

NICODEMOS 1.0

Manual

May 2012 Ayres Freitas¹

¹afreitas@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 1.0 can only handle one-loop integrals (an extension to two loops is foreseen for future versions). For details see Ref. [1].

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.

It also needs the *Cuba* numerical integration library by Thomas Hahn [2]. NICODEMOS 1.0 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 1.0 has been tested with *Mathematica 6.0* and *8.0* on a platform with *Scientific Linux SL release 5.1* 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 nicodemus10b.zip
```

Unpacking creates the directory `nicodemus10b/`. 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 1.0 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

<code>\$Compiler = "f77"</code>	<i>command for Fortran compiler</i>
<code>\$Cuba = "Cuba-1.4/"</code>	<i>directory where libcuba.a is located</i>

2 Overview

In this section, the usage of NICODEMOS is demonstrated with an example, the one-loop QED vertex correction to $\gamma^* \rightarrow q\bar{q}$.

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 definition 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.

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 functions

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

SubSoft returns the subtracted numerator.

SubColl $[d, n, k, k \rightarrow p_{coll}]$	subtracts a collinear singularity that occurs for $k \rightarrow p_{coll}$ 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.

The function **OneLoop** performs the remaining algebraic manipulations and has several options to control its operations:

<code>OneLoop[d, n, k]</code>	transforms the integral specified by denominator d , numerator n , and loop momentum k into a Feynman parameter integral
-------------------------------	--

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

Larger values of `SimpLevel` lead to shorter *Fortran* code, but require more computing 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.

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

<code>WriteCode[e]</code>	produces <i>Fortran</i> code from the expression e (e is the output of <code>OneLoop</code>)
---------------------------	--

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

Similar to the options for `OneLoop`, 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:

<i>runpar.sys:</i>	
<code>v</code>	integration routine (<code>v</code> = VEGAS, <code>c</code> = CUHRE)
<code>200000</code>	maximal number of integration points
<code>1e-16</code>	desired relative accuracy
<code>1e-16</code>	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].