Nicodemos 1.1

Manual

August 2012

Ayres Freitas[1]

[1] afreitas@pitt.edu

Disclaimer

NICODEMOS is a *Mathematica* program for the numerical evaluation of loop integrals, using subtraction terms for IR singularities. The *Mathematica* code performs the subtraction and additional algebraic steps, such as necessary variable transformations. It produces a *Fortran* executable for the numerical evaluation. The version 1.1 can handle one-loop and types of two-loop integrals as described in Ref. [1] (see reference for details).

NICODEMOS may be used freely for non-commercial purposes, with proper acknowledgement of the author and citation of the original publication [1]. If you include it as part of a different program, proper acknowledgement must be given in the documentation of the program.

The NICODEMOS package includes the package *Format* 2.0 by Mark Sofroniou (Copyright 1992-2003, Mark Sofroniou).

NICODEMOS requires *Mathematica 6.0* or above, installed on a Unix/Linux compatible operating system with a Fortran 77 compiler and Perl interpreter.

It also needs the *Cuba* numerical integration library by Thomas Hahn [2]. NICODEMOS 1.1 has been tested with CUBA version 1.4, which can be obtained from `http://www.feynarts.de/cuba/Cuba-1.4.tar.gz`.

The author does not promise that this software works. NICODEMOS 1.1 has been tested with *Mathematica 6.0* and *8.0* on a platform with *Scientific Linux SL releases 5.1 and 6.3* and *GNU Fortran 3.4.6*. Usage on other systems may lead to compatibility issues which in most cases should be straightforward to resolve by an experienced user. Bug reports are welcome (but their timely resolution is not guaranteed).

1 Setup

NICODEMOS can be downloaded from

```
http://www.pitt.edu/~afreitas/
```

It comes in a compressed zip file, which can be unpacked with

```
unzip nicodemos10b.zip
```

Unpacking creates the directory `nicodemos10b/`. The subdirectory `main/` contains the NICODEMOS code, while several examples are collected in the subdirectory `examples/`.

NICODEMOS requires *Mathematica 6.0* or above and the *Cuba* library. NICODEMOS 1.1 has been tested with version 1.4 which can be downloaded from `http://www.feynarts.de/cuba/Cuba-1.4.tar.gz`. Installation instructions for *Cuba* are provided in the downloaded tar archive.

Before running NICODEMOS, the entries in `main/paths` must be adjusted as needed, for example

```
$Compiler = "f77"          command for Fortran compiler
$Cuba = "Cuba-1.4/"        directory where libcuba.a is located
```

2 Overview

In this section, the usage of NICODEMOS is demonstrated with two examples. First, let us consider the one-loop QED vertex correction to $\gamma^* \to q\bar{q}$.

After starting *Mathematica* the program is loaded with

```
In[1]:= << main/oneloop.m
```

The expression for the one-loop integrand is supplied in two parts, the denominator and the numerator:

```
In[2]:= gaqqden = {k1.k1, (k1-p1).(k1-p1), (k1+p2).(k1+p2)};

In[3]:= gaqqnum = (2*($d-4)*k1.k1*p1.p2 - 8*(p1.k1 + p1.p2)*
        (p1.p2 - p2.k1))/(27*Pi*p1.p2);
```

Furthermore one needs to define the external momenta, the external parameters (such as masses and kinematic quantities), and any relevant relations between them:

```
In[4]:= momext = {p1,p2};

In[5]:= pars = {s};

In[6]:= rels = {p1.p1 -> 0, p2.p2 -> 0, p1.p2 -> s/2};
```

All relevant input parameters must be listed in `pars`, *i. e.* after application of `rels` the final expression must not contain any undefined symbols.

As the next step, all IR singularities are subtracted:

```
In[7]:= gaqqnum = SubSoft[gaqqden, gaqqnum, k1, k1];

In[8]:= gaqqnum = SubColl[gaqqden, gaqqnum, k1, k1->p1];

In[9]:= gaqqnum = SubColl[gaqqden, gaqqnum, k1, k1->-p2];
```

`SubSoft` performs the subtraction of soft divergencies, where the third and fourth arguments specify the loop momentum and the soft momentum, respectively (which in this example are identical). Similarly, `SubColl` is used for collinear subtraction, where the third argument again denotes the loop momentum, while the fourth argument defines the collinear limit for this particular singularity. Soft singularities must be subtracted before collinear ones. Note that the collinear subtraction is well-defined only for physical, gauge-invariant amplitudes.

The symbolic part of the calculation is completed by the following two commands:

```
In[10]:= gaqqres = OneLoop[gaqqden, gaqqnum, k1];

In[11]:= WriteCode[gaqqres, Deform->True, WorkingDirectory->"num1"];
```

The `OneLoop` function performs the Feynman parameterization and tensor reduction and all other algebraic steps that finally produce a Feynman parameter integral. This result is turned into *Fortran* code by the routine `WriteCode`. The option `Deform->True` indicates that the integration contour is deformed into the complex plane to avoid threshold singularities. The second options `WorkingDirectory` allows the user to specify the directory where the numerical code is written.

After `WriteCode` has been executed, this directory will contain an executable `run1` which performs the numerical integration. For example, a typical call to this program is

```
num1 > num1/run1 5 0.3
```

The command line arguments correspond to the input parameters in `pars`, in the same order as specified there. In this example, this is the single parameter `s`, which is set to the value 5. If contour deformation is used, the last command line parameter is the deformation parameter λ (which here is set to 0.5). See Ref. [1] for the defintion of λ. `run1` produces the following output:

```
(0.135636514,0.016214035) (0.000171570896,0.000149737364)
(0.00516076584,-0.148148148) (0.,0.)
(-0.0471570202,0.)  (0.,0.)
```

Here the first line corresponds to the finite part of the result, given as real and imaginary part, followed by the numerical error for each. The second and third line are the $1/\varepsilon$ and $1/\varepsilon^2$ poles, respectively.

Now let us turn to a two-loop example. The treatment of two-loop integrals in NICODEMOS is very similar to the one-loop case, except that now one uses the command `TwoLoop` instead of `OneLoop`, and there are two symbols for the loop momenta.

As an example, let us consider an electroweak vertex diagram contributing to $Z \to b\bar{b}$ in the limit $m_b \to 0$, with UV but without IR divergencies:

In[1]:=
```
<< main/twoloop.m
```

In[2]:=
```
verden = {k1.k1 - MTs, (k1+p1+p2).(k1+p1+p2) - MTs, k2.k2 - MWs,
      (k2-p2).(k2-p2) - MTs, (k2-p1-p2).(k2-p1-p2) - MWs,
      (k1+k2).(k1+k2)};
```

In[3]:=
```
vernum = (MZs^2*(2*(MZs-4*MWs)*($d-4)^2*(k1.k2)^2 +
```
... [abridged]

In[4]:=
```
pars = {MZs,MWs,MTs};
```

In[5]:=
```
muUV2 = MZs;
```

In[6]:=
```
momext = {p1,p2};
```

In[7]:=
```
rels = {p1.p1 -> 0, p2.p2 -> 0, p1.p2 -> MZs/2};
```

In[8]:= `res = TwoLoop[verden, vernum, k1, k2];`

In[9]:= `WriteCode[res, WorkingDirectory->"num2"];`

Here `k1` and `k2` are the loop momenta.

Note that for a UV-divergent two-loop integral the user must define the global symbol `muUV2` before running `TwoLoop` (besides the global symbols `momext, pars,` and `rels` mentioned on page 2). It specifies the mass scale for the two-loop subtraction terms. The final numerical result is independent of the value of `muUV2`, but the numerical integration converges faster if it is not too small.

The numerical code for a two-loop example is operated in just the same way as at the one-loop level, *i. e.* the user invokes the executable `run1` with the necessary numerical input parameters as command line arguments. For more complex two-loop examples, NICODEMOS may create several subdirectories for the numerical program, but this does not affect the usage.

3 List of Functions and Options

Before any function of the package can be called, the user must define the integrand numerator and denominator, as well as the relevant input parameters, external momenta, and relations between them, as described in the previous section. After that, if the integral contains soft and/or collinear divergencies, they should be subtracted with the following functions. In the current version, only subtractions of singularities in a one-loop integral or in one subloop of a two-loop integral are supported.

`SubSoft[`d, n, k, p_{soft}`]`	subtracts a single-loop soft singularity that occurs for $p_{soft} \to 0$ from an integrand specified by the denominator d and numerator n, where k is the loop momentum

`SubSoft` returns the subtracted numerator.

`SubColl[`d, n, k, k `->` p_{coll}`]`	subtracts a single-loop collinear singularity that occurs for $k \to p_{coll}$ from an integrand specified by the denominator d and numerator n, where k is the loop momentum

Similar to `SubSoft`, `SubColl` returns the subtracted numerator.

For one-loop integrals, the function `OneLoop` performs the remaining algebraic manipulations and has several options to control its operations:

`OneLoop[`d, n, k`]`	transforms the integral specified by denominator d, numerator n, and loop momentum k into a Feynman parameter integral

option	*default*	
`SimpLevel`	`3`	specifies how the level of simplification that `OneLoop` tries to perform; values from 0 to 3 are possible
`NestedExpressions`	`False`	when set to `True`, breaks down long expressions by introducing placeholder symbols for subexpressions

Larger values of `SimpLevel` lead to shorter *Fortran* code, but require more time and memory in *Mathematica*. For large expressions it is advisable to reduce the value of `SimpLevel`.

`NestedExpressions` offers another way to keep long expressions under control and avoid memory overflows in *Mathematica*. If set to true it will break down a large expression by introducing symbols for subexpressions. This typically leads to *Fortran* code that compiles faster but may be slightly slower in the numerical evaluation.

For two-loop integrals, one uses the function `TwoLoop` instead of `OneLoop`:

`TwoLoop[`d, n, k_1, k_2`]`	transforms the integral specified by denominator d, numerator n, and loop momenta k_1 and k_2 into a Feynman integral

option	*default*	
`SimpLevel`	`3`	specifies how the level of simplification that `OneLoop` tries to perform; values from 0 to 5 are possible, but levels 4 and 5 should be used with caution since they may lead to very long evaluation times and/or memory overflow
`StatusInfo`	`0`	Specifies if and how much progress and status information should be printed out during the algebraic computation; `0` = none, `1` = basic information, `2` = progress and timing info

As mentioned above, for large expressions it is advisable to set `SimpLevel` to smaller values. `TwoLoop` always introduces nested subexpressions to break down large expressions, so there is no corresponding user option.

When evaluating a two-loop integral with UV singularities, the user must define the global symbol `muUV2` before running `TwoLoop`. Any symbol listed in `pars` may be assigned to `muUV2`. The final numerical result does not depend on the value of `muUV2`, but the numerical integration can be made more efficient by choosing a parameter that is not far from the largest mass or momentum scale in the problem.

The *Fortran* code is finally generated by `WriteCode`:

`WriteCode[`e`]`	produces *Fortran* code from the expression e (e is the output of `OneLoop`)

option	*default*	
`Deform`	`False`	whether to perform contour deformation to avoid threshold singularities
`EpsRange`	`{0,-Infinity}`	which terms of the ε-expansion to include in the *Fortran* code
`SimpFunc`	`Simplify`	function used to simplify the ε-expanded result
`WorkingDirectory`	`"."`	the directory where the *Fortran* code should be written

Note that only non-positive values in `EpsRange` are meaningful. This option can be used to produce numbers for, say, only the finite part of the integral.

The option `SimpFunc` can be adjusted to prevent *Mathematica* from consuming too much time and/or memory. The operation of `WriteCode` can be sped up by setting `SimpFunc` to some function that performs simpler operations than `Simplify`, As a last resort one could simply set `SimpFunc -> Identity`. However, this may lead to larger *Fortran* code.

Options for the numerical code The directory where the executable `run1` is created will contain a file `runpar.sys`, which options for to control the numerical integration. By default it contains the following values:

runpar.sys:	
`v`	integration routine (`v` = VEGAS, `c` = CUHRE)
`200000`	maximal number of integration points
`1e-16`	desired relative accuracy
`1e-16`	desired absolute precision

The numerical integration will stop when it reaches either of these three: *(i)* maximal number of integration points, *(ii)* relative accuracy goal (given as relative amount of the final answer), *(iii)* absolute precision goal (given as absolute error on the final answer).

References

[1] A. Freitas, JHEP **1207**, 132 (2012) [arXiv:1205.3515 [hep-ph]].

[2] T. Hahn, Comput. Phys. Commun. **168**, 78 (2005) [arXiv:hep-ph/0404043].