{AM}\frac{\partial f}{\partial x}(t_{n+1}, \xi_{n+1})h\tau_n^P \qquad (10.28)$$

Predictor-Corrector Scheme

Remark: One could reduce the order of the predictor step:

$$x_{n+1}^P = x_n + h\sum_{j=0}^{p-1} b_j^{AB} f(t_{n-j}, x_{n-j})$$

So, for both steps to evaluate f at the same nodes.

Example Modified Euler

1. $x_{n+1}^P = x_n + hf(t_n, x_n) \longrightarrow$ $O(h)$
2. $x_{n+1}^C = x_n + \frac{h}{2}f(t_n, x_n) + \frac{h}{2}f(t_{n+1}, x_{n+1}^P) \longrightarrow$ $O(h^2)$
3. $\tau_n^P = hx''(\xi_n), \tau_n^C = -\frac{h^2}{12}x'''(\zeta_n)$ $\overset{(10.28)}{\Longrightarrow}$ $O(h^2)$

Predictor-Corrector Scheme: Convergence

Theorem

$$|x(t_n) - x_n| \le \max_{0\le n\le p} |x(t_n) - x_n| e^L + \frac{\|\tau\|_\infty (e^L - 1)}{\|\frac{\partial f}{\partial x}\|_\infty (A + h\|\frac{\partial f}{\partial x}\|_\infty B|b_{-1}^{AM}|)}$$

with $$L = (t_n - t_0)A + hB\left\|\frac{\partial f}{\partial x}\right\|_\infty |b_{-1}^{AM}|),$$

$$A = \sum_{j=-1}^{p-1} |b_j^{AM}|, \quad B = \sum_{j=0}^{p} |b_j^{AB}|.$$

$$x_{n+1} \overset{PC}{=} x_n + h\sum_0^{p-1} b_j^{AM} f(t_{n-j}, x_{n-j}) + hb_{-1}^{AM} f(t_{n+1}, x_{n+1})$$

$$x(t_{n+1}) \overset{LTE}{=} x(t_n) + h\sum_0^{p-1} b_j^{AM} f(t_{n-j}, x(t_{n-j})) + hb_{-1}^{AM} f(t_{n+1}, x(t_{n+1}) - h\tau_n^P) + h\tau_n$$

Applying the multistep method

$$x_{n+1} = \sum_{j=0}^{p} a_j x_{n-j} + h \sum_{j=-1}^{p} b_j f(t_{n-j}, x_{n-j})$$

to the model equation

$$x' = \lambda x, \qquad x(0) = 1$$

we obtain the homogeneous linear difference equation

$$(1 - h\lambda b_{-1}) x_{n+1} - \sum_{j=0}^{p} (a_j + h\lambda b_j) x_{n-j} = 0, \qquad n \geq p$$

Looking for a solution of the form $\boldsymbol{x_n = r^n}$,

we find the *characteristic equation*

$$(1 - h\lambda b_{-1}) r^{p+1} - \sum_{j=0}^{p} (a_j + h\lambda b_j) r^{p-j} = 0$$

or equivalently $\rho(r) - h\lambda\sigma(r) = 0$

$$\underbrace{r^{p+1} - \sum_{j=0}^{p} a_j r^{p-j}}_{\rho(r)} - h\lambda \underbrace{(b_{-1} r^{p+1} + \sum_{j=0}^{p} b_j r^{p-j})}_{\sigma(r)} = 0$$

Remark

The consistency condition (10.24) implies $\rho(1) = 0$.

Let $r_0 = 1, r_1, \ldots, r_p$ be the roots of ρ.

Definition

The multistep method satisfies the *root condition* if

(i) $|r_j| \leq 1, j = 0, \ldots, p$

(ii) if $|r_j| = 1$, then $\rho'(r_j) \neq 0$ (i.e. these roots are of multiplicity one).

Recall **stability**: if the perturbed initial values $z_0, \ldots, z_p$ satisfy

$$|x_i - z_i| \leq \varepsilon, \quad i = 0, 1, \ldots, p$$

then the numerical solutions $\{x_i\}$ and $\{z_i\}$ also satisfy

$$\max_{t_0 \leq t_n \leq T} |x_i - z_i| \leq c_f \varepsilon, \quad \forall h \leq h_0.$$

Theorem

If the multistep method is consistent, then:

$$\textit{stability} \Leftrightarrow \textit{the root condition.}$$

Recall convergence: if the error in the initial values $x_0, \ldots, x_p$ $\eta(h) = \max\limits_{i=0,\ldots,p} |x(t_0 + ih) - x_i| \to 0$ implies $\max\limits_{0 \le n \le N} |x(t_n) - x_n| \to 0$.

Theorem

Assume the multistep method is consistent.

convergence $\Leftrightarrow$ root condition.

Example

$\underbrace{x_{n+1} = 10x_n - 9x_{n-1}}_{\rho} + h(-3f(t_n, x_n) - 5f(t_{n-1}x_{n-1}))$, 2 step method.

Is it stable, is it convergent?

$$\rho(r) = r^2 - 10r + 9 = (r-1)(r-9),$$

$$\underbrace{r_0 = 1}_{\Rightarrow \text{consistency}}, r_1 = 9 > 1 \Rightarrow \text{Not stable, Not convergent}$$

. . . .

A concrete example to show that is not stable:

$$x' = 0, x(0) = 0 \Rightarrow x(t) = 0$$

$$x_{n+1} = 10x_n - 9x_{n-1}, \ \textit{if } x_0 = x_1 = 0 \quad \Rightarrow x_n = 0 \ \forall n$$

choose $z_0 = \frac{\varepsilon}{9}, z_1 = \varepsilon,$ *so* $\max\{|x_0 - z_0|, |x_1 - z_1|\} = \varepsilon.$

Then one can show by induction $z_n = \varepsilon 9^{n-1} \Rightarrow |x_n - z_n| \xrightarrow[n\to\infty]{} +\infty.$

Definition

The multistep method is **strongly stable** (relatively stable) if $|r_j(\lambda h)| \le |r_0(\lambda h)| \quad \forall j = 1, \dots, p, \forall \lambda h$ sufficiently small.

(The *parasitic* to be well behaved compared to the *dominant*).

If the method is stable, but not strongly stable, it is called **weakly stable**.

Theorem

The method is strongly stable if the roots r_j's of ρ satisfy the

strong root condition: $\quad |r_j| < 1, \quad j = 1, \dots, p.$

Example

Midpoint method:

$$x_{n+1} = x_{n-1} + 2hf(t_n, x_n),$$

$$\rho(r) = r^2 - 1 = (r-1)(r+1), r_0 = 1, r_1 = -1,$$

$\star$ stable: $|r_i| \le 1$, **not** strongly stable

Recall $r_0(\lambda h) = \lambda h + \sqrt{1 + \lambda^2 h^2}, r_1(\lambda h) = \lambda h - \sqrt{1 + \lambda^2 h^2} \Rightarrow |r_1(\lambda h)| > |r_0(\lambda h)|$ when $\lambda < 0$.

Example

Adams-Bashforth 2nd order
(considered the most stable of the two-step methods):

$$x_{n+1} = x_n + \frac{h}{2}\Big(3f(t_n, x_n) - f(t_{n-1}, x_{n-1})\Big)$$

$$\rho(r) = r^2 - r = r(r-1), r_0 = 1, r_1 = 0$$

stable, **strongly stable**.

Adams-Bashforth in general:

$$x_{n+1} = x_n + h\sum_{j=0}^{p} b_j^{AB} f(t_{n-j}, x_{n-j}) \quad (p \textit{ steps})$$

$$\rho(r) = r^{p+1} - r^p = r^p(r-1), r_0 = 1, r_1 = 0$$

strongly stable.

Definition

The **region of strong stability** is the set of points ($z = \lambda h$) in the complex plain $\mathbb{C}$ such that the strong stability condition is satisfied.

Example Adams-Bashforth 2nd order

$$x_{n+1} = x_n + \tfrac{h}{2}\Big(3f(t_n, x_n) - f(t_{n-1}, x_{n-1})\Big)$$

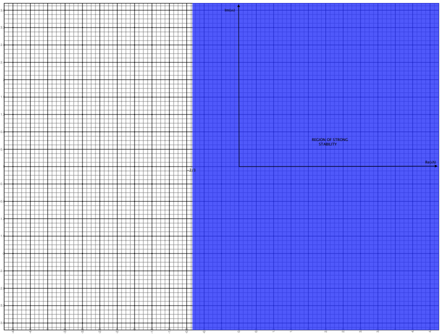
r_j's are the roots of $\rho(r) - \lambda\sigma(r) = 0$:

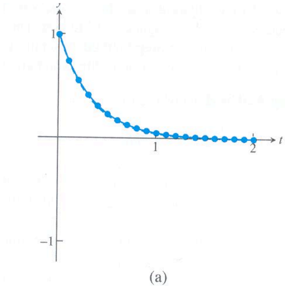
$$r^2 - r - \lambda h\Big(\frac{3}{2}r - \frac{1}{2}\Big) = r^2 - (1 + 3/2\lambda h)r + \frac{\lambda h}{2}$$

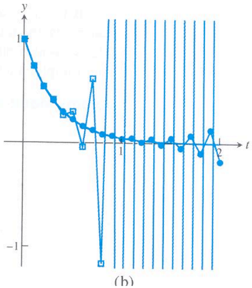
$$r_0(\lambda h) = \frac{1}{2}(1 + \frac{3}{2}\lambda h) + \frac{1}{2}\sqrt{(1 + \frac{3\lambda h}{2})^2 - 2\lambda h} \xrightarrow[\lambda \to 0]{} +1$$

$$r_1(\lambda h) = \frac{1}{2}(1 + \frac{3}{2}\lambda h) - \frac{1}{2}\sqrt{(1 + \frac{3\lambda h}{2})^2 - 2\lambda h}$$

We want $|r_1(\lambda h)| \leq |r_0(\lambda h)|$. If $\lambda \in \mathbb{R}$ then $\lambda h \geq -\frac{2}{3}$.



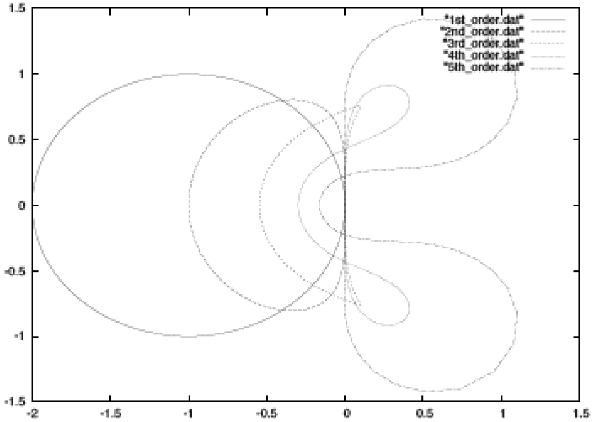




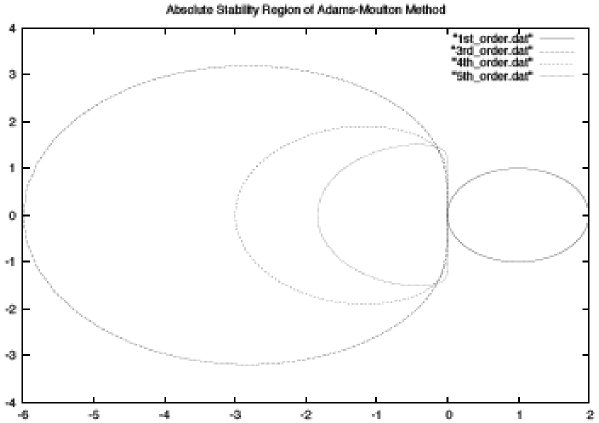
The Adams-Bashforth, midpoint (*weakly stable*) and unstable method $(x_{n+1} = -x_n + 2x_{n-1} + \frac{h}{2}(5f_n + f_{n-1}))$ for $\begin{cases} x' = -3x, \\ x(0) = 1 \end{cases}$ in $[0, 2]$.

Note that the size of λ affects convergence even when it is not involved in the truncation error. (We need $\lambda h \in$ absolute stability regions to have sensible behaviour in the convergence of the numerical method.

Hence very small h, even if the truncation error $T_n = h^2$ does not depend on λ. That does not happen for Backward Euler method.



The stability regions of Adams-Bashforth methods.



The stability regions of Adams-Moulton regions.

Recall regions of absolute stability for Euler, trapezoidal (A) and Backward Euler method (A).

Theorem (Dahlquist)

There are **no** A-stable multistep methods of order greater than 2.

Stiff Differential Equations

Qualitative Definition

An IVP is stiff in some interval if the step size needed to maintain stability of an explicit method is much smaller than the step size requiered to represent the solution accurately.

Quantitative Definition

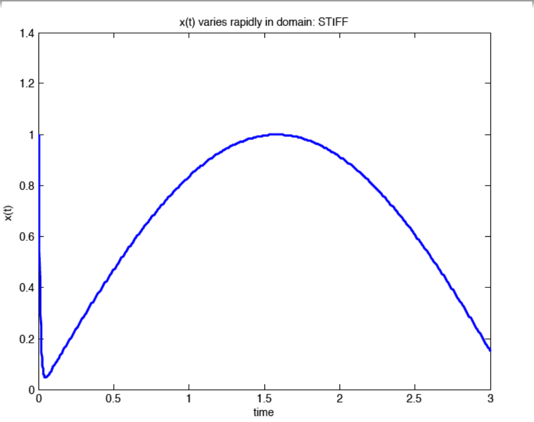
We say that $x' = \lambda x$ is stiff if $Re(\lambda) < 0$ and $|Re(\lambda)|$ is very large.
We say that $\boldsymbol{x}' = \boldsymbol{f}(t, \boldsymbol{x})$ is stiff if some eigenvalues of the Jacobian matrix $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}$ have a negative real part, large in magnitude.

Any A-stable method is appropriate for stiff problems.

Stiff Differential Equations

Example

$$\begin{cases} x' = -100x + 100\sin(t) \\ x(0) = 1 \end{cases} \Rightarrow x(t) = c_1 e^{-100t} + c_2 \sin(t) + c_3 \cos(t).$$



Consider the equation

$$\begin{cases} x' = \lambda x + (1-\lambda)\cos t - (1-\lambda)\sin t \\ x(0) = 1 \end{cases} \qquad x(t) = \sin t + \cos t.$$

and the perturbed equation

$$\begin{cases} z' = \lambda z + (1-\lambda)\cos t - (1-\lambda)\sin t \\ z(0) = 1 + \varepsilon \end{cases} \qquad z(t) = \sin t + \cos t + \varepsilon e^{\lambda t}.$$

If $\lambda < 0$ large, then $z(t) \simeq x(t)$ very fast in t (i.e., take $\lambda = -10^4$).
But the numerical method requires, let say for Euler method:

$$\lambda h \in (0,2) \iff 0 < h < 0.0002$$

Backward Euler method for stiff DEs

When solving $x' = f(t,x)$ using the Backward Euler method

$$x_{n+1} = x_n + hf(t_{n+1}, x_{n+1}) \tag{10.29}$$

we find x_{n+1} by fixed point iteration

$$x_{n+1}^{(k+1)} = x_n + hf(t_{n+1}, x_{n+1}^{(k)}), \qquad k = 0, 1, \dots \tag{10.30}$$

or say Newton's method.
Despite (10.29) being A-stable, when using (10.30) on stiff problems we get a restriction on λh for the convergence of the fixed point iteration:

$$x_{n+1} - x_{n+1}^{(k+1)} \simeq h \underbrace{\frac{\partial f}{\partial x}(t_{n+1}, x_{n+1})}_{\text{negative, large in magnitude}} (x_{n+1} - x_{n+1}^{(k)})$$

Hence Newton's method is needed - very costly for large systems.

The Lorenz equations

Simplification of a miniature atmosphere model designed to study Rayleigh-Bénard convection, the movement of heat in a fluid, such as air, from a lower warm medium (such as ground) to a higher cool medium (like the upper atmosphere).
In this model of a two-dimensional atmosphere, a circulation of air develops that can be described by three equations:

$$\begin{aligned} x' &= -sx + sy \\ y' &= -xz + rx - y \\ z' &= xy - bz \end{aligned}$$

Here x= clockwise circulation velocity, y=temperature difference between the ascending and descending columns of air, z= the deviation of a strictly linear temperature profile in vertical direction. s is the Prandl number, r is the Reynolds number, a and b are parameters of the system.

The most common setting is $s = 10, r = 28, b = 8/3$.

The Lorenz equations

Plot the attractor

Coupled and uncoupled systems

Coupled system

$$\begin{cases} x'(t) = x(t) - y(t) + 2t - t^2 - t^3 \\ y'(t) = x(t) + y(t) - 4t^2 + t^3 \end{cases}$$

Uncoupled system

$$\begin{cases} x'(t) = x(t) + 2t - t^2 - t^3 \\ y'(t) = y(t) - 4t^2 + t^3 \end{cases}$$

Taylor Series Method

$$\begin{cases} x' = x - y + 2t - t^2 - t^3 \\ y' = x + y - 4t^2 + t^3 \end{cases}$$

$$\begin{cases} x'' = x' - y' + 2 - 2t - 3t^2 \\ y'' = x' + y' - 8t + 3t^2 \end{cases}$$

$$\begin{cases} x''' = x'' - y'' - 2 - 6t \\ y''' = x'' + y'' - 8 + 6t \end{cases}$$

$$\begin{cases} x^{(4)} = x''' - y''' - 6 \\ y^{(4)} = x''' + y''' + 6 \end{cases}$$

$$\vdots$$

To compute $x(t+h)$ and $y(t+h)$ from $x(t), y(t)$ we procced as in the scalar case, using a few terms in the Taylor series:

$$x(t+h) = x(t) + hx'(t) + \frac{h^2}{2}x'' + \frac{h^3}{3!}x''' + \frac{h^4}{4!}x^{(4)} + \cdots$$

$$y(t+h) = y(t) + hy'(t) + \frac{h^2}{2}y'' + \frac{h^3}{3!}y''' + \frac{h^4}{4!}y^{(4)} + \cdots$$

Vector Notation

The initial value problem

$$\begin{cases} \boldsymbol{X}' = \boldsymbol{F}(t, \boldsymbol{X}) \\ \boldsymbol{X}(t_0) = \boldsymbol{S}, \text{given} \end{cases}$$

is a vector form of the previous system

$$\begin{cases} \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x - y + 2t - t^2 - t^3 \\ x + y - 4t^2 + t^3 \end{bmatrix} \\ \begin{bmatrix} x(0) \\ y(0) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{cases}$$

where $\boldsymbol{X} = \begin{bmatrix} x \\ y \end{bmatrix}, \boldsymbol{X}' = \begin{bmatrix} x' \\ y' \end{bmatrix}$.

Systems of ODEs

$$\boldsymbol{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \boldsymbol{X}' = \begin{bmatrix} x_1' \\ x_2' \\ \vdots \\ x_n' \end{bmatrix}, \quad \boldsymbol{F} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix}, \quad \boldsymbol{S} = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{bmatrix}.$$

where

$$\begin{cases} x_1' = f_1(t, x_1, x_2, \cdots, x_n) \\ x_2' = f_2(t, x_1, x_2, \cdots, x_n) \\ \vdots \\ x_n' = f_n(t, x_1, x_2, \cdots, x_n) \\ x_1(a) = s_1, x_2(a) = s_2, \cdots, x_n(a) = s_n. \end{cases}$$

Taylor series method: vector notation

The m-order method:

$$\boldsymbol{X}(t+h) = \boldsymbol{X} + h\boldsymbol{X}' + \frac{h^2}{2}\boldsymbol{X}'' + \cdots + \frac{h^m}{m!}\boldsymbol{X}^{(m)}$$

with $\boldsymbol{X} = \boldsymbol{X}(t), \boldsymbol{X}' = \boldsymbol{X}'(t), \boldsymbol{X}'' = \boldsymbol{X}''(t), \ldots$.

Runge-Kutta method

The classical fourth-order Runge-Kutta method:

$$\boldsymbol{X}(t+h) = \boldsymbol{X} + \frac{h}{6}(\boldsymbol{K}_1 + 2\boldsymbol{K}_2 + 2\boldsymbol{K}_3 + \boldsymbol{K}_4)$$

with

$$\begin{cases} \boldsymbol{K}_1 = \boldsymbol{F}(t, \boldsymbol{X}) \\ \boldsymbol{K}_2 = \boldsymbol{F}(t + \frac{1}{2}h, \boldsymbol{X} + \frac{1}{2}h\boldsymbol{K}_1) \\ \boldsymbol{K}_3 = \boldsymbol{F}(t + \frac{1}{2}h, \boldsymbol{X} + \frac{1}{2}h\boldsymbol{K}_2) \\ \boldsymbol{K}_4 = \boldsymbol{F}(t + h, \boldsymbol{X} + h\boldsymbol{K}_3) \end{cases}$$

Predictor-Corrector scheme

$$\begin{cases} \boldsymbol{X}' = \boldsymbol{F}(t, \boldsymbol{X}) \\ \boldsymbol{X}(t_0) = \boldsymbol{S}, \text{given} \end{cases}$$

Adams-Bashforth multistep method:

$$\begin{aligned} \boldsymbol{X}^P(t+h) = \boldsymbol{X}(t) + \frac{h}{24}\{&55\boldsymbol{F}[\boldsymbol{X}(t)] - 59\boldsymbol{F}[\boldsymbol{X}(t-h)] + 37\boldsymbol{F}[\boldsymbol{X}(t-2h)] \\ &-9\boldsymbol{F}[\boldsymbol{X}(t-3h)]\} \end{aligned}$$

As corrector - the **Adams-Moulton** formula:

$$\begin{aligned} \boldsymbol{X}^C(t+h) = \boldsymbol{X}(t) + \frac{h}{24}\{&9\boldsymbol{F}[\boldsymbol{X}^P(t+h)] + 19\boldsymbol{F}[\boldsymbol{X}(t)] - 5\boldsymbol{F}[\boldsymbol{X}(t-h)] \\ &+\boldsymbol{F}[\boldsymbol{X}(t-2h)]\} \end{aligned}$$

Example

The surface tension S in a liquid is known to be a linear function of the temperature T.
For a particular liquid, measurements of the surface tension at certain temperatures:

T	0	10	20	30	40	80	90	95
S	68.0	67.1	66.4	65.6	64.6	61.8	61.0	60.0

What can be the most probable values of the constants in the equation

$$S = aT + b$$

be determined?

Linear Least Squares

Experimental data:

x	x_0	x_1	$\cdots$	x_m
y	y_1	y_2	$\cdots$	y_m

Assuming that

$$y = ax + b$$

what are the coefficients a and b?

What line passes most nearly through the points plotted?

- $ax_k + b - y_k = 0$
- $|ax_k + b - y_k| \neq 0$

- ℓ_1 **approximation**; the total absolute error:

$$\sum_{k=0}^{m} |ax_k + b - y_k|$$

  (linear programming)

- ℓ_2 **approximation**

$$\varphi(a,b) = \sum_{k=0}^{m} |ax_k + b - y_k|^2$$

  (if the errors follow a *normal probability distribution*, the minimization of $\varphi(a,b)$ produces a best estimate for a and b)

- ℓ_p **approximation**

$$\left(\sum_{k=0}^{m} |ax_k + b - y_k|^p\right)^{\frac{1}{p}}, \qquad 1 \le p < \infty$$

Linear Least Squares: Necessary conditions for ℓ_2 minimization

$\dfrac{\partial\varphi}{\partial a} = \dfrac{\partial\varphi}{\partial b} = 0$, i.e. the **normal equations**

$$\begin{cases} \sum_{k=0}^{m} 2(ax_k + b - y_k)x_k = 0 \\ \sum_{k=0}^{m} 2(ax_k + b - y_k) = 0 \end{cases} \Leftrightarrow \begin{cases} \underbrace{\left(\sum_{k=0}^{m} x_k^2\right)}_{s} a + \underbrace{\left(\sum_{k=0}^{m} x_k\right)}_{p} b = \underbrace{\sum_{k=0}^{m} y_k x_k}_{r} \\ \left(\sum_{k=0}^{m} x_k\right) a + (m+1)b = \underbrace{\sum_{k=0}^{m} y_k}_{q} \end{cases}$$

or

$$\begin{bmatrix} s & p \\ p & m+1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ q \end{bmatrix}$$

which can be solved by Gaussian elimination or directly

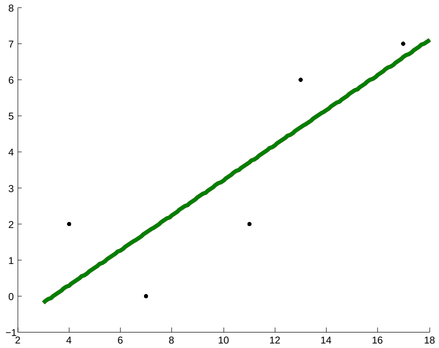
$$a = \frac{1}{d}[(m+1)r - pq]$$
$$b = \frac{1}{d}[sq - pr], \qquad \text{where } d = (m+1)s - p^2.$$

Linear Example

x	4	7	11	13	17
y	2	0	2	6	7

$$\Rightarrow \begin{cases} 644a + 52b = 227 \\ 52a + 5b = 17 \end{cases}$$

with solution $a = 0.4864$, $b = -1.6589$ and $\varphi(a, b) = 10.7810$.



Linear Least Squares. Key Idea

Goal: to determine the equation of a line

$$y = ax + b$$

that best fits the data, in the $\boldsymbol{\ell}_2$ sense.

With four data points (x_i, y_i), i.e. four equations

$$y_i = ax_i + b_i, \qquad i = 1:4$$

we can write

$$\boldsymbol{Ax} = \boldsymbol{b}$$

where

$$\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \\ x_4 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

Linear Least Squares. Key Idea

In general, we want to solve a linear system

$$\boldsymbol{Ax} = \boldsymbol{b}$$

where $\boldsymbol{A}$ is an $m \times n$ matrix and $m > n$.
The solution coincides with the solution of the **normal equations**

$$\boldsymbol{A^T Ax} = \boldsymbol{A^T b}$$

This corresponds to minimizing $\|\boldsymbol{Ax} - \boldsymbol{b}\|^2$.

Inconsistent system of equations

It is not hard to write down a system of equations that has no solutions:

$$\begin{aligned} x_1 + x_2 &= 2 \\ x_1 - x_2 &= 1 \\ x_1 + x_2 &= 3 \end{aligned}$$

What is the meaning of a system with no solutions (*inconsistent*)??

- Perhaps the coefficients are slightly inaccurate.
- m equations with n unknowns typically have no solution when $m > n$.

Even though Gaussian elimination will not give a solution to an inconsistent system $Ax = b$, we should not completely give up.

An alternative is to find a vector x that comes “closest” to being a solution.

Inconsistent system of equations

The matrix form of the previous system is $Ax = b$, i.e.,

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix}$$

or

$$x_1 \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + x_2 \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix}$$

In general, any $m \times n$ system $\boldsymbol{Ax} = \boldsymbol{b}$ can be viewed as a vector equation

$$x_1 v_1 + x_2 v_2 + \cdots + x_n v_n = \boldsymbol{b},$$

which expresses b as a linear combination of the columns v_i of $\boldsymbol{A}$, with coefficients $x_1, \ldots, x_n$.

Inconsistent system of equations

In our case, we are trying to hit the target vector $\boldsymbol{b}$ as a linear combination of two other 3-dimensional vectors.
Since the combinations of two 3-dimensional vectors form a plane inside $\mathbb{R}^3$, our system has a solution iff $\boldsymbol{b}$ lies in that plane.
There is a point in the plane $\boldsymbol{Ax}$ of all candidates that is closest to $\boldsymbol{b}$, denoted $A\overline{x}$, satisfying

$$\boldsymbol{b} - \boldsymbol{A\overline{x}} \perp \text{plane } \{\boldsymbol{Ax} | \boldsymbol{x} \in \mathbb{R}^n\}$$

Least squares is based on orthogonality.
The normal equations are a computational way to locate the line segment (the shortest distance from a point to a plane) which represents the least squares error.

Inconsistent system of equations

Indeed

$$b - A\overline{x} \perp \text{ plane } \{Ax | x \in \mathbb{R}^n\}$$

implies

$$(Ax)^T(b - A\overline{x}) = x^T A^T (b - A\overline{x}) = 0 \quad \text{for all } x \in \mathbb{R}^n$$

which happens iff

$$A^T(b - A\overline{x}) = 0$$

i.e., the normal equations

$$A^T A\overline{x} = A^T b$$

which gives the least squares solution to the system $Ax = b$.

Inconsistent system of equations

Example

Use the normal equations to find the least squares solution of the previous inconsistent system.

$$A = \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix}$$

and the components of the normal equations are

$$A^T A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}$$

and

$$A^T b = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix} = \begin{bmatrix} 6 \\ 4 \end{bmatrix}$$

Inconsistent system of equations

The normal equations

$$\begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 6 \\ 4 \end{bmatrix}$$

can be solved by Gaussian elimination to get

$$\overline{x} = \begin{bmatrix} \frac{7}{4} \\ \frac{3}{4} \end{bmatrix}$$

Substituting the least squares solution into the original problem yields

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} \frac{7}{4} \\ \frac{3}{4} \end{bmatrix} = \begin{bmatrix} 2.5 \\ 1 \\ 2.5 \end{bmatrix} \neq \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix}$$

Inconsistent system of equations

To measure the success at fitting the data,
we calculate the residual of the least-squares solution $\overline{x}$ as

$$r = b - A\overline{x} = \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix} - \begin{bmatrix} 2.5 \\ 1 \\ 2.5 \end{bmatrix} = \begin{bmatrix} -0.5 \\ 0.0 \\ 0.5 \end{bmatrix}$$

with the ℓ_2 norm

$$\|r\|_2 = \sqrt{0.5} \approx 0.707$$

and the root mean squared error

$$\|\frac{r}{m}\|_2 = \sqrt{\frac{0.5}{3}} = \frac{1}{\sqrt{6}} \approx 0.408.$$

Inconsistent system of equations

Example

Solve the least squares problem $\begin{bmatrix} 1 & -4 \\ 2 & 3 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -3 \\ 15 \\ 9 \end{bmatrix}$.

The normal equations $A^TAx = A^Tb$ are

$$\begin{bmatrix} 9 & 6 \\ 6 & 29 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 45 \\ 75 \end{bmatrix}.$$

The solution of the normal equations is $\overline{x} = [3.8; 1.8]$, and the residual vector is

$$r = b - A\overline{x} = \begin{bmatrix} -3 \\ 15 \\ 9 \end{bmatrix} - \begin{bmatrix} 1 & -4 \\ 2 & 3 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} 3.8 \\ 1.8 \end{bmatrix} = \begin{bmatrix} -3 \\ 15 \\ 9 \end{bmatrix} - \begin{bmatrix} -3.4 \\ 13 \\ 11.2 \end{bmatrix} = \begin{bmatrix} 0.4 \\ 2 \\ -2.2 \end{bmatrix}$$

with the euclidean norm $\|e\|_2 = 3$.

Conditioning of least squares

Since the data is assumed to be subject to errors in least squares problems, it is important to reduce error magnification.

The normal equations are the most straightforward approach to solving least squares problems, and it is fine for *small* problems.

However, the conditioning number

$$cond(A^T A) \approx (cond(A))^2$$

which will greatly increase the possibility that the problem is ill-conditioned.

Alternatives: QR factorization, SVD.

MATLAB's `backslash` command applied to $Ax = b$ ("$x = A\backslash b$")

- carries out Gaussian elimination if the system is consistent
- solves the least squares problem by QR factorization if inconsistent.

Matlab's `polyfit` command carries out least squares approximation to polynomials.

Conditioning of least squares

Example

Use the normal equations to find the least squares polynomial
$$P(x) = c_1 + c_2 x + \cdots + c_8 x^7$$
fitting the table (x_i, y_i), where $x_1 = 2.0, x_2 = 2.2, x_3 = 2.4, \ldots, x_{11} = 40$, are equally spaced in $[2,4]$ and $y_i = \sum_{k=0}^{7} x_i^k$, $1 \le i \le 11$.

A degree 7 polynomial is being fit to 11 data points lying on the degree 7 polynomial
$$P(x) = 1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + x^7.$$
Obviously, the correct least squares solution is
$$c_1 = c_2 = \cdots = c_8 = 1.$$
Substituting the data points into the model $P(x)$ yields the system $Ac = b$
$$\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^7 \\ 1 & x_2 & x_2^2 & \cdots & x_2^7 \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{11} & x_{11}^2 & \cdots & x_{11}^7 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_8 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{11} \end{bmatrix}$$

Conditioning of least squares

The coefficient matrix A is a *Van der Monde matrix*.
Solving the normal system in Matlab (in double precision) gives

$$c = [3.8938 \ -6.1483 \ 8.4914 \ -3.3182 \ 2.4788 \ 0.6990 \ 1.0337 \ 0.9984]$$

and the conditioning number is

$$cond(A' * A) = 1.3674e+019$$

Solving the normal equations in double precision cannot deliver an accurate value for the least squares solution. The conditioning number of A^TA is too large to deal with in double precision arithmetic, and the normal equations are ill-conditioned, even though the original least squares problem is moderately conditioned.

Nonpolynomial Example

The method of least squares is not restricted to first-degree polynomials.
Fit the table (x_k, y_k), $k = 0 : m$ by the function

$$y = a\ln x + b\cos x + ce^x$$

in the least squares sense. Then a, b, c are the unknowns and consider

$$\varphi(a,b,c) = \sum_{k=0}^{m} (a\ln x_k + b\cos x_k + ce^{x_k} - y_k)^2$$

and set $\frac{\partial\varphi}{\partial a} = \frac{\partial\varphi}{\partial b} = \frac{\partial\varphi}{\partial c} = 0$, i.e. the normal equations

$$\begin{cases} a\sum_{k=0}^{m}(\ln x_k)^2 & +b\sum_{k=0}^{m}(\ln x_k)(\cos x_k) & +c\sum_{k=0}^{m}(\ln x_k)e^{x_k} & =\sum_{k=0}^{m} y_k\ln x_k \\ a\sum_{k=0}^{m}(\ln x_k)(\cos x_k) & +b\sum_{k=0}^{m}(\cos x_k)^2 & +c\sum_{k=0}^{m}(\cos x_k)e^{x_k} & =\sum_{k=0}^{m} y_k\cos x_k \\ a\sum_{k=0}^{m}(\ln x_k)e^{x_k} & +b\sum_{k=0}^{m}(\cos x_k)e^{x_k} & +c\sum_{k=0}^{m}(e^{x_k})^2 & =\sum_{k=0}^{m} y_k e^{x_k} \end{cases}$$

Nonpolynomial Example of Least Squares

Example

Fit a function $y = a\ln x + b\cos x + ce^x$ to the values

x	0.24	0.65	0.95	1.24	1.73	2.01	2.23	2.52	2.77	2.99
y	0.23	−0.26	−1.10	−0.45	0.27	0.10	−0.29	0.24	0.56	1.00

$$\left\{\begin{array}{rrrr} 6.79410a & -5.34749b & +63.25889c & = 1.61627 \\ -5.34749a & +5.10842b & -49.00859c & = -2.38271 \\ 63.25889a & -49.00859b & +1002.50650c & = 26.77277 \end{array}\right.$$

with the solution

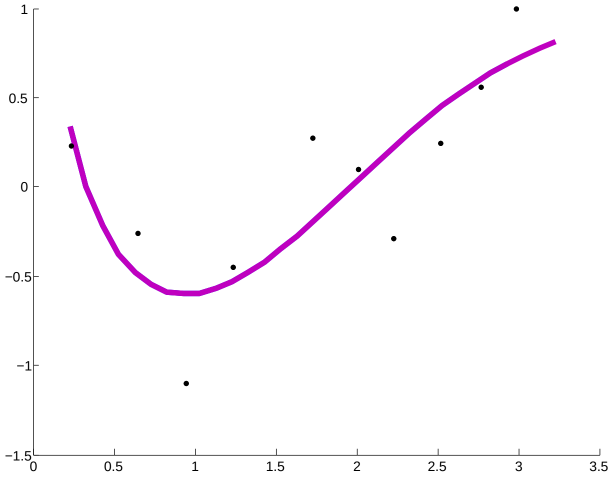
$$a = -1.04103, b = -1.26132, c = 0.03073,$$

hence the curve

$$y = -1.04103\ln x - 1.26132\cos x + 0.03073e^x$$

has the required form and fits the table in the least-squares sense, $\varphi(a, b, c) = 0.92557$.

Nonpolynomial Example



Basis Functions $\{g_0, g_1, \ldots, g_n\}$

Suppose that the data is to be fit to a function of the form

$$y = \sum_{j=0}^{n} c_j g_j(x)$$

where the functions

$$g_0, g_1, \ldots, g_n$$

(**basis functions**) are known,
and the coefficients

$$c_0, c_1, \ldots, c_n$$

are to be determined by the least squares principle.
Seek $c_0, c_1, \ldots, c_n$ that minimize

$$\varphi(c_0, c_1, \ldots, c_n) = \sum_{k=0}^{m} \left[\sum_{j=0}^{n} c_j g_j(x_k) - y_k \right]^2$$

Basis Functions $\{g_0, g_1, \dots, g_n\}$

The necessary conditions for the minimum are

$$\frac{\partial \varphi}{\partial c_i} = 0 \qquad i = 0 : n,$$

where

$$\frac{\partial \varphi}{\partial c_i} = \sum_{k=0}^{m} 2 \left(\sum_{j=0}^{n} c_j g_j(x_k) - y_k \right) g_i(x_k)$$

and can be rewritten as the **normal equations**

$$\sum_{j=0}^{n} \left(\sum_{k=0}^{m} g_i(x_k) g_j(x_k) \right) c_j = \sum_{k=0}^{m} y_k g_i(x_k), \quad i = 0 : n$$

The normal equations are linear in c_i; can be solved by Gaussian elimination.

Basis Functions $\{g_0, g_1, \dots, g_n\}$

In practice, the normal equations may be difficult to solve.

1. $\{g_0, g_1, \dots, g_n\}$ should be **linearly independent**.
2. $\{g_0, g_1, \dots, g_n\}$ should be *appropriate* to the problem at hand.
3. $\{g_0, g_1, \dots, g_n\}$ should be *well conditioned*.

Orthonormal Basis Functions $\{g_0, g_1, \ldots, g_n\}$

What basis for the vector space

$$\mathcal{G} = \{g : \exists c_0, c_1, \ldots, c_n \text{ such that } g(x) = \sum_{j=0}^{n} c_j g_j(x)\}$$

should we choose so that the normal equations

$$\sum_{j=0}^{n} \left(\sum_{k=0}^{m} g_i(x_k) g_j(x_k) \right) c_j = \sum_{k=0}^{m} y_k g_i(x_k), \quad i = 0 : n$$

can be *easily* and *accurately* solved?

The ideal situation: the coefficient matrix is the identity matrix, i.e., the basis $\{g_0, g_1, \ldots, g_n\}$ is orthonormal

$$\sum_{k=0}^{m} g_i(x_k) g_j(x_k) = \delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

Orthonormal Basis Functions $\{g_0, g_1, \dots, g_n\}$

In this case

$$c_j = \sum_{k=0}^{m} y_k g_j(x_k), \qquad j = 0 : n$$

So one could use the Gram-Schmidt procedure to orthonormalize $\mathcal{G}$.

Choose well the basis for the space $\mathcal{G}$!!!

Orthonormal Basis Functions $\{g_0, g_1, \dots, g_n\}$

$\mathcal{G} = \mathcal{P}_n$, polynomials of degree $\le n$.
(An important example of the least-squares theory).
The natural basis:

$$g_0(x) = 1, \quad g_1(x) = x, g_2(x) = x^2, \qquad g_n(x) = x^n$$

so a general element $g \in \mathcal{G}$ writes

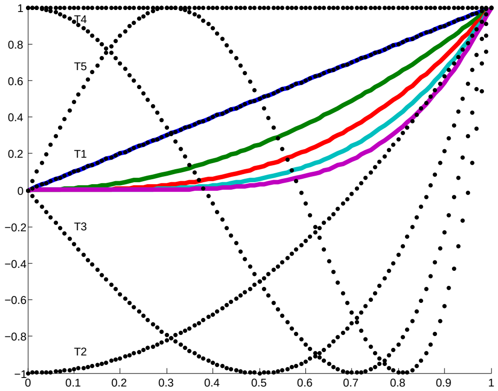
$$g(x) = \sum_{j=0}^{n} c_j g_j(x) = \sum_{j=0}^{n} c_j x^j = c_0 + c_1 x + c_2 x^2 + \cdots + c_n x^n$$

This basisi is almost always a *poor* choice for numerical work.

Orthonormal Basis Functions $\{g_0, g_1, \dots, g_n\}$

These functions are too much alike!!!

It would be difficult to determine precisely c_j in $g = \sum_{j=0}^{n} c_j x^j$.



Chebyshev Polynomials $\{g_0, g_1, \ldots, g_n\}$

1. Assume that the points in the least-squares problem have the property

$$-1 \leq x_0 < x_1 < \cdots < x_m < 1$$

The Chebyshev polynomials can be used:

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_2(x) &= 2x^2 - 1 \\ T_3(x) &= 4x^3 - 3x \\ T_4(x) &= 8x^4 - 8x^2 + 1, \ldots \end{aligned}$$

computed with a recursive formula

$$T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x), \quad \forall n \geq 2$$

Alternatively, $T_n(x) = \cos(k \,\text{arccos}(x))$.

Chebyshev Polynomials $\{g_0, g_1, \ldots, g_n\}$

2. If the original data do not satisfy

$$\min\{x_k\} = -1 \text{ and } \max\{x_k\} = 1$$

by lie in the interval $[a, b]$, use the change of variable

$$x = \frac{1}{2}(b-a)z + \frac{1}{2}(a+b).$$

The new variable traverses $[-1, 1]$ as x traverses $[a, b]$.

$$g(x) = \sum_{j=0}^{n} c_j T_j(x)$$

Smoothing data: Polynomial Regression

An application of the least-squares procedure: **smoothing** of data, i.e., fitting a "smooth" curve to a set of "noisy" values (experimental data containing errors).

- If one knows the type of function the data should conform,
  then use least-squares to compute any unknown parameters in the function.
- If one only wishes to smooth the data by fitting them with any convenient function,
  then polynomials of increasing degree can be used until a reasonable smoothness is attained.

Smoothing data: Polynomial Regression

"Smoothing" $=$ fitting a *smooth curve* to a set of "noisy" data.

For given data

x	x_0	x_1	$\cdots$	x_m
y	y_0	y_1	$\cdots$	y_m

one can determine a polynomial of degree m that passes through these points **exactly**.
If the points are 'noisy', it is better to seek a lower-degree polynomial that fits the data **approximately** in the least squares sense (curvilinear regression).

We do **not** know what degree of polynomial should be used.

Smoothing data: Polynomial Regression

Assume that there is a polynomial

$$p_N(x) = \sum_{i=0}^{N} a_i x^i$$

that represents the trend of the data and
the data satisfies the equation

$$y_i = p_N(x_i) + \varepsilon_i, \qquad i = 0 : m$$

where ε_i is the error in y_i.
Assume that ε_i are independent random variables that are normally distributed.

For a fixed n, the least squares method provides p_n.

Smoothing data: Polynomial Regression

Once p_n known,
one can compute the **variance**

$$\sigma_n^2 = \frac{1}{m-n}\sum_{i=0}^{n}[y_i - p_n(x_i)]^2 \qquad (m > n)$$

Statistical theory says that if the trend of the data table is given by p_N, then

$$\sigma_0^2 > \sigma_1^2 > \cdots > \sigma_N^2 = \sigma_{N+1}^2 = \sigma_{N+2}^2 = \cdots = \sigma_{m-1}^2$$

Strategy !?!

Smoothing data: Polynomial Regression

Back to the least squares problem:
say that F is the function to fit by a polynomial p_n of degree n.
We will find the polynomial that minimizes

$$\sum_{i=0}^{m}[F(x_i) - p_n(x_i)]^2$$

Orthogonality $(F(x_i) - p_n(x_i) \perp p_n(x_i))$ implies:
the solution is given by the formulae

$$p_n = \sum_{i=0}^{n} c_i q_i, \quad c_i = \frac{\langle F, q_i\rangle}{\langle q_i, q_i\rangle}$$

where $\{q_i\}_{i=0}^n$ is an orthogonal basis of $\mathcal{P}^n$.
(The ℓ_2 inner product $\langle f, g\rangle = \sum_{i=0}^{m} f(x_i)g(x_i)$.)

Smoothing data: Polynomial Regression

The algorithm to compute the variance

$$\sigma_n^2 = \frac{1}{m-n}\rho_n, \qquad \rho_n = \sum_{i=0}^{m}[y_i - p_n(x_i)]^2$$

is given by

$$\rho_0 = \langle F, F\rangle - \frac{\langle F, q_0\rangle^2}{\langle q_0, q_0\rangle}$$

$$\rho_n = \rho_{n-1} - \frac{\langle F, q_n\rangle^2}{\langle q_n, q_n\rangle}$$

Indeed,

$$\rho_n = \langle F - p_n, F - p_n\rangle = \langle F - p_n, F\rangle = \langle F, F\rangle - \langle F, p_n\rangle = \langle F, F\rangle - \sum_{i=0}^{n} c_i \langle F, q_i\rangle$$

$$= \langle F, F\rangle - \frac{\langle F, q_i\rangle^2}{\langle q_i, q_i\rangle}$$

SVD and Least Squares

Suppose that a given $m \times n$ matrix has the factorization

$$\boldsymbol{A = UDV^T}$$

where

- $\boldsymbol{U = [u_1, u_2, \cdots, u_m]}$ is an $m \times m$ orthogonal matrix
- $\boldsymbol{V = [v_1, v_2, \ldots, v_n]}$ is an $n \times n$ orthogonal matrix
- $\boldsymbol{D}$ is $m \times n$ diagonal matrix containing the singular values of $\boldsymbol{A}$ on its diagonal, in decreasing order.

The singular values of $\boldsymbol{A}$ are the positive square roots of the eigenvalues of $\boldsymbol{A^T A}$, namely

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r \geq 0.$$

$[U, S, V] = \texttt{svd}(A)$

$$\boldsymbol{U^T A V} = \boldsymbol{D} = \begin{bmatrix} \sigma_1 & & & & & & \\ & \sigma_2 & & & & & \\ & & \ddots & & & & \\ & & & \sigma_r & & & \\ & & & & 0 & & \\ & & & & & \ddots & \\ & & & & & & 0 \\ & & & & & & \end{bmatrix}_{m \times n}$$

where $\boldsymbol{U^T U} = \boldsymbol{I_m}, \boldsymbol{V^T V} = \boldsymbol{I_n}$.
Moreover,

1. $\boldsymbol{A v_i} = \sigma_i \boldsymbol{u_i}$
2. $\sigma_i = \|\boldsymbol{A v}_i\|_2$

where $\boldsymbol{v}_i$ is the ith column of $\boldsymbol{V}$ and u_i is the ith column of $\boldsymbol{U}$.

$$\begin{aligned}
\|\boldsymbol{Ax}-\boldsymbol{b}\|_2^2 &= \|\boldsymbol{U^T}(\boldsymbol{Ax}-\boldsymbol{b})\|_2^2 = \|\boldsymbol{U^T Ax}-\boldsymbol{U^T b}\|_2^2 \qquad (\boldsymbol{U} \text{ is orthogonal})\\
&= \|\boldsymbol{U^T A}(\boldsymbol{VV^T})\boldsymbol{x}-\boldsymbol{U^T b}\|_2^2 \qquad (\boldsymbol{V} \text{ is orthogonal})\\
&= \|(\boldsymbol{U^T AV})(\boldsymbol{V^T x})-\boldsymbol{U^T b}\|_2^2 = \|\boldsymbol{DV^T x}-\boldsymbol{U^T b}\|_2^2 = \|\boldsymbol{Dy}-\boldsymbol{c}\|_2^2\\
&= \sum_{i=1}^{r}(\sigma_i y_i - c_i)^2 + \sum_{i=r+1}^{m} c_i^2
\end{aligned}$$

where $\boldsymbol{y} = \boldsymbol{V^T x}$ and $\boldsymbol{c} = \boldsymbol{U^T b}$.

The least squares solution

$$\boldsymbol{x}_{LS} = V\begin{bmatrix} c_1/\sigma_1 \\ \vdots \\ c_1/\sigma_1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \sum_{i=1}^{n} y_i \boldsymbol{v}_i = \sum_{i=1}^{r} \sigma_i^{-1} c_i \boldsymbol{v}_i = \sum_{i}^{r} \sigma_i^{-1}(\boldsymbol{u}_i^T \boldsymbol{b})\boldsymbol{v}_i$$

The residual

$$\|\boldsymbol{Ax}_{LS}-\boldsymbol{b}\|_2^2 = \sum_{i=r+1}^{m} c_i^2 = \sum_{i=r+1}^{m} (\boldsymbol{u}_i^T \boldsymbol{b})^2$$

What about the conditioning??

SVD and Least Squares

Example

Find the least squares solution of the nonsquare system $\begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}$ using Singular Value Decomposition.

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{3}\sqrt{6} & 0 & \frac{1}{3}\sqrt{3} \\ \frac{1}{6}\sqrt{6} & \frac{1}{2}\sqrt{2} & -\frac{1}{3}\sqrt{3} \\ \frac{1}{6}\sqrt{6} & -\frac{1}{2}\sqrt{2} & -\frac{1}{3}\sqrt{3} \end{bmatrix} \begin{bmatrix} \sqrt{3} & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{2}\sqrt{2} & \frac{1}{2}\sqrt{2} \\ \frac{1}{2}\sqrt{2} & -\frac{1}{2}\sqrt{2} \end{bmatrix}$$

$r = \text{rank}(\boldsymbol{A})$ =2 and the singular values are $\sigma_1 = \sqrt{3}, \sigma_2 = 1$.

$$c_1 = \boldsymbol{u_1^T b} = \begin{bmatrix} \frac{1}{3}\sqrt{6} & \frac{1}{6}\sqrt{6} & \frac{1}{6}\sqrt{6} \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} = \frac{1}{3}\sqrt{6}$$

$$c_2 = \boldsymbol{u_2^T b} = \begin{bmatrix} 0 & -\frac{1}{2}\sqrt{2} & \frac{1}{2}\sqrt{2} \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} = \sqrt{2}$$

SVD and Least Squares

and

$$\begin{aligned} x_{LS} &= (\sigma_1^{-1}c_1)\boldsymbol{v}_1 + (\sigma_2^{-1}c_2)\boldsymbol{v}_2 \\ &= \frac{1}{\sqrt{3}}\left(\frac{1}{3}\sqrt{6}\right)\begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} + \sqrt{2}\begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{3} \\ \frac{1}{3} \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} \frac{4}{3} \\ -\frac{2}{3} \end{bmatrix} \end{aligned}$$

which is the same solution as that from the normal equations.

The generalized inverse: $A^{+} = pinv(\mathtt{A})$

For the matrix

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^{\boldsymbol{T}}$$

we define its *pseudo-inverse* $\boldsymbol{A}^{+}$ by

$$\boldsymbol{A}^{+} = \boldsymbol{V}\boldsymbol{D}^{+}\boldsymbol{U}^{\boldsymbol{T}}$$

where the pseudo-inverse of the diagonal matrix $\boldsymbol{D}$ is

$$\boldsymbol{D} = \begin{bmatrix} \sigma_1 & & & & & & \\ & \sigma_2 & & & & & \\ & & \ddots & & & & \\ & & & \sigma_r & & & \\ 0 & 0 & \cdots & 0 & 0 & & \\ \vdots & \vdots & & & & \ddots & \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & & & & & & \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \end{bmatrix}_{m\times n}, \quad \boldsymbol{D}^{+} = \begin{bmatrix} \frac{1}{\sigma_1} & & & & 0 & & 0 & \cdots & 0 \\ & \frac{1}{\sigma_2} & & & 0 & & 0 & \cdots & 0 \\ & & \ddots & & \vdots & & & & \\ & & & \frac{1}{\sigma_r} & 0 & & 0 & \cdots & 0 \\ & & & & 0 & & & & \\ & & & & & \ddots & & & \\ & & & & & & 0 & \cdots & 0 \end{bmatrix}_{n\times m}$$

When $\boldsymbol{A}$ is a square matrix, $\boldsymbol{A}^{-1} = \boldsymbol{A}^{+}$

Gram-Schmidt orthogonalization

Let $v_1, \ldots, v_k$ be linearly independent vectors in $\mathbb{R}^n$.

$$y_1 := q_1 = \frac{v_1}{\|v_1\|_2}$$

Note that

$$v_1 = r_{11} q_1$$

where $r_{11} = \|v_1\|_2$ and q_1 is a unit vector in the direction of v_1.
To find the second unit vector, define

$$y_2 := v_2 - q_1(q_1^T v_2)$$

and

$$q_2 = \frac{y_2}{\|y_2\|_2}.$$

Note that

$$v_2 = y_2 + q_1(q_1^T v_2) = r_{22} q_2 + r_{12} q_1$$

where $r_{22} = \|y_2\|_2$ and $r_{12} = q_1^T v_2$.

Gram-Schmidt orthogonalization

In general, define

$$y_i := v_i - q_1(q_1^T v_i) - q_2(q_2^T v_1) - \cdots - q_{i-1}(q_{i-1}^T v_i)$$

and

$$q_i = \frac{y_i}{\|y_i\|_2}$$

which implies

$$v_i = r_{ii} q_i + r_{1i} q_1 + \cdots + r_{i-1,i} q_{i-1},$$

where $r_{ii} = \|y_i\|$ and $r_{ji} = q_j^T v_i$ for $j = 1, \ldots, i$.

Gram-Schmidt orthogonalization

Example

Apply Gram-Schmidt to the columns of $A = \begin{bmatrix} 1 & -4 \\ 2 & 3 \\ 2 & 2 \end{bmatrix}$.

Set $y_1 = v_1 = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}$.

Then

$$r_{11} = \|y_1\|_2 = \sqrt{1^2 + 2^2 + 2^2} = 3,$$

and the first unit vector is

$$q_1 = \frac{v_1}{\|v_1\|_2} = \begin{bmatrix} \frac{1}{3} \\ \frac{2}{3} \\ \frac{2}{3} \end{bmatrix}.$$

Gram-Schmidt orthogonalization

To find the second vector, set

$$y_2 = v_2 - q_1 q_1^T v_2 = \begin{bmatrix} -4 \\ 3 \\ 2 \end{bmatrix} - \begin{bmatrix} \frac{1}{3} \\ \frac{2}{3} \\ \frac{2}{3} \end{bmatrix} 2 = \begin{bmatrix} -\frac{14}{3} \\ \frac{5}{3} \\ \frac{2}{3} \end{bmatrix}$$

and

$$q_2 = \frac{y_2}{\|y_2\|_2} = \frac{1}{5} \begin{bmatrix} -\frac{14}{3} \\ \frac{5}{3} \\ \frac{2}{3} \end{bmatrix} = \begin{bmatrix} -\frac{14}{15} \\ \frac{1}{3} \\ \frac{2}{15} \end{bmatrix}$$

with $r_{12} = 2, r_{22} = 5$.

Gram-Schmidt orthogonalization - QR factorization

For the $n \times k$ matrix $\boldsymbol{A} = (\boldsymbol{v}_1 | \cdots | \boldsymbol{v}_k)$, we have the QR factorization $\boldsymbol{A} = \boldsymbol{QR}$

$$(\boldsymbol{v}_1 | \cdots | \boldsymbol{v}_k) = (\boldsymbol{q}_1 | \cdots | \boldsymbol{q}_n) \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1k} \\ & r_{22} & \cdots & r_{2k} \\ & & \ddots & \vdots \\ & & & r_{kk} \\ 0 & \cdots & \cdots & 0 \\ \vdots & & & \vdots \\ 0 & \cdots & \cdots & 0 \end{bmatrix}$$

where

- $\boldsymbol{Q}$ is a $n \times n$ orthogonal matrix and
- the upper triangular matrix $\boldsymbol{R}$ is $n \times k$, same as $\boldsymbol{A}$.

Gram-Schmidt orthogonalization - QR factorization

Remark

1. *The key property of an orthogonal matrix is that it preserves the Euclidean norm of a vector.*
2. *Doing calculations with orthogonal matrices is preferable because*
   - *they are easy to invert by definition*
   - *they do not magnify errors.*

The previous example

$$\begin{bmatrix} 1 & -4 \\ 2 & 3 \\ 2 & 2 \end{bmatrix} = QR = \frac{1}{15}\begin{bmatrix} 5 & -14 & 2 \\ 10 & 5 & 10 \\ 10 & 2 & -11 \end{bmatrix}\begin{bmatrix} 3 & 2 \\ 0 & 5 \\ 0 & 0 \end{bmatrix}.$$

In MATLAB

$$[Q,R] = \mathtt{qr}(A)$$

Least Squares by QR factorization: $\|Ax-b\|_2 = \|QRx-b\|_2 = \|Rx - \underbrace{Q^T}_{d}\|_2$

$$\begin{bmatrix} e_1 \\ \vdots \\ e_n \\ e_{n+1} \\ \vdots \\ e_m \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ & r_{22} & \cdots & r_{2n} \\ & & \ddots & \vdots \\ & & & r_{nn} \\ 0 & \cdots & \cdots & 0 \\ \vdots & & & \vdots \\ 0 & \cdots & \cdots & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} - \begin{bmatrix} d_1 \\ \vdots \\ d_n \\ d_{n+1} \\ \vdots \\ d_m \end{bmatrix}$$

and $\|e\|_2^2 = d_{n+1}^2 + \cdots d_m^2$.

Given the $m \times n$ inconsistent system $Ax = b$, factor $A = QR$ and set

- $\hat{R} =$ upper $n \times n$ submatrix of R
- $\hat{d} =$ upper n entries of $d = Q^T b$

Solve $\hat{R}\overline{x} = \hat{d}$ for least squares solution $\overline{x}$.

What about the conditioning??

Boundary Value Problems

Seek
$$y = y(x)$$
that passes through
$$(0,1) \text{ and } (1,a),$$
and minimizes
$$\int_0^1 \left(y(y')^2 + b(x)y^2\right)dx$$
in which $b(x)$ is a known function.
The Euler equation with its initial and terminal values:
$$\begin{cases} -(y')^2 - 2b(x)y + 2yy'' = 0 \\ y(0) = 1, \qquad y(1) = a. \end{cases}$$

Boundary Value Problems

$$\begin{cases} x'' = f(t,x,x') \quad \text{on } (a,b) \\ A\begin{bmatrix} x(a) \\ x'(b) \end{bmatrix} + B\begin{bmatrix} x(b) \\ x'(b) \end{bmatrix} = \begin{bmatrix} \gamma_1 \\ \gamma_2 \end{bmatrix} \end{cases}, \quad A, B \in \mathscr{M}_{2\times 2}(\mathbb{R}). \tag{14.1}$$

The BVPs may not be solvable or uniquely solvable.

Example

Example

Find the height trajectory of a projectile that begins at the top of a 120-foot building and reaches the ground 3 seconds later.

Let $y(t)$= height at time t.
From Newton's second law

$$F = my'',$$

where $F = -mg$ and $g = 32\ ft/sec^2$.
The trajectory can be expressed as the solution of the IVP

$$\begin{cases} y'' = -g \\ y(0) = 120 \\ y'(0) = v_0 \end{cases}$$

or the BVP

$$\begin{cases} y'' = -g \\ y(0) = 120 \\ y(3) = 0 \end{cases}$$

Example

Since we don't know the initial velocity v_0, we must solve the BVP.

$$\begin{aligned} y(t) &= -\frac{1}{2}gt^2 + v_0 t + y_0 \\ &= -16t^2 + v_0 t + y_0. \end{aligned}$$

Using the boundary conditions yields

$$\begin{aligned} 120 &= y(0) = y_0 \\ 0 &= y(3) = -144 + 3v_0 + 120, \end{aligned}$$

which implies

$$v_0 = 8 \ ft/sec.$$

The solution trajectory is

$$y(t) = -16t^2 + 8t + 120$$

Boundary Value Problems

Counterexample 1°

$$\begin{cases} x'' = -\lambda x \quad t \in (0,1) \\ x(0) = x(1) = 0 \end{cases}$$ linear undamped pendulum $(\star)$

The general solution to $x'' + \lambda x = 0$ is:

$$x(t) = \begin{cases} c_1 e^{\mu t} + c_2 e^{-\mu t}, & \lambda < 0, \mu = (-\lambda)^{\frac{1}{2}} \\ c_1 + c_2 t, & \lambda = 0 \\ c_1 \cos(\mu t) + c_2 \sin(\mu t), & \lambda > 0, \mu = \lambda^{\frac{1}{2}} \end{cases}$$

- If $\lambda \leq 0$, then BVP's boundary conditions imply $c_1 = c_2 = 0$ and $x \equiv 0$ the unique solution to $(\star)$.
- If $\lambda > 0$, then $c_1 = 0, c_2 \sin(\mu) = 0$, i.e., $\mu = \pm\pi, \pm 2\pi, \pm 3\pi, \ldots$ and $\lambda = \pi^2, 4\pi^2, \ldots$ so $\aleph_0$ solutions.

Counterexample 2°

$$\begin{cases} x'' = -\lambda x + g(t) & t \in (0,1) \\ x(0) = x(1) = 0 \end{cases} \qquad \text{linear damped pendulum} \qquad (\star\star)$$

- If $\lambda \notin \{\pi^2, 4\pi^2, \dots\}$, then $\exists!$ unique solution:

$$x(t) = \int_0^1 G(t,s) g(s) ds \quad \text{(Green's formula)}$$

$$G(t,s) = \begin{cases} G_0 \equiv -\frac{\sin(\mu s)\sin\mu(1-t)}{\mu \sin\mu}, & t \geq s \\ G_1 \equiv -\frac{\sin(\mu t)\sin\mu(1-s)}{\mu \sin\mu}, & t \leq s \end{cases}$$

- If $\lambda \in \{\pi^2, 4\pi^2, \dots\}$, then in general there is no solution. Let $\lambda = \pi^2$. Then $(\star\star)$ has a solution iff

$$\int_0^1 g(t)\sin(\pi t) dt = 0$$

which gives the general solution

$$x(t) = c\sin(\pi t) + \frac{1}{\pi}\int_0^1 g(s)\sin(\pi(t-s))ds, \quad c \in \mathbb{R}.$$

Shooting method

2 Point BVP

$$\begin{cases} x''(t) - p(t)x'(t) - q(t)x(t) = 0, & a \le t \le b \\ x(a) = \alpha \\ x(b) = \beta \end{cases} \tag{14.2}$$

Associate to this problem the following IVP$_s$

IVP

$$\begin{cases} x'' - px' - qx = 0, & a \le t \le b \\ x(a) = \alpha \\ x'(a) = s \end{cases} \tag{14.3}$$

Let $x(s;t)$ be the solution of the (IVP)$_s$, and write

$$x(s;t) = c_1 x_1(t) + c_2 x_2(t),$$

where x_1, x_2 are independent.

$$\left.\begin{array}{ll} x_1(a)=1 & x_2(a)=0 \\ x_1'(a)=0 & x_2'(a)=1 \end{array}\right\} \Rightarrow c_1=\alpha, c_2=s, x(s;t) = \alpha x_1(t) + s x_2(t).$$

The Boundary Conditions: $\begin{cases} t=a: & \alpha = \alpha x_1(a) + s x_2(a), \\ t=b: & \beta = \alpha x_1(b) + s x_2(b). \end{cases}$

1. We can solve for s if $x_2(b) \neq 0 : s = \frac{\beta - \alpha x_1(b)}{x_2(b)}$ and the solution to (14.2).
2. What if $x_2(b) = 0$?
   1. either $\alpha x_1(b) = \beta$: there are an infinite # of solutions,
   2. or $\alpha x_1(b) \neq \beta$: there is **no** solution.

Theorem

$$\begin{cases} x'' = px' + qx + g \quad \text{on } (a,b) \\ A \begin{bmatrix} x(a) \\ x'(b) \end{bmatrix} + B \begin{bmatrix} x(b) \\ x'(b) \end{bmatrix} = \begin{bmatrix} \gamma_1 \\ \gamma_2 \end{bmatrix}. \end{cases}$$

$\exists!$ *solution for each set* (g, γ_1, γ_2) **iff** *the homogeneous problem* $(g = \gamma_1 = \gamma_2 = 0)$ *has a unique solution (trivial solution* $x \equiv 0$*).*

Shooting method

Let define the function

$$\varphi(s) = x(b)$$

where x is the solution on $[a, b]$ of the IVP

$$\begin{cases} x'' = f(t, x, x') & \text{on } (a, b) \\ x(a) = \alpha, \quad x'(a) = s \end{cases}$$

The goal of the shooting method is to adjust s until we find a value for which

$$\varphi(s) = \beta$$

Given two guesses s_1 and s_2 for $x'(a)$, use linear interpolation between $\varphi(s_1)$ and $\varphi(s_2)$:

$$\frac{s_3 - s_2}{\beta - \varphi(s_2)} = \frac{s_2 - s_1}{\varphi(s_2) - \varphi(s_1)}$$

which implies

$$s_3 = s_2 + (\beta - \varphi(s_2)) \frac{s_2 - s_1}{\varphi(s_2) - \varphi(s_1)}$$

Shooting method algorithm

We can generate the sequence

$$s_{n+1} = s_n + (\beta - \varphi(s_n))\frac{s_n - s_{n-1}}{\varphi(s_n) - \varphi(s_{n-1})}$$

based on two starting points s_1, s_2.
Monitor

$$\varphi(s_{n+1}) - \beta$$

and stop when this is sufficiently small.

Example

What is the function $\varphi(s)$ for this BVP

$$\begin{cases} x'' = x & \text{on } (0,1) \\ x(0) = 1, & x(1) = 7 \end{cases}$$

The solution to the DE is

$$x(t) = c_1 e^t + c_2 e^{-t}$$

and the solution to the IVP

$$\begin{cases} x'' = x & \text{on } (0,1) \\ x(0) = 1, & x'(0) = s \end{cases}$$

is

$$x(t) = \frac{1}{2}(1+s)e^t + \frac{1}{2}(1-s)e^{-t}$$

which yields

$$\varphi(s) = x(1) = \frac{1}{2}(1+s)e + \frac{1}{2}(1-s)e^{-1}$$

Finite difference method to solve BVP

$$\begin{cases} x'' = f(t, x, x') \\ x(a) = \gamma_1 \\ x(b) = \gamma_2 \end{cases}$$

Fix $N \geq 1$ and let $h = \frac{b-a}{N}, t_j = a + jh, j = 0, \dots, N$.

$$x''(t_i) = f(t_i, x(t_i), x'(t_i)).$$

Approximate the derivatives by Finite Differences:

$$x''(t_i) \simeq \frac{x(t_{i+1}) - 2x(t_i) + x(t_{i-1})}{h^2}, \qquad \texttt{error = } O(h^2), \quad (\textbf{??})$$

$$x'(t_i) \simeq \frac{x(t_{i+1}) - x(t_{i-1})}{2h}, \texttt{error = } O(h^2) \qquad (10.7)$$

(standard central differences formulae)

Finite difference method to solve BVP

Find the sequence $\{x_i\}_{0\le i\le N}$:

FD scheme
$$\begin{cases} x_0 = \gamma_1, \\ \frac{x_{i+1}-2x_i+x_{i-1}}{h^2} = \underbrace{f(t_i, x_i, \frac{x_{i+1}-x_{i-1}}{2h})}_{\equiv f_i}, \quad 1\le i\le N-1 \\ x_N = \gamma_2. \end{cases}$$

Let

$$\boldsymbol{x_h} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{N-1} \end{bmatrix}, \boldsymbol{A} = \begin{bmatrix} -2 & 1 & & 0 \\ 1 & \ddots & & \\ & & \ddots & 1 \\ 0 & & 1 & -2 \end{bmatrix}, \boldsymbol{b(x)} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{N-1} \end{bmatrix}, \boldsymbol{g} = \begin{bmatrix} -\frac{\gamma_1}{h^2} \\ 0 \\ \vdots \\ -\frac{\gamma_2}{h^2} \end{bmatrix}$$

$$\frac{1}{h^2}\boldsymbol{Ax} = \boldsymbol{b(x)} + \boldsymbol{g}. \tag{14.4}$$

$n-1$ nonlinear equations with $n-1$ unknowns

Finite difference method to solve BVP

The result: $\max_i |x_i - x(t_i)| = O(h^2)$.

If f is nonlinear in $\boldsymbol{x}$, use Newton-Raphson's method to solve (14.4)

(the method will converge quadratically to a root α of (14.4) if $J(\alpha)$ is nonsingular and $x^{(0)}$ is close to the root α, i.e. $\|\alpha -$

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \left(J(\boldsymbol{x}^{(k)})\right)^{-1}\left(\frac{1}{h^2}\boldsymbol{A}\boldsymbol{x}^k - \boldsymbol{b}(\boldsymbol{x}^k) - \boldsymbol{g}\right)$$

where $J(\boldsymbol{x}^{(k)})$, the Jacobian matrix, is tridiagonal:

$$J(\boldsymbol{x}^{(k)}) = \frac{1}{h^2}\boldsymbol{A} - \begin{bmatrix} \frac{\partial f_1}{\partial u} & \frac{1}{2h}\frac{\partial f_1}{\partial v} & & 0 \\ -\frac{1}{2h}\frac{\partial f_2}{\partial v} & \frac{\partial f_2}{\partial u} & & \\ & & \ddots & \frac{1}{2h}\frac{\partial f_{N-2}}{\partial v} \\ 0 & & -\frac{1}{2h}\frac{\partial f_{N-1}}{\partial v} & \frac{\partial f_{N-1}}{\partial u} \end{bmatrix}.$$

The matrix $J(\boldsymbol{x}^{(k)})$ can be kept constant several iterations (to save on calculations) but the convergence will be linear.

Finite difference method to solve BVP

If we write $f = f(t,u,v)$, $f_i \equiv f(t_i, x_i, \frac{x_{i+1}-x_{i-1}}{2h})$, compute $\frac{\partial f_i}{\partial x_j}$:

$$\frac{\partial}{\partial x_j}\Big(f(t_i, x_i, \frac{x_{i+1}-x_{i-1}}{2h})\Big) = 0 \text{ if } j \neq \{i, i-1, i+1\}$$

$$\frac{\partial}{\partial x_i}\Big(f(t_i, x_i, \frac{x_{i+1}-x_{i-1}}{2h})\Big) = \frac{\partial f}{\partial u}(t_i, x_i, \frac{x_{i+1}-x_{i-1}}{2h})$$

$$\frac{\partial}{\partial x_{i-1}}\Big(f(t_i, x_i, \frac{x_{i+1}-x_{i-1}}{2h})\Big) = \frac{\partial f}{\partial v}(t_i, x_i, \frac{x_{i+1}-x_{i-1}}{2h}) \cdot \Big(-\frac{1}{2h}\Big)$$

$$\frac{\partial}{\partial x_{i+1}}\Big(f(t_i, x_i, \frac{x_{i+1}-x_{i-1}}{2h})\Big) = \frac{\partial f}{\partial v}(t_i, x_i, \frac{x_{i+1}-x_{i-1}}{2h}) \cdot \Big(+\frac{1}{2h}\Big)$$

Finite difference method. The linear case

When the RHS f is linear

$$f(t, x, x') = u(t) + v(t)x + w(t)x'$$

then the principal equation writes

$$\frac{1}{h^2}(x_{i+1} - 2x_i + x_{i-1}) = \underbrace{u(t_i)}_{u_i} + \underbrace{v(t_i)}_{v_i} x_i + \underbrace{w(t_i)}_{w_i} \left[\frac{1}{2h}(x_{i+1} - x_{i-1})\right]$$

i.e.

$$\underbrace{-\left(1 + \frac{h}{2}w_i\right)}_{a_i} x_{i-1} + \underbrace{\left(2 + h^2 v_i\right)}_{d_i} x_i - \underbrace{\left(1 - \frac{h}{2}w_i\right)}_{c_i} x_{i+1} = \underbrace{-h^2 u_i}_{b_i}$$

$$a_i x_{i-1} + d_i x_i + c_i x_{i+1} = b_i, \quad i = 1 : N-1$$

Finite difference method. The linear case

$$a_i x_{i-1} + d_i x_i + c_i x_{i+1} = b_i, \quad i = 1 : N-1$$

hence

$$\begin{cases} & d_1 x_1 & +c_1 x_2 & = b_1 - a_1\gamma_1 & \\ a_i x_{i-1} + & d_i x_i & +c_i x_{i+1} & = b_i & 2 \le i \le n-2 \\ a_{n-1} x_{n-2} + & d_{n-1} x_{n-1} & & = b_{n-1} - c_{n-1}\gamma_2 & \end{cases}$$

and in matrix form

$$\begin{bmatrix} d_1 & c_1 & & & & \\ a_2 & d_2 & c_2 & & & \\ & a_3 & d_3 & c_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & a_{n-2} & d_{n-2} & c_{n-2} \\ & & & & a_{n-1} & d_{n-1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-2} \\ x_{n-1} \end{bmatrix} = \begin{bmatrix} b_1 - a_1\gamma_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{n-2} \\ b_{n-1} - c_{n-1}\gamma_2 \end{bmatrix}$$

Finite difference method. Pseudocode and numerical example

Example

$x(t) = e^t - 3\cos(t)$

$$\begin{cases} x'' = e^t - 3\sin(t) + x' - x \\ x(1) = 1.09737491, \quad x(2) = 8.63749661 \end{cases}$$

See the pseudocode online.

Collocation and the Finite Element Method

Like the Finite Difference Method, the idea behind

- **Collocation** and the
- **Finite Element Method**

is to reduce the BVP to a set of solvable equations.
However, instead of discretizing the DE by replacing derivatives with finite differences, the solution is given a functional form whose parameters are fit by the method.

Choose a set of basis functions

$$\phi_1(t), \ldots, \phi_n(t),$$

which may be polynomials, trigonometric functions, splines, or other simple functions.
Then consider the possible solution

$$y(t) = c_1\phi_1(t) + \cdots + c_n(t)\phi_n(t) \tag{14.5}$$

Finding an approximate solution reduces to determining values for c_i.
We will consider two different ways to find the coefficients.

Collocation and the Finite Element Method

1. The **Collocation** approach is to substitute (14.5) into the BVP and evaluate at grid points. This method is straightforward, reducing the problem to solving a system of equations, linear if the original problem was linear.
   Each point gives an equation, and solving them for c_i is a type of interpolation.
2. The **FEM** proceeds by treating the fitting as a least squares problem instead of interpolation.
   The **FEM** employs the Galerkin projection to minimize the difference between (14.5) and the exact solution in the sense of squared error.

Collocation

Choosing n points, beginning and ending with the boundary points a and b

$$a = t_1 < t_2 < \cdots < t_n = b$$

the **collocation method** works by substituting the candidate solution (14.5) into the DE and evaluating the DE at these points to get n equations in the n unknowns $c_1, \ldots, c_n$.

Example

Solve by the Collocation Method the BVP $\begin{cases} y'' = 4y \\ y(0) = 1 \\ y(1) = 3 \end{cases}$

Start simple, choose the basis $\phi_j(t) = t^{j-1}, 1 \leq j \leq n$.
The solution will be of the form

$$y(t) = \sum_{j=1}^{n} c_j \phi_j(t) = \sum_{j=1}^{n} c_j t^{j-1}$$

We will write n equations in the n unknowns $c_1, \ldots, c_n$.

Example

The last and the first are the boundary conditions

$$i = 1: \qquad c_1 = \sum_{j=1}^{n} c_j \phi_j(0) = y(0) = 1$$

$$i = n: \qquad c_1 + \cdots + c_n = \sum_{j=1}^{n} c_j \phi_j(1) = y(1) = 3$$

The remaining $n-2$ equations from the DE evaluated at t_i for $2 \le i \le n-1$. The differential equation applied to $y(t) = \sum_{j=1}^{n} c_j t^{j-1} = 0$ is

$$\sum_{j=1}^{n} (j-1)(j-2)c_j t^{j-3} - 4\sum_{j=1}^{n} c_j t^{j-1} = 0$$

Evaluating at t_i yields for each i

$$\sum_{j=1}^{n} [(j-1)(j-2)t_j^{j-3} - 4t_i^{j-1}]c_j = 0.$$

Example

The n equations form a linear system $Ac = g$, where the coefficient matrix A is

$$A_{ij} = \begin{cases} 1\ 0\ 0\ \dots\ 0 & \text{row } i = 1 \\ (j-1)(j-2)t_j^{j-3} - 4t_i^{j-1} & \text{rows } i = 2 \text{ through } n-1 \\ 1\ 1\ 1\ \dots\ 1 & \text{row } i = n \end{cases}$$

and

$$g = (1, 0, 0, \dots, 0, 3)^T.$$

Usually use a evenly-spaced grid

$$t_i = a + \frac{i-1}{n-1}(b-a) = \frac{i-1}{n-1}.$$

After solving for the c_i, we obtain the solution

$$y(t) = \sum c_i t^{i-1}.$$

Example

- For $n = 2$, the system $Ac = g$ is

$$\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \end{bmatrix},$$

  and the solution is $c = (1, 2)^T$. The approximate solution is the straight line

$$y(t) = c_1 + c_2 t = 1 + 2t.$$

- For $n = 4$ yields the approximate solution

$$y(t) \approx 1 - 0.886t + .0273t^2 + 1.1613t^3$$

  which is very close to the exact solution. ■

The system to be solved in this example was linear, because the DE is linear.

Nonlinear BVPs can be solved by collocation in a similar way, using e.g. Newton's method to solve the resulting nonlinear system of equations.

Collocation

Although we used monomial basis functions for simplicity, there are many better choices. Polynomial bases are not generally recommended.

- The fact that monomial basis elements t^j are not orthogonal to one another as functions makes the coefficient matrix of the linear system ill-conditioned when n is large.
  Using the roots of the Chebyshev polynomials as evaluation points, rather than evenly-spaced points, improves the conditioning.
- The choice of trigonometric functions as basis leads to Fourier analysis and **spectral methods**, which are heavily used for both BVP and PDEs. This is a "global" approach, where the basis are nonzero over a large range of t, but have good orthogonality properties.
- The choice of splines as a basis functions leads to the **Finite Element Method**. In this approach, each basis function is nonzero only over a short range of t. FEMs are heavily used for BVPs and PDEs in higher dimensions, especially when irregular boundaries make parametrization by standard basis functions inconvenient.

Finite Elements and the Galerkin Method

- In **collocation**, we assumed a functional form
$$y(t) = \sum c_i \phi(t)$$
and solved for the coefficients c_i by forcing the solution to satisfy
  - the boundary conditions and
  - exactly satisfy the DE at discrete points.
- The **Galerkin** approach
minimizes the squared error of the DE along the solution.
This leads to a different system of equations for c_i.

Consider the BVP
$$\begin{cases} y'' = f(t, y, y') \\ y(a) = y_a \\ y(b) = y_b \end{cases}$$
The goal is to choose the approximate solution y so that

the **residual** $r = y'' - f$ is as small as possible.

In analogy with the least squares methods, this is accomplished by choosing y to make the residual orthogonal to the vector space of potential solutions.

Finite Elements and the Galerkin Method

Recall $$L^2(a,b) = \left\{ y(t) \text{ defined on } (a,b) \,\middle|\, \exists \int_a^b y^2(t)dt < \infty \right\}$$
with the inner product

$$\langle y_1, y_2 \rangle = \int_a^b y_1(t) y_2(t) dt$$

(bilinear and symmetric, with the norm $\|y\|_2 = \langle y, y \rangle^{\frac{1}{2}}$).
Two functions y_1 and y_2 are **orthogonal** in $L^2(a,b)$ if

$$\langle y_1, y_2 \rangle = 0.$$

Since $L^2(a,b)$ is an infinite-dimensional vector space, we cannot make the residual $r = y'' - f$ orthogonal to all of $L^2(a,b)$.
However, we can choose a basis that spans as much of L^2 as possible with the available computational resource.

Finite Elements and the Galerkin Method

Let the set of $n+2$ basis functions be denoted by $\phi_0(t), \dots, \phi_{n+1}(t)$.
The Galerkin method consists of two main ideas:

1. Minimize r by forcing it to be orthogonal to the basis functions, in the sense of the L^2 inner product:
$$\int_a^b (y'' - f)\phi_i(t)dt = 0,$$
or (the **weak form** of the BVP)
$$\int_a^b y''(t)\phi_i(t)dt = \int_a^b f(t,y,y'))\phi_i(t), \qquad i = 1:n+1.$$

2. Use integration by parts to eliminate the second derivatives:
$$\begin{aligned}\int_a^b y''(t)\phi_i(t)dt &= \phi_i(t)y'(t)\Big|_a^b - \int_a^b y'(t)\phi_i'(t)dt \\ &= \phi_i(b)y'(b) - \phi_i(a)y'(a) - \int_a^b y'(t)\phi_i'(t)dt\end{aligned}$$

Finite Elements and the Galerkin Method

Using these together gives a set of equations

$$\int_a^b f(t, y, y'))\phi_i(t) = \phi_i(b)y'(b) - \phi_i(a)y'(a) - \int_a^b y'(t)\phi_i'(t)dt$$

for each i that can be solved for the c_i in the functional form

$$y(t) = \sum_{i=1}^{n+1} c_i\phi_i(t)$$

The two ideas of Galerkin make it convenient to use extremely simple functions as finite elements $\phi_i(t)$.

We introduce piecewise-linear B-splines.

Finite Elements and the Galerkin Method

Start with a grid $t_0 < t_1 < \cdots < t_n < t_{n+1}$ of points on the axis. For $i = 1, \ldots, n$ define

$$\phi_i(t) \begin{cases} \frac{t - t_{i-1}}{t_i - t_{i-1}} & \text{for } t_{i-1} < t \leq t_i \\ \frac{t_{i+1} - t}{t_{i+1} - t} & \text{for } t_i < t \leq t_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

and

$$\phi_0(t) = \begin{cases} \frac{t_1 - t}{t_1 - t_0} & \text{for } t_0 < t \leq t_1 \\ 0 & \text{otherwise} \end{cases} \qquad \phi_1(t) = \begin{cases} \frac{t - t_n}{t_{n+1} - t_n} & \text{for } t_n < t \leq t_{n+1} \\ 0 & \text{otherwise} \end{cases}$$

The piecewise-linear "tent" functions $\phi_i(t)$ satisfy the following property

$$\phi_i(t_j) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

Finite Elements and the Galerkin Method

For a set of data points (t_i, c_i), define the **piecewise-linear B-spline**

$$S(t) = \sum_{i=0}^{n+1} c_i \phi_i(t).$$

It follows immediately from the property of the tent functions that

$$S(t_j) = \sum_{i=0}^{n+1} c_i \phi_i(t_j) = c_j$$

hence $S(t)$ is a piecewise-linear function that interpolates the data points (t_i, c_i).
In other words, the y-coordinates are the coefficients!
This simplifies the interpretation of the solution.
The c_i **are not only the coefficients, but also the solution values at the grid points** t_i**.**

Finite Elements and the Galerkin Method

Example

Apply the Finite Element Method to the BVP

$$\begin{cases} y'' = 4y \\ y(0) = 1 \\ y(1) = 3 \end{cases}$$

Let $\phi_0, \ldots, \phi_{n+1}$ be piecewise-linear B-splines on a grid on $[a,b]$, the basis function for the Galerkin method.
The first and last of the c_i are found from collocation:

$$1 = y(0) = \sum_{i=0}^{n+1} c_i\phi_i(0) = c_0\phi_0(0) = c_0$$

$$3 = y(1) = \sum_{i=0}^{n+1} c_i\phi_i(1) = c_{n+1}\phi_{n+1}(1) = c_{n+1}$$

Finite Elements and the Galerkin Method

For $i = 1, \ldots, n$, use the FE equations

$$\int_a^b \underbrace{f(t, y, y'))}_{=4y} \phi_i(t) = \underbrace{\phi_i(b)}_{=0} y'(b) - \underbrace{\phi_i(a)}_{=0} y'(a) - \int_a^b y'(t)\phi_i'(t)dt$$

and substitute the functional form of $y(t) = \sum_{i=0}^{n+1} c_i\phi_i(t)$

$$\begin{aligned} 0 &= \int_0^1 \left(4\phi_i(t) \sum_{j=0}^{n+1} c_j\phi_j(t) + \sum_{j=0}^{n+1} c_j\phi_j'(t)\phi_i'(t) \right) dt \\ &= \sum_{j=0}^{n+1} c_j \left(\int_0^1 4\phi_i(t)\phi_j(t)dt + \int_0^1 \phi_j'(t)\phi_i'(t)dt \right) \end{aligned}$$

Finite Elements and the Galerkin Method

Assuming that the grid is evenly-spaced with step size h, we have:

$$\int_a^b \phi_i(t)\phi_{i+1}(t)dt = \int_0^h \frac{t}{h}\left(1-\frac{t}{h}\right)dt = \int_0^h \left(\frac{t}{h}-\frac{t^2}{h^2}\right)dt = \frac{t^2}{2h} - \frac{t^3}{3h^2}\bigg|_0^h = \frac{h}{6}$$

$$\int_a^b (\phi_i(t))^2 dt = 2\int_0^h \left(\frac{t}{h}\right)^2 dt = \frac{2}{3}h$$

$$\int_a^b \phi_i'(t)\phi_{i+1}'(t)dt = \int_0^h \frac{1}{h}\left(-\frac{1}{h}\right)^2 dt = -\frac{1}{h}$$

$$\int_a^b (\phi_i'(t))^2 dt = 2\int_0^h \left(\frac{1}{h}\right)^2 dt = \frac{2}{h},$$

for all $i = 1, \ldots, n$.

Finite Elements and the Galerkin Method

For $i = 1, \dots, n$

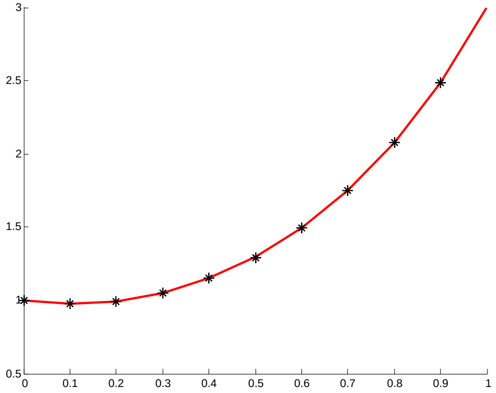
$$\begin{aligned}
\left(\frac{2}{3}h - \frac{1}{h}\right) c_0 + \left(\frac{8}{3}h + \frac{2}{h}\right) c_1 + \left(\frac{2}{3}h - \frac{1}{h}\right) c_2 &= 0 \\
\left(\frac{2}{3}h - \frac{1}{h}\right) c_1 + \left(\frac{8}{3}h + \frac{2}{h}\right) c_2 + \left(\frac{2}{3}h - \frac{1}{h}\right) c_3 &= 0 \\
&\vdots \\
\left(\frac{2}{3}h - \frac{1}{h}\right) c_{n-1} + \left(\frac{8}{3}h + \frac{2}{h}\right) c_n + \left(\frac{2}{3}h - \frac{1}{h}\right) c_{n+1} &= 0
\end{aligned}$$

where $c_0 = y_a = 1$ and $c_{n+1} = y_b = 3$,
so the matrix form of the equations is symmetric diagonal

$$\begin{bmatrix}
\alpha & \beta & 0 & \cdots & 0 \\
\beta & \alpha & \ddots & \ddots & \vdots \\
0 & \beta & \ddots & \beta & 0 \\
\vdots & \ddots & \ddots & \alpha & \beta \\
0 & \cdots & 0 & \beta & \alpha
\end{bmatrix}
\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{n-1} \\ c_n \end{bmatrix}
=
\begin{bmatrix} -y_1\beta \\ 0 \\ \vdots \\ 0 \\ -y_b\beta \end{bmatrix}$$

where $\alpha = \frac{8}{3}h + \frac{2}{h}, \beta = \frac{2}{3}h - \frac{1}{h}$.

Finite Elements and the Galerkin Method. Example



PDEs

A second order PDE of two independent variables has the form

$$Au_{xx} + Bu_{xy} + Cu_{yy} = f(x, y, u, u_x, u_y) \tag{15.1}$$

for some constants A, B, C and a function $f(x, y, u, p, q)$.
Denote the discriminant

$$\Delta = B^2 - 4AC.$$

Then the equation is

1. **elliptic** if $\Delta < 0$
2. **parabolic** if $\Delta = 0$
3. **hyperbolic** if $\Delta > 0$

It is useful to classify the PDEs, as different types have different solution properties, and these properties have major impacts on the choice and performance of numerical methods.
For parabolic and hyperbolic equations one of the two independent variables can be interpreted as time. For this reason we will use x and t for the independent variables in parabolic and hyperbolic equations.

Examples of PDEs of two independent variables

- **The Poisson equation** (elliptic)
$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x,y), \qquad (x,y) \in \Omega \subset \mathbb{R}^2$$
(When $f(x,y) = 0$, it is called the *Laplace equation*)
- **The heat equation** (parabolic)
$$\frac{\partial u}{\partial t} = a\frac{\partial^2 u}{\partial x^2} + f(t,x), \qquad x \in (0,L), t > 0$$
- **The wave equation** (hyperbolic)
$$\frac{\partial^2 u}{\partial t^2} = a\frac{\partial^2 u}{\partial x^2} + f(t,x) \qquad x \in (0,L), t > 0$$

The heat equation

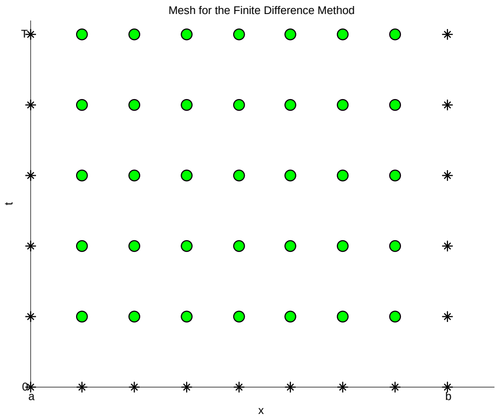
The heat equation (parabolic) in one spatial variable,
with Dirichlet Boundary Conditions and Initial Condition

$$\begin{cases} \dfrac{\partial}{\partial t}u(x,t) = c\dfrac{\partial^2}{\partial x^2}u(x,t) & \text{for all } a \leq x \leq b, t \geq 0 \\ u(a,t) = \ell(t), u(b,t) = r(t), & \text{for all } t \geq 0 \\ u(x,0) = f(x) & \text{for all } a \leq x \leq b \end{cases} \tag{15.2}$$

where $c > 0$ is the **diffusion** coefficient, x, t independent variables.

Describes the diffusion of substance from regions of higher concentration to regions pof lower concentration.

Forward difference method



The finite difference method for the heat equation

We denote the exact solution by

$$u(x_i, t_j)$$

and its approximation at (x_i, t_j) by

$$w_{ij}$$

Let M and N be the total number of steps in the x and t directions, and let

$$h = \frac{b-a}{M} \qquad k = \frac{T}{N}$$

be the step sizes in the x and t directions.

The finite difference method for the heat equation

We use the centered-difference formula for u_{xx}

$$\frac{\partial^2 u}{\partial x^2}(x,t) \approx \frac{1}{h^2}\left(u(x+h,t) - 2u(x,t) + u(x-h,t)\right)$$

with the error

$$h^2 \frac{u_{xxxx}(c_1,t)}{12}$$

and the forward difference formula for u_t

$$\frac{\partial u}{\partial t}(x,t) \approx \frac{1}{k}\left(u(x,t+k) - u(x,t)\right)$$

with the error

$$k\frac{u_{tt}(x,c_2)}{2}$$

where $x-h < c_1 < x+h$ and $t < c_2 < t+h$.

The finite difference method for the heat equation

Substituting into the heat equation at the point (x_i, t_j) yields

$$\frac{c}{h^2}\left(w_{i+1,j} - 2w_{ij} + w_{i-1,j}\right) \approx \frac{1}{k}\left(w_{i,j+1} - w_{ij}\right) \tag{15.3}$$

with the local truncation error given by

$$\mathcal{O}(k) + \mathcal{O}(h^2)$$

which is how the solution advances, step by step, in the t variable:

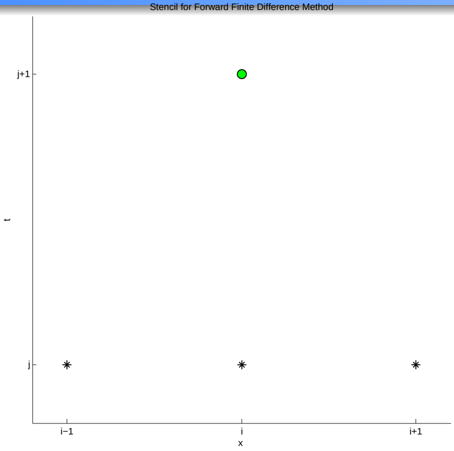
$$\begin{aligned} w_{i,j+1} =& w_{ij} + \frac{ck}{h^2}(w_{i+1,j} - 2w_{ij} + w_{i-1,j}) \\ =& \sigma w_{i+1,j} + (1-2\sigma)w_{ij} + \sigma w_{i-1,j} \end{aligned}$$

where

$$\sigma = c\frac{k}{h^2}$$

That is, if $u(x,t)$ is known for $0 \le x \le 1$ and $0 \le t \le t_0$, one can compute the solution at time $t = t_0 + k$.

Stencil for Forward Finite Difference Method



The Forward Finite Difference Method is **explicit**, since there is a way to determine new values (in the sense of time) directly from previously known values.

Forward Difference - explicit heat equation

We can compute $w_{i,j+1}$ at time t_{j+1} by computing the matrix multiplication

$$w_{j+1} = Aw_j + s_j$$

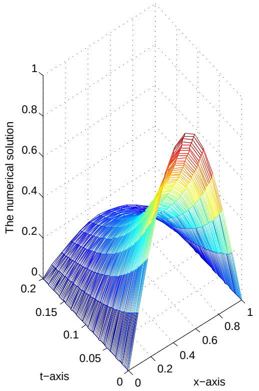
or

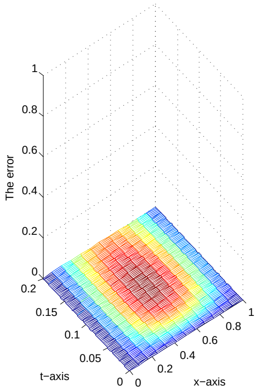
$$\begin{bmatrix} w_{1,j+1} \\ \\ \vdots \\ \\ w_{m,j+1} \end{bmatrix} = \begin{bmatrix} 1-2\sigma & \sigma & 0 & \cdots & 0 \\ \sigma & 1-2\sigma & \sigma & \cdots & \vdots \\ 0 & \sigma & 1-2\sigma & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \sigma \\ 0 & \cdots & 0 & \sigma & 1-2\sigma \end{bmatrix} \begin{bmatrix} w_{1j} \\ \\ \vdots \\ \\ w_{m,j+1} \end{bmatrix} + \begin{bmatrix} w_{0,j} \\ 0 \\ \vdots \\ 0 \\ w_{m+1,j} \end{bmatrix}$$

Forward Difference - explicit heat equation

$c = 1$, initial condition $f(x) = \sin(\pi x), M_x = 12, N_t = 96, T = 0.2, \ell = r = 0$

$$\sigma = c\frac{k}{h^2} = \frac{.2}{96}\frac{12^2}{1} = .3 < \frac{1}{2}$$

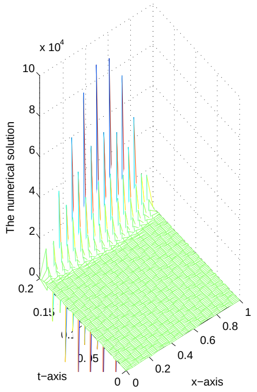


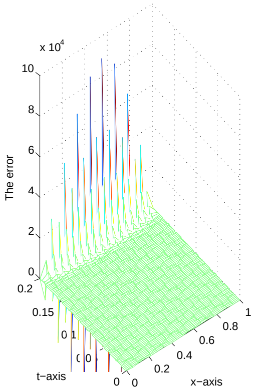


Forward Difference - explicit heat equation

$c = 1$, initial condition $f(x) = \sin(\pi x), M_x = 18, N_t = 96, T = 0.2, \ell = r = 0$

$$\sigma = c\frac{k}{h^2} = \frac{.2}{96}\frac{18^2}{1} = .675 > \frac{1}{2}$$





Stability of Forward Difference Method

The main advantage of an explicit method is that there is no need to solve linear systems, as the approximate solution at time level $t = t_{j+1}$ can be computed directly from that at previous steps.

However, a *useful numerical method must be stable*, i.e., error propagation through time advancing should be controllable.

Suppose that the computed solution at time level $t = t_j$ is $\tilde{w}_{ij} = w_{ij} + \varepsilon_{ij}$, $|\varepsilon_{ij}| \leq \varepsilon_j$, $1 \leq i \leq M + 1$.

$$\tilde{w}_{i,j+1} = w_{i,j+1} + \varepsilon_{i,j+1} = \sigma \tilde{w}_{i+1,j} + (1 - 2\sigma)\tilde{w}_{i,j} + \sigma \tilde{w}_{i-1,j}$$

hence

$$\varepsilon_{i,j+1} = \sigma \varepsilon_{i+1,j} + (1 - 2\sigma)\varepsilon_{i,j} + \sigma \varepsilon_{i-1,j}, \quad 2 \leq i \leq M_x$$

Note that if

$$\boldsymbol{\sigma = c\frac{k_t}{h_x^2} \leq \frac{1}{2}}$$

then

$$|\varepsilon_{i,j+1}| \leq \sigma|\varepsilon_{i+1,j}| + (1 - 2\sigma)|\varepsilon_{i,j}| + \sigma|\varepsilon_{i-1,j}| \leq |\varepsilon_j|, \quad 2 \leq i \leq M_x$$

i.e., the error is not amplified in time - **conditionally stable**.

$$\max_{1 \leq i \leq M_x+1, 1 \leq j \leq N_t+1} |u(x_i, t_j) - w_{ij}| = \mathcal{O}(k_t + h_x^2)$$

Stability of Forward Difference Method

Von Neumann stability analysis measures the error magnification, or amplification. For a stable method, step sizes must be chosen so that the amplification factor is no larger than 1.

Let u_j be the exact solution that satisfies

$$u_{j+1} = Au_j + s_j$$

and let the computed approximation, satisfying

$$w_{j+1} = Aw_j + s_j$$

The difference $e_j = w_j - u_j$ satisfies

$$e_j = w_j - u_j = Aw_{j-1} + s_{j-1} - (Au_{j-1} + s_{j-1}) = A(w_{j-1} - u_{j-1}) = Ae_{j-1}$$

We saw that the method is stable iff the spectral radius

$$\rho(A) < 1.$$

This requirement puts limits on the step sizes h_x and k_t of the forward finite difference method.

Stability of Forward Difference Method

Notice that

$$A = -\sigma T + (1-\sigma)I, \quad \text{where} \quad T = \begin{bmatrix} 1 & -1 & 0 & \cdots & 0 \\ -1 & 1 & -1 & \cdots & \vdots \\ 0 & -1 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & -1 \\ 0 & \cdots & 0 & -1 & 1 \end{bmatrix}$$

Theorem

The eigenvectors of T are

$$v_j = \left[\sin\frac{j\pi}{m+1}, \sin\frac{2\pi j}{m+1}, \ldots, \sin\frac{m\pi j}{m+1}\right]$$

and the eigenvalues

$$\lambda_j = 1 - 2\cos\frac{\pi j}{m+1}$$

Stability of Forward Difference Method

Hence the eigenvalues of $A = -\sigma T + (1-\sigma)I$ are

$$-\sigma\left(1 - 2\cos\frac{\pi j}{m+1}\right) + 1 - \sigma = 2\sigma\underbrace{\left(\cos\frac{\pi j}{m+1} - 1\right)}_{\in(-2,0)} + 1, \quad j = 1, \dots, m.$$

So the eigenvalues range from

$$-4\sigma + 1 \quad \text{to} \quad 1$$

hence the method is stable iff $\rho(A) < 1 \Leftrightarrow \sigma < \frac{1}{2}$.

Theorem

*The Forward Difference Method applied to the heat equation with $c > 0$ is stable iff $2ck_y < h^2$ (**conditionally stable**).*

Backward Difference Method

We have seen that for the explicit method k_t and h_x must be chosen so the stability condition

$$\frac{k_t}{h_x^2} \leq \frac{1}{2c}$$

is satisfied. This imposes a restriction on the relative size of h_x and k_t. Suppose that the numerical solution corresponding to the current values of k_t and h_x is not accurate enough. It is natural to try smaller values for k_t and h_x. Now assume we use $\tilde{h}_x = \frac{h_x}{2}$.

To maintain the same ratio σ, we need to choose $\tilde{k}_t$ such that

$$c\frac{\tilde{k}_t}{\tilde{h}_x^2} = c\frac{k_t}{h_x^2}$$

i.e.

$$\tilde{k}_t = \frac{1}{4}k_t$$

So, whenever h_x is halved, k_t must be quartered - this may lead to use of prohibitively small time stepsize.

Backward Difference Method

Approximating

$$\frac{\partial u}{\partial t}(x_i, t_j) \approx \frac{u_{ij} - u_{i,j-1}}{k_t}$$

we obtain the **backward Euler method**

$$\frac{u_{ij} - u_{i,j-1}}{k_t} \approx c\frac{u_{i+1,j} - 2u_{ij} + u_{i-1,j}}{h_x^2}, \qquad 2 \le i \le M_x$$

which is always stable.

Again denote by

$$\sigma = c\frac{k_t}{h_x^2}.$$

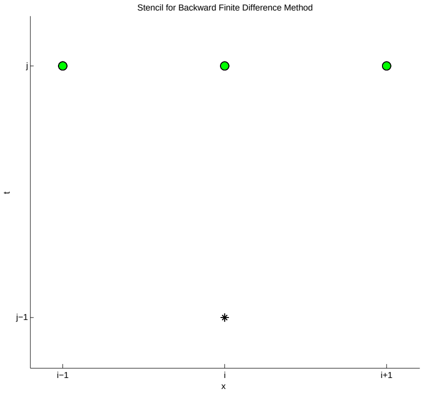
Then the numerical scheme is

$$u_{i,1} = u_0(x_i), \qquad 1 \le i \le M_x + 1$$

and for $j \ge 2$

$$\begin{cases} -\sigma u_{i-1,j} + (1+2\sigma)u_{ij} - \sigma u_{i+1,j} = u_{i,j-1}, & 2 \le i \le M_x \\ u_{1,j} = \ell(t_j), \quad u_{M_x+1,j} = r(t_j) \end{cases}$$

Stencil for the Backward Difference Method



For a given approximate solution at $t = t_{j-1}$,
we need to solve a linear system to find the approximate solution at the next time level $t = t_j$ - **implicit method**.

Backward Difference Method

The Backward Difference Method can be viewed as

$$Au(:,j) = u(:,j-1) + b_j$$

where

$$A = \begin{bmatrix} 1+2\sigma & -\sigma & 0 & \cdots & 0 \\ -\sigma & 1+2\sigma & -\sigma & \cdots & \vdots \\ 0 & -\sigma & 1+2\sigma & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & -\sigma \\ 0 & \cdots & 0 & -\sigma & 1+2\sigma \end{bmatrix}$$

or

$$w_j = A^{-1}w_{j-1} + b_j.$$

As in the Von Neumann stability analysis of the Forward Difference Method, the relevant quantities are the eigenvalues of A^{-1}.

Stability of the Backward Difference Method

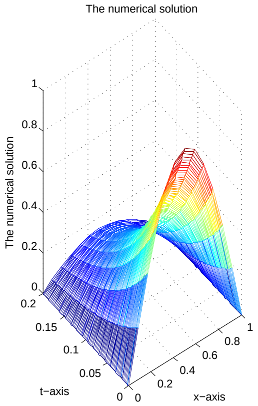
Since
$$A = \sigma T + (1+\sigma)I$$
the eigenvalues of A are
$$\sigma\left(1 - 2\cos\frac{\pi j}{m+1}\right) + 1 + \sigma = 1 + 2\sigma - 2\sigma\cos\frac{\pi j}{m+1}$$
and the eigenvalues of A^{-1} are the reciprocals. To ensure
$$\rho(A^{-1}) < 1$$
we need
$$|1 + 2\underbrace{\sigma}_{>0}\underbrace{(1-\cos x)}_{>0}| > 1$$
which is true for all σ.

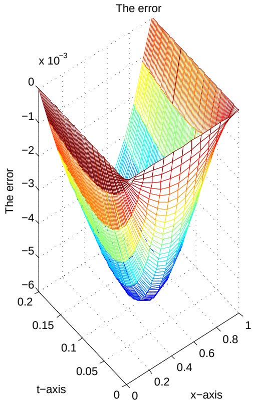
Therefore the **implicit method is stable for all σ, i.e., all choices of h_x, k_t.**

If the solution is sufficiently smooth, the error bound
$$\max_{1\le i\le M_x+1, 1\le j\le N_t+1} |u(x_i,t_j) - w_{ij}| = \mathcal{O}(k_t + h_x^2)$$

Backward Difference - implicit heat equation

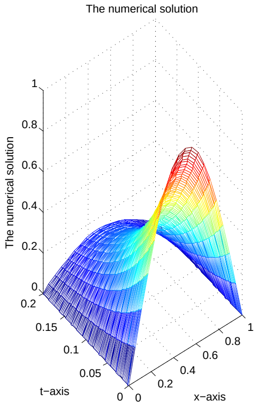
$$c = 1, \text{ initial condition } f(x) = \sin(\pi x), M_x = 12, N_t = 96, T = 0.2, \ell = r = 0$$
$$\sigma = c\frac{k}{h^2} = \frac{.2}{96}\frac{12^2}{1} = .3 < \frac{1}{2}$$

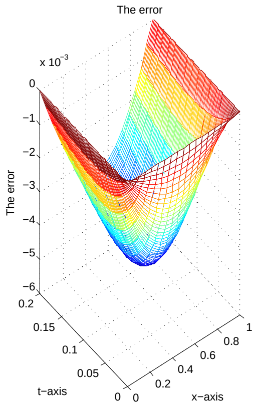




Backward Difference - implicit heat equation

$$c = 1, \text{ initial condition } f(x) = \sin(\pi x), M_x = 18, N_t = 96, T = 0.2, \ell = r = 0$$
$$\sigma = c\frac{k}{h^2} = \frac{.2}{96}\frac{18^2}{1} = .675 > \frac{1}{2}$$





Crank-Nicolson Method

So far, our methods for the heat equation consist of

1. explicit method that is sometimes stable (conditionally stable) - forward difference
2. implicit method that is always stable (unconditionally stable) - backward difference

Both have errors of size

$$\mathcal{O}(k_t + h_x^2)$$

when stable.
The time step size k_t needs to be fairly small to obtain good accuracy.

The Crank-Nicolson Method

- is a combination of the explicit method and implicit methods,
- is unconditionally stable, and
- has error

$$\mathcal{O}(k_t^2 + h_x^2)$$

Crank-Nicolson Method

Approximating u_t with the **backward difference formula**

$$\frac{\partial u}{\partial t}(x_i, t_j) \approx \frac{u_{ij} - u_{i,j-1}}{k_t}$$

and u_{xx} with the **central difference formula** at $(x, t - \frac{1}{2}k)$

$$\frac{\partial^2 u}{\partial x^2}(x_i, t_j - \frac{1}{2}k) = \frac{1}{h_x^2}\left(u_{i+1,j-\frac{1}{2}} - 2u_{i,j-\frac{1}{2}} + u_{i-1,j-\frac{1}{2}}\right)$$

we obtain the **Crank-Nicolson method**

$$\frac{u_{ij} - u_{i,j-1}}{k_t} \approx c\frac{1}{h_x^2}\left(u_{i+1,j-\frac{1}{2}} - 2u_{i,j-\frac{1}{2}} + u_{i-1,j-\frac{1}{2}}\right).$$

Crank-Nicolson Method

$$\frac{u_{ij} - u_{i,j-1}}{k_t} \approx c\frac{1}{h_x^2}\left(u_{i+1,j-\frac{1}{2}} - 2u_{i,j-\frac{1}{2}} + u_{i-1,j-\frac{1}{2}}\right).$$

Since the u values are known only on grid points, replacing $u(x,t-\frac{1}{2}k)$ with the average of the values at adjacent points

$$u(x,t-\frac{1}{2}k) \approx \frac{1}{2}(u(x,t) + u(x,t-k))$$

we obtain

$$\frac{u_{ij} - u_{i,j-1}}{k_t} \approx c\frac{1}{2h_x^2}\left(u_{i+1,j} - 2u_{i,j} + u_{i-1,j} + u_{i+1,j-1} - 2u_{i,j-1} + u_{i-1,j-1}\right)$$

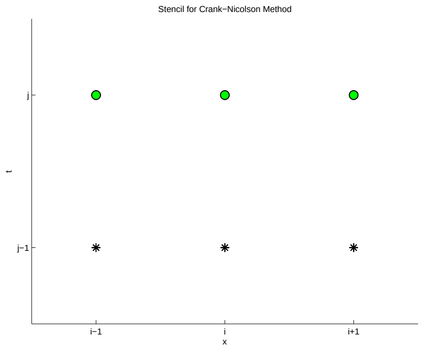
or

$$-\sigma u_{i-1,j} + (2+2\sigma)\boldsymbol{u_{ij}} - \sigma u_{i+1,j} = \sigma u_{i-1,j-1} + (2-2\sigma)u_{i,j-1} + \sigma u_{i+1,j-1}$$

where again

$$\sigma = c\frac{k_t}{h_x^2}.$$

Stencil for the Crank-Nicolson Method



At each time step, the green open circles are the unknowns and the black stars are known from the previous step.

Crank-Nicolson Method

$$-\sigma u_{i-1,j} + (2+2\sigma)\boldsymbol{u_{ij}} - \sigma u_{i+1,j} = \sigma u_{i-1,j-1} + (2-2\sigma)u_{i,j-1} + \sigma u_{i+1,j-1}$$

Set $u_j = [u_{1j}, \dots, u_{mj}]^T$. In matrix form, the **Crank-Nicolson Method** is

$$Au_j = Bu_{j-1} + \sigma(s_{j-1} + s_j)$$

where

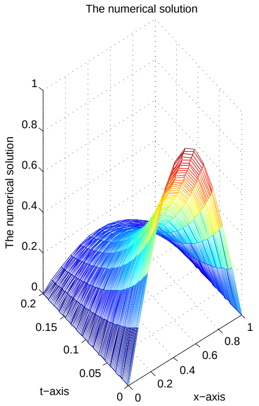
$$A = \begin{bmatrix} 2+2\sigma & -\sigma & 0 & \cdots & 0 \\ -\sigma & 2+2\sigma & -\sigma & \ddots & \vdots \\ 0 & -\sigma & 2+2\sigma & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -\sigma \\ 0 & \cdots & 0 & -\sigma & 2+2\sigma \end{bmatrix}, \quad B = \begin{bmatrix} 2-2\sigma & \sigma & 0 & \cdots & 0 \\ \sigma & 2-2\sigma & \sigma & \ddots & \vdots \\ 0 & \sigma & 2-2\sigma & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \sigma \\ 0 & \cdots & 0 & \sigma & 2-2\sigma \end{bmatrix}$$

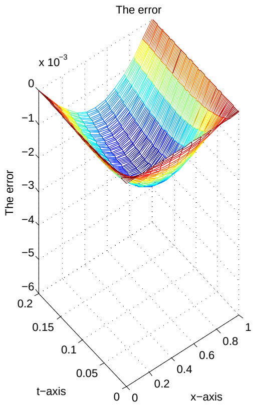
and $s_j = [u_{0j}, 0, \dots, 0, u_{mj}]^T$.

Crank-Nicolson Method - heat equation

$c = 1$, initial condition $f(x) = \sin(\pi x), M_x = 12, N_t = 96, T = 0.2, \ell = r = 0$

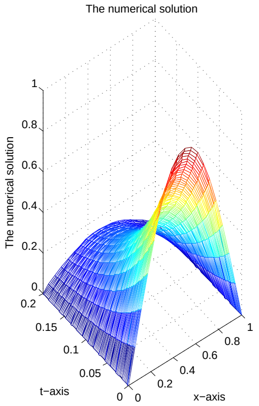
$$\sigma = c\frac{k}{h^2} = \frac{.2}{96}\frac{12^2}{1} = .3 < \frac{1}{2}$$

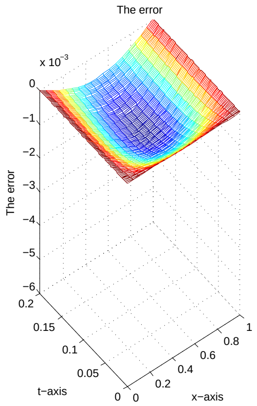




Crank-Nicolson Method - heat equation

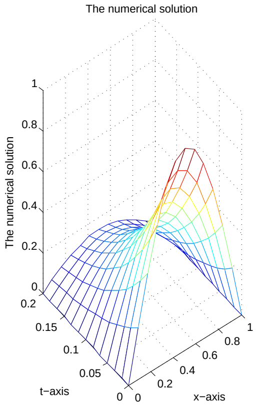
$$c = 1, \text{ initial condition } f(x) = \sin(\pi x), M_x = 18, N_t = 96, T = 0.2, \ell = r = 0$$
$$\sigma = c\frac{k}{h^2} = \frac{.2}{96}\frac{18^2}{1} = .675 > \frac{1}{2}$$

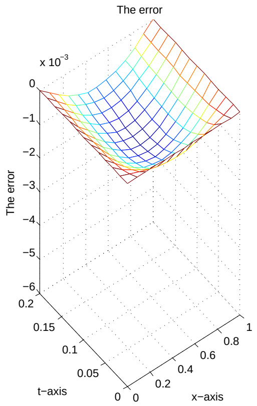




Crank-Nicolson Method - heat equation

$$c = 1, \text{ initial condition } f(x) = \sin(\pi x), M_x = 12, N_t = 12, T = 0.2, \ell = r = 0$$
$$\sigma = c\frac{k}{h^2} = \frac{.2}{12}\frac{12^2}{1} = 2.4 > \frac{1}{2}$$





Stability of the Crank-Nicolson Method

$$Au_j = Bu_{j-1} + \sigma(s_{j-1} + s_j)$$

where

$$A = \begin{bmatrix} 2+2\sigma & -\sigma & 0 & \cdots & 0 \\ -\sigma & 2+2\sigma & -\sigma & \ddots & \vdots \\ 0 & -\sigma & 2+2\sigma & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -\sigma \\ 0 & \cdots & 0 & -\sigma & 2+2\sigma \end{bmatrix}, \quad B = \begin{bmatrix} 2-2\sigma & \sigma & 0 & \cdots & 0 \\ \sigma & 2-2\sigma & \sigma & \ddots & \vdots \\ 0 & \sigma & 2-2\sigma & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \sigma \\ 0 & \cdots & 0 & \sigma & 2-2\sigma \end{bmatrix}$$

We must find the spectral radius of the matrix $A^{-1}B$.
Note that

$$A = \sigma T + (2+\sigma)I, \qquad B = -\sigma T + (2-\sigma)I.$$

Let v_j, λ_j be an (eigenvector,eigenvalue) of T, then

$$\begin{aligned} A^{-1}Bv_j &= (\sigma T + (2+\sigma)I)^{-1}(-\sigma\lambda_j v_j + (2-\sigma)v_j) \\ &= \frac{1}{\sigma\lambda_j + 2 + \sigma}(-\sigma\lambda_j + 2 - \sigma)v_j \end{aligned}$$

Crank-Nicolson Method - **Unconditionally Stable**

The eigenvalues of $A^{-1}B$ are

$$\frac{-\sigma\lambda_j + 2 - \sigma}{\sigma\lambda_j + 2 + \sigma} = \frac{4 - (\sigma(\lambda_j + 1) + 2)}{\sigma(\lambda_j + 1) + 2} = \underbrace{\frac{4}{L} - 1}_{\in(-1,1)}$$

where

$$L = \sigma(\underbrace{\lambda_j}_{>-1} + 1) + 2 > 2$$

because

$$\lambda_j = 1 - 2\cos\frac{\pi j}{m+1}$$

Theorem

The Crank-Nicolson Method applied to the heat equation $u_t = cu_{xx}$ with $c > 0$ is stable for any step sizes $h_x, k_t > 0$.

The wave equation with speed c

$$u_{tt} - c^2 u_{xx} = 0, \qquad \text{on } (a,b), t \geq 0$$

It is hyperbolic: $\Delta = B^2 - 4AC = 4c^2 > 0$.
Boundary conditions:

$$\begin{aligned} u(a,t) &= \ell(t), \\ u(b,t) &= r(t) \end{aligned} \qquad \text{for all } t \geq 0$$

Initial Conditions:

$$\begin{aligned} u(x,0) &= f(x), \\ u_t(x,0) &= g(x) \end{aligned} \qquad \text{for all } x \in [a,b]$$

Describes the of a wave propagation along the x-direction.
It models:

- magnetic waves in sun's atmosphere
- oscillations of a violin string

u is amplitude represents

- physical displacement of the string (for the violin)
- local air pressure (for the a sound wave traveling in the air)

The wave equation - analytic solution

Consider the problem

$$u_{tt} - c^2 u_{xx} = 0, \qquad \text{on } (0,1), t \geq 0$$

$$\begin{aligned} u(0,t) &= 0, \\ u(1,t) &= 0 \end{aligned} \qquad \text{for all } t \geq 0$$

$$\begin{aligned} u(x,0) &= f(x), \\ u_t(x,0) &= 0 \end{aligned} \qquad \text{for all } x \in [0,1]$$

The solution is

$$u(x,t) = \frac{1}{2}\left(f(x+ct) + f(x-ct)\right)$$

provided that $f \in C^2(\mathbb{R})$ and is extended to the whole $\mathbb{R}$ by

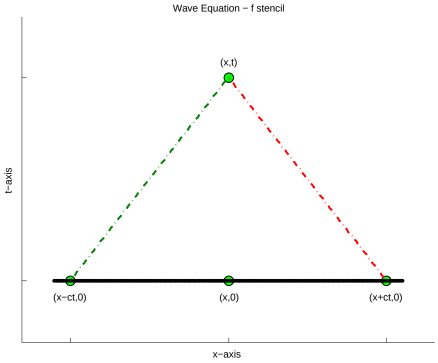
$$f(-x) = -f(x) \quad \text{and} \quad f(x+2) = f(x)$$

Indeed

$$u_x = \frac{1}{2}\left(f'(x+ct) + f'(x-ct)\right), \quad u_t = \frac{c}{2}\left(f'(x+ct) - f'(x-ct)\right)$$

$$u_{xx} = \frac{1}{2}\left(f''(x+ct) + f''(x-ct)\right), \quad u_{tt} = \frac{c^2}{2}\left(f''(x+ct) + f''(x-ct)\right)$$

The wave equation



To compute $u(x, t)$ one needs to know f at only two point on the x-axis, $x + ct$ and $x - ct$.

The wave equation

To discretize the wave equation use centered-difference:

$$\frac{u_{i,j+1} - 2u_{ij} + u_{i,j-1}}{k^2} - c^2\frac{u_{i-1,j} - 2u_{ij} + u_{i+1,j}}{h^2} = 0$$

and with

$$\sigma = c\frac{k}{h}$$

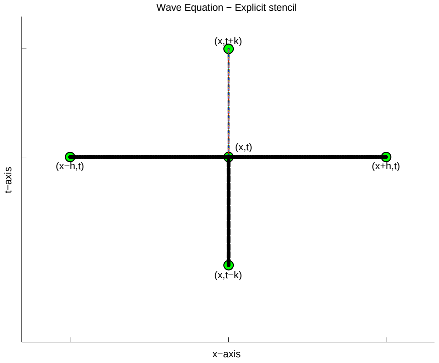
we can solve for the solution at the next step

$$\boxed{u_{i,j+1} = (2-2\sigma^2)u_{ij} + \sigma^2 u_{i-1,j} + \sigma^2 u_{i+1,j} - u_{i,j-1}}$$

This formula cannot be used for the first time step:

$$u_{i,1} = (2-2\sigma^2)u_{i0} + \sigma^2 u_{i-1,0} + \sigma^2 u_{i+1,0} - u_{i,\mathbf{-1}}$$

The wave equation



$$u_{i,j+1} = (2-2\sigma^2)u_{ij} + \sigma^2 u_{i-1,j} + \sigma^2 u_{i+1,j} - u_{i,j-1}$$

The wave equation

We approximate the initial data at the first time step with three-point centered difference formula

$$g(x_i) = u_t(x_i, t_0) \approx \frac{u_{i1} - u_{i,-1}}{2k}$$

i.e.,

$$u_{i,-1} \approx u_{i1} - 2kg(x_i)$$

and the difference formula at $j = 0$ gives

$$u_{i1} = (2 - 2\sigma^2)u_{i0} + \sigma^2 u_{i-1,0} + \sigma^2 u_{i+1,0} - u_{i1} + 2kg(x_i)$$

or

$$u_{i1} = (1 - \sigma^2)u_{i0} + kg(x_i) + \frac{\sigma^2}{2}(u_{i-1,0} + u_{i+1,0})$$

The wave equation

$$A = \begin{bmatrix} 2-2\sigma^2 & \sigma^2 & 0 & \cdots & 0 \\ \sigma^2 & 2-2\sigma^2 & \sigma^2 & \ddots & \vdots \\ 0 & \sigma^2 & 2-2\sigma^2 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \sigma^2 \\ 0 & \cdots & 0 & \sigma^2 & 2-2\sigma^2 \end{bmatrix}$$

The initial equation

$$\begin{bmatrix} w_{11} \\ \vdots \\ w_{m1} \end{bmatrix} = \frac{1}{2}A \begin{bmatrix} w_{10} \equiv f(x_1) \\ \vdots \\ w_{m0} \equiv f(x_m) \end{bmatrix} + k \begin{bmatrix} g(x_1) \\ \vdots \\ g(x_m) \end{bmatrix} + \frac{1}{2}\sigma^2 \begin{bmatrix} w_{00} \equiv \ell(t_0) \\ 0 \\ \vdots \\ 0 \\ w_{m+1,0} \equiv r(t_0) \end{bmatrix}$$

and the subsequent steps

$$\begin{bmatrix} w_{1,j+1} \\ \vdots \\ w_{m,j+1} \end{bmatrix} = A \begin{bmatrix} w_{1j} \\ \vdots \\ w_{mj} \end{bmatrix} - \begin{bmatrix} w_{1,j-1} \\ \vdots \\ w_{m,j-1} \end{bmatrix} + \sigma^2 \begin{bmatrix} w_{0j} \equiv \ell(tj0) \\ 0 \\ \vdots \\ 0 \\ w_{m+1,j} \equiv r(t_j) \end{bmatrix}$$

Stability of the wave equation - CFL condition

The matrix form allows to analyze the stability characteristics of the **explicit** Finite Difference Method applied to the wave equation.

Theorem

The Finite Difference Method applied to the wave equation with wave speed $c > 0$ *is stable of* $\sigma = c\frac{k}{h} \leq 1$.

Proof. In matrix form

$$u_{j+1} = Au_j - u_{j-1} + \sigma^2 s_j$$

where s_j holds the side conditions. Since u_{j+1} depends on both u_j and u_{j-1}, to study error magnification we rewrite it as

$$\begin{bmatrix} w_{j+1} \\ w_j \end{bmatrix} = \begin{bmatrix} A & -I \\ I & 0 \end{bmatrix} \begin{bmatrix} w_j \\ w_{j-1} \end{bmatrix} + \sigma^2 \begin{bmatrix} s_j \\ 0 \end{bmatrix}$$

to view the method as one-step method recursion.

Error will not be magnified as long as the eigenvalues of $\mathcal{A} = \begin{bmatrix} A & -I \\ I & 0 \end{bmatrix}$ are bounded by 1 in absolute value.

Stability of the wave equation - CFL condition

Proof (continued)
Let $\lambda \neq 0$, $(y, z)^T$ be an eigenvalue/eigenvector of $\mathcal{A}$, i.e.,

$$\begin{aligned} \lambda y &= Ay - z \\ \lambda z &= y \end{aligned} \quad \Rightarrow Ay = \underbrace{\left(\frac{1}{\lambda} + \lambda\right)}_{=\mu} y.$$

Hence μ is an eigenvalue of A,

$$\mu \in (2 - 4\sigma^2, 2).$$

Now

$$\sigma \leq 1 \Rightarrow -2 \leq \mu \leq 2$$

To finish, show that for a complex number λ, $\frac{1}{\lambda} + \lambda$ is real and has magnitude at most 2 implies $|\lambda| = 1$. ■

Elliptic Equations

Laplacian

Let $u(x,y)$ be a twice-differentiable function, define the **Laplacian** of u as

$$\Delta u = u_{xx} + u_{yy}$$

The **Poisson Equation**:

$$\Delta u(x,y) = f(x,y).$$

The Poisson equation with $f(x,y) = 0$ is called **Laplace equation**.
A solution to Laplace equation is called a **harmonic** function.
The **Helmholtz equation**:

$$\Delta u(x,y) + cu(x,y) = f(x,y).$$

Mesh for the Finite Difference solver of the Poisson Equation with Dirichlet BC

Use the centered difference formula

$$f''(x) = \frac{1}{h^2}\left(f(x-h) - 2f(x) + f(x+h)\right)$$

to obtain the five-point formula for the Laplacian

$$\Delta u = \frac{1}{h^2}\Big(u(x-h,y) + u(x+h,y) + u(x,y-h) + u(x,y+h) - 4u(x,y)\Big)$$

which is of order $\mathcal{O}(h^2)$, because the local error is

$$-\frac{h^2}{12}(u_{xxxx}(\xi,y) + u_{yyyy}(x,\eta))$$

The elliptic equation

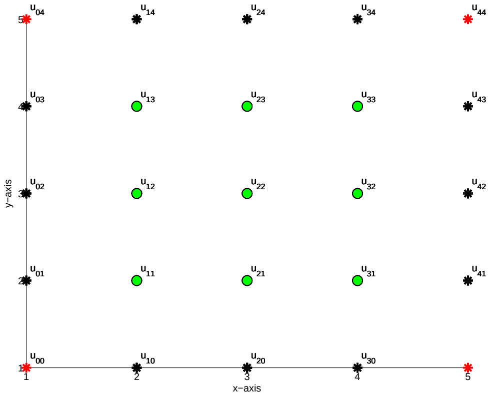
$$\Delta u + fu = g$$

on grid points writes

$$-u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} + (4 - h^2 f_{ij})u_{ij} = -h^2 g_{ij}$$

Helmholtz Equation: Five Point Star

$\Omega = [0,1] \times [0,1], h = \frac{1}{4}$, nine interior grid points.



There are nine equations

$$\begin{cases}
-u_{21}-u_{01}-u_{12}-u_{10}+(4-h^2f_{11})u_{11}=-h^2g_{11}\\
-u_{31}-u_{11}-u_{22}-u_{20}+(4-h^2f_{21})u_{21}=-h^2g_{21}\\
-u_{41}-u_{21}-u_{32}-u_{30}+(4-h^2f_{31})u_{31}=-h^2g_{31}\\
-u_{22}-u_{02}-u_{13}-u_{11}+(4-h^2f_{12})u_{12}=-h^2g_{12}\\
-u_{32}-u_{12}-u_{23}-u_{21}+(4-h^2f_{22})u_{22}=-h^2g_{22}\\
-u_{42}-u_{22}-u_{33}-u_{31}+(4-h^2f_{32})u_{32}=-h^2g_{32}\\
-u_{23}-u_{03}-u_{14}-u_{12}+(4-h^2f_{13})u_{13}=-h^2g_{13}\\
-u_{33}-u_{13}-u_{24}-u_{22}+(4-h^2f_{23})u_{23}=-h^2g_{23}\\
-u_{43}-u_{23}-u_{34}-u_{32}+(4-h^2f_{33})u_{33}=-h^2g_{33}
\end{cases}$$

45 coefficients - 12 known (at the boundary terms) = 33 nonzero entries, out of 81 in a 9×9 system.

$Ax = b$

The coefficient matrix

$$A=\begin{bmatrix} 4-h^2f_{11} & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4-h^2f_{21} & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4-h^2f_{31} & -1 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & -1 & 4-h^2f_{12} & -1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 4-h^2f_{22} & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 & 4-h^2f_{32} & -1 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & -1 & 4-h^2f_{13} & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4-h^2f_{23} & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4-h^2f_{33} \end{bmatrix}$$

The unknowns (in natural ordering)

$$u = [u_{11}, u_{21}, u_{u31}, u_{12}, u_{22}, u_{32}, u_{13}, u_{23}, u_{33}]^T$$

The RHS

$$b = \begin{bmatrix} -h^2 g_{11} + u_{10} + u_{01} \\ -h^2 g_{21} + u_{20} \\ -h^2 g_{31} + u_{30} + u_{41} \\ -h^2 g_{12} + u_{02} \\ -h^2 g_{22} \\ -h^2 g_{32} + u_{42} \\ -h^2 g_{13} + u_{14} + u_{03} \\ -h^2 g_{23} + u_{24} \\ -h^2 g_{33} + u_{34} + u_{43} \end{bmatrix}$$

Random Walks

Random walk

A **random walk** $W_t : \mathbb{R}_+ \to \mathbb{R}$ is defined on the real line, by

$$\begin{array}{l} W_0 = 0 \\ \text{moving a step of size } s_i \text{ at each integer time } i \end{array}$$

where s_i are independent and identically distributed random variables.

Here we assume each $s_i = \pm 1$ with equal probability $\frac{1}{2}$.

Discrete Brownian motion

Discrete Brownian motion is the random walk given by the sequance of accumulated steps

$$W_t = W_0 + s_1 + s_2 + \cdots + s_t,$$

for $t = 0, 1, 2, \ldots$.

Random Walks

Since a random walk is a probabilistic device, let's recall some concepts from probability. $t \mapsto W_t$

- For each t, the value of W_t is a random variable
- A sequence of random variables $\{W_0, W_1, W_2, \ldots\}$ is a **stochastic process**

- The expected value of a single step s_i of the random walk W_t is

$$E(s_i) = (0.5)(1) + (0.5) \times (-1) = 0$$

- and the variance of s_i is

$$E[(s_i - 0)^2] = (0.5)(1)^2 + (0.5) \times (-1)^2 = 1$$

Random Walks

- The expected value after an integer t steps is

$$E(W_t) = E(s_1 + \cdots + s_t) = \underbrace{E(s_1)}_{=0} + \cdots + \underbrace{E(s_t)}_{=0} = 0$$

- and the variance is

$$V(W_t) = V(s_1 + \cdots + s_t) = \underbrace{V(s_1)}_{=1} + \cdots + \underbrace{V(s_t)}_{=1} = t$$

because the variance is additive over independent variables.

The mean and variance are statistical quantities that summarize information about probability distribution.

Random Walks

The fact that

- the mean of W_t: $E(W_t) = 0$, and
- the variance $V(W_t) = t$

indicates that if we compute n different realizations of the random variable W_t, then the

$$\textbf{sample mean} = E_{\text{sample}}(W_t) = \frac{W_t^1 + \cdots + W_t^n}{n} \approx 0$$

and

$$\textbf{sample variance} = V_{\text{sample}}(W_t) = \frac{(W_t^1 - E_s)^2 + \cdots + (W_t^n - E_s)^2}{n-1} \approx t$$

should approximate ≈ 0 and $\approx t$.

The **sample standard deviation** $= \sqrt{\text{sample variance}}$, also called **standard error**.

Random Walks

Many interesting applications of random walks are based on escape times, also called first passage times.

The escape time

Let a, b be positive integers, and consider the first time the random walk starting at 0 reaches the boundary of the interval $[-b, a]$.

It can be shown that the probability that the escape time happens at a (rather than $-b$) is exactly $\frac{b}{a+b}$.
The expected length of the escape time from $[-b, a]$ is known to be ab.

Continuous Brownian motion

So for the standard random walk W_t at time t,
the expected value and variance

$$E(W_t) = 0 \quad \text{and} \quad V(W_t) = t.$$

Imagine now that double the number of steps are taken per unit time. If a step is taken every $\frac{1}{2}$ time unit, the expected value of the random walk at time t is still 0

$$E(W_t) = 0$$

but the variance is changed to

$$V(W_t) = V(s_1 + \cdots + s_{2t}) = V(s_1) + \cdots + V(s_{2t}) = 2t,$$

since $2t$ steps have been taken.
To keep the variance fixed while we increase the number of steps, we need to reduce the (vertical) size of each step.

If we increase the number of steps by a factor k, we need to change the step height by a factor of $\frac{1}{\sqrt{k}}$. This is because multiplication of a random variable by a constant changes the variance by the square of the constant.

Continuous Brownian motion

W_t^k

W_t^k = the random walk that takes a step s_i^k

- of horizontal length $\frac{1}{k}$ and with
- step height $\pm\frac{1}{\sqrt{k}}$ with equal probability

Then the expected value at time t is still

$$E(W_t^k) = \sum_{i=1}^{kt} E(s_i^k) = \sum_{i=1}^{kt} 0 = 0,$$

and the variance is

$$V(W_t^k) = \sum_{i=1}^{kt} V(s_i) = \sum_{i=1}^{kt} \left[\left(\frac{1}{\sqrt{k}}\right)^2 (.5) + \left(-\frac{1}{\sqrt{k}}\right)^2 (.5) \right] = kt\frac{1}{k} = t.$$

The limit W_t^∞ of this progression as $k \to \infty$ yields **continuous Brownian motion**.

Now $t \in \mathbb{R}$ and $B_t \equiv W_t^\infty$ is a random variable for each $t \geq 0$.

Continuous Brownian motion

The continuous Brownian motion B_t has three important properties:

1. for each t, the random variable B_t is normally distributed with mean 0 and variance t (Central Limit Theorem).
2. for each $t_1 < t_2$, the normal random variable $B_{t_2} - B_{t_1}$ is independent of the random variable B_{t_1}, and in fact independednt of all B_s, $0 \leq s \leq t_1$.
3. Brownian motion B_t can be represented by continuous paths.

Computer simulation of Brownian motion

Establish grid of steps:

$$0 = t_0 \leq t_1 \leq \cdots \leq t_n$$

on the t-axis and start with $B_0 = 0$.
Property 2 says that

$$B_{t_1} - B_{t_0}$$

is a normal random variable with mean =0 and variance $= t_1$.
Therefore a realization of the random variable B_{t_1} can be made by choosing from the normal distribution

$$N(0, t_1) = \sqrt{t_1 - t_0} N(0, 1).$$

We choose B_{t_2} similarly:

$$B_{t_2} - B_{t_1} = N(0, t_2 - t_1) = \sqrt{t_2 - t_1} N(0, 1),$$

i.e., choose a standard normal random number, multiply by $\sqrt{t_2 - t_1}$ and add it to B_{t_1}.

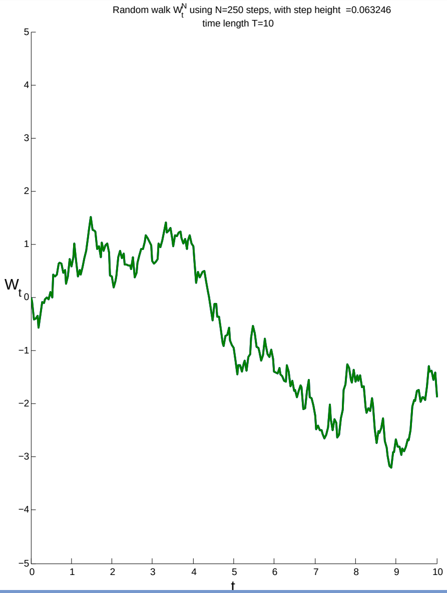
In general the increment of Brownian motion is the square root of the time step multiplied by a standard normal random number.

Computer simulation of Brownian motion

In MATLAB use the normal random number generator `randn`, with stepsize $\Delta t = \frac{1}{25}$.

```
k = 250;
sqdelt=sqrt(1/25);
b=0;
for i = 1 : k
  b = b+ sqdelt* randn;
end
```

Computer simulation of Brownian motion



SDEs

Solutions to ordinary differential equations / stochastic differential equations are functions / stochastic processes

Definition

A set of random variables $\{x_t\}_{t\geq 0}$ is called a **continuous-time stochastic process**.

$$\omega \mapsto x(\omega, \boldsymbol{t}) \in \boldsymbol{C^1[0,T]}.$$

Each instance, or **realization** of the stochastic process is a choice of the random variable x_t for each t, and is therefore a function of t.

Continuous-time stochastic processes

1. Brownian motion is a stochastic process
2. Any (deterministic) function f can also be trivially considered as a stochastic process, with variance $V(f(t)) = 0$.
3. The solution of the SDE i.v.p.

$$\begin{cases} dy = rdt + \sigma dB_t \\ y(0) = 0 \end{cases}$$

   with constants r and σ, is the stochastic process $y(t) = rt + \sigma B_t$.

Notice that the SDE is given in differential form, unlike the DE.
That is because many interesting stochastic proceses, like Brownian motion, are continuous, but not differentiable.

Itô integral

Therefore, the meaning of the SDE

$$dy = f(t,y)dt + g(t,y)dB_t$$

is, by definition, the integral equation

$$y(t) = y(0) + \int_0^t f(s,y)ds + \underbrace{\int_0^t g(s,y)dB_s}_{\text{Itô integral}}.$$

Itô integral

Let $a = t_0 < t_1 < \cdots < t_{n-1} < t_n = b$ be a grid of points on $[a, b]$.

- The Riemann integrall is defined as a limit

$$\int_a^b f(t)dt = \lim_{\delta \to 0} f(t_i')\Delta t_i,$$

  where $\Delta t_i = t_i - t_{i-1}$ and $t_{i-1} \leq t_i' \leq t_i$.

- Similarly, the **Itô integral** is the limit

$$\int_a^b f(t)dB_t = \lim_{\delta \to 0} f(t_{i-1})\Delta B_i,$$

  where $\boldsymbol{\Delta B_i = B_{t_i} - B_{t_{i-1}}}$, a step in the Brownian motion across the interval.

While t_i' in the Riemann integral may be chosen at any point in (t_{i-1}, t_i), the corresponding point for the Itô integral is required to be the left endpoint of that interval.

White noise

Because f and B_t are random variables, so is the Ito integral $I = \int_a^b f(t)dB_t$. The **differential** dI is a notational convention; thus

$$I = \int_a^b f \, dB_t$$

is equivalent by definition to

$$dI = f dB_t.$$

The differential $\boldsymbol{dB_t}$ of Brownian motion B_t is called **white noise**.

We can approximate a solution to an SDE in a way similar to the Euler-Method.
The *Euler-Maruyama Method* works by discretizing the time axis, just as Euler does.
We define the approximate solution path at a grid of points

$$a = t_0 < t_1 < \cdots < t_{n-1} < t_n = b$$

and will assign approximate y values

$$w_0 < w_1 < w_2 < \cdots < w_n$$

at the respective t points.

$$\begin{cases} dy(t) = f(t,y)dt + g(t,y)dB_t \\ y(a) = y_a \end{cases}$$

Euler-Maruyama Method

Given the SDE i.v.p.

$$\begin{cases} dy(t) = f(t,y)dt + g(t,y)dB_t \\ y(a) = y_a \end{cases}$$

we compute approximately:

Euler-Maruyama Method

```
$w_0 = y_0$
**for** $i = 0, 1, 2, \dots$
    $w_{i+1} = w_i + f(t_i, w_i)(\Delta t_i) + g(t_i, w_i)(\Delta B_i)$
**end**
```

where $\begin{array}{l} \Delta t_i = t_{i+1} - t_i \\ \Delta B_i = B_{t_{i+1}} - B_{t_i} \end{array}$.

The crucial part is how to model the Brownian motion ΔB_i:

$$\Delta B_i = z_i \sqrt{\Delta t_i},$$

where z_i is chosen from $N(0,1)$ (generate it with `randn`).

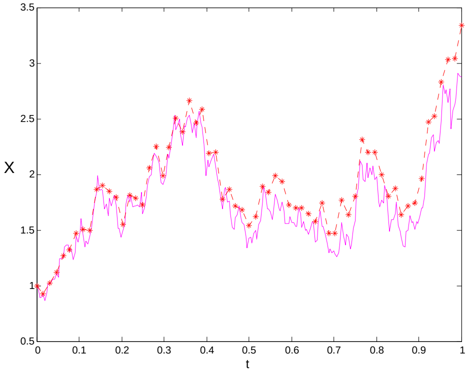
Each set of $\{w_0, \dots, w_n\}$ we produce is an approximate realization of the solution stochastic process $y(t)$, which depends on the random numbers z_i that were chosen.

Since B_t is a stochastic process, each realization will be different, and so the approximations.

Euler-Maruyama method to a linear SDE

Solve the stochastic differential equation
$dX = \lambda * X dt + \mu * X dW,$
$X(0) = X_0,$
where
$\lambda = 2,$
$\mu = 1, \beta = 1,$
$X_0 = 1.$
The discretized Brownian path over [0,1] has $dt = 2^{-8}$.

Euler-Maruyama method to a linear SDE



Black-Scholes

The references:
% Desmond Higham,
% An Algorithmic Introduction to Numerical Simulation of
% Stochastic Differential Equations,
% SIAM Review,
% Volume 43, Number 3, September 2001, pages 525-546.

% Desmond Higham,
% Black-Scholes for Scientific Computing Students,
% Computing in Science and Engineering,
% November/December 2004, Volume 6, Number 6, pages 72-79.

The codes:
http://www.sc.fsu.edu/˜burkardt/m_src/sde/sde.html
http://www.sc.fsu.edu/˜burkardt/m_src/black_scholes/black_scholes.html

Definition

An SDE has **order** m if the expected value of the error is of mth order in the step size, i.e.,
for any time T,

$$E\{|y(T) - w(T)|\} = \mathcal{O}((\Delta t)^m)$$

as $\Delta t \to 0$.

It is a surprise that,

- unlike the ODE case where the Euler Method has order $m = 1$,
- the Euler-Maruyama Method for SDEs has order $m = \frac{1}{2}$.

Milstein Method

To build an order 1 method for SDEs, another term in the "stochastic Taylor series" must be added to the method.

$$\begin{cases} dy(t) = f(t,y)dt + g(t,y)dB_t \\ y(a) = y_a \end{cases}$$

Milstein Method

```
$w_0 = y_0$
**for** $i = 0, 1, 2, \ldots$
    $w_{i+1} = w_i + f(t_i, w_i)(\Delta t_i) + g(t_i, w_i)(\Delta B_i)$
        $+\frac{1}{2} g(t_i, w_i) \frac{\partial g}{\partial y}(t_i, w_i)((\Delta B_i)^2 - \Delta t_i)$
**end**
```

where $\begin{array}{l} \Delta t_i = t_{i+1} - t_i \\ \Delta B_i = B_{t_{i+1}} - B_{t_i} \end{array}$.

Milstein Method

The Milstein Method has order 1.
Note that the Milstein Method is identical to the Euler-Maruyama Method if there is no y term in the diffusion part $g(y, t)$ of the equation.
In case there is, Milstein will converge to the correct stochastic solution procce3ss more quickly than Euler-Maruyama as the step size $h \to 0$.