ICalc v.1.3 user guide

Zoltán Szőr

November 17, 2015

1 Introduction

The ICalc is a collection of various programs and scripts, a tool for the user to calculate bunch of integrals numerically in a *"press a button – wait – get results & be happy"* way. The specific aim is to automate every step of the computation of integrals using SecDec when the number of the integrations are high enough to be tedious to do it by "hand". The user provides the inputs and specify some simple parameters and after that just hit the button and everything is going by itself. Unfortunately not all the possible errors are handled, the program is not completely user safe so please be careful!

The program contains C++ programs, shell scripts, python scripts written by the author of this manual and also the program called SecDec-2.1 written by *Sophia Borowka, Jonathon Carter, Gudrun Heinrich*. Some small modifications has been made to avoid some issues during calculations.

In the next section I will provide how the program works through a simple example.

2 How it works?

2.1 Installation and structure

First we need to tar out the .tar file containing ICalc which I assume has been already done since you are reading this manual.

The ICalc directory contains the following directories and files:

- SecDec-2.1.tar.gz – source for SecDec
- **secdecpatch** – contains some scripts and modifications to patch the original SecDec

- **cprograms** – home of `collectres` and `makemath` (further description see at section 5)
- **pythonscripts** – two python scripts to set relative accuracy
- **shscripts** – shellscripts to run the `collectres` and `makemath` programs multiple times
- **Test** – our test example for learning
- `install.py` – python script to install `ICalc`
- `run.py` – python script to run `ICalc`

To install type `python install.py` in your terminal. What happened? First the script compiles the `C++` programs. As nowadays everyone has a multicore machine we would like to use this advantage during our calculations. During installation of `ICalc` the script counts the number of processor cores, creates and installs as many `SecDec` folders as many cores the current machine has. This could take a few minutes. After a successful installation you are ready to use the program, so we move on to the next step.

2.2 What input need to be provided?

Now go to the **Test** directory and see what is inside.

SecDec inputs

We have a directory called **SecDecsource** to store the necessary input files for `SecDec` (I assume you know how `SecDec` works, if not please check it first). It contains a subdirectory named `I2C1` which well be our *type* of integral to be calculated. The integral folder contains four `.input` files and the corresponding `.m` files. These are the concrete integrands of type I2C1 what we need[1]. Note that the input files are numbered although the specific integrand names are different. In general the structure should be the following: the folder **SecDecsource** should contain directories of the integral types which contain the specific input files using the numbered notation. Thou the parameter setting could be done real-time during calculations, it is easier to do it right once for all as handle them with scripts.

The folder **Values** contains the input files for calculations using multiple values. For SecDec calculations the name pattern must be *Integral*

[1]This means that the I2C1 actually depends on some parameters. The different input files contain different sets of these parameters.

*type*_values_*numbering*.input. These has not to be provided if you use *secdec_ single* as the **runtype** option (see further below).

ICalc run files

ICalc run files with extension .run stand for set some options for the program before running it. If we take a look into the file sdtest.run we see the following four setting variables:

- **runname** – this will be the codename of our run, the results will be found after calculations in the directory named after it.
- **cpumod** – though ICalc can run with multiple threads, you may not want to use all cores of your CPU. To set this four possible setting is available: *low* use only one thread; *moderate* take half of the cores; *full* will run on every core; or you can set with a number explicitly (it has to be less or equal as the number of available cores).
- **runtype** – *secdec* for *every* SecDec calculations from the decomposition with also multinumerics; *secdec_ single* runs only the ./launch command; *secdec_ multi* for multinumerics only (decomposition must be done in advance!); and *mb* is for Mellin-Barnes computations which only works for *I2Si* integrals at the moment.
- **collect** – collect results and errors into a single *Mathematica* .m file or not.

Config files

In **Configfiles** you can found .cfg files for each integrand separately. They provide information about the integrals for the program makemath which converts table-like results into *Mathematica* format. If you look into some of them the structure and the meaning of variables should be quite clear. Maybe three of them worth a little bit further explanation:

- **outputfile** – this is the name of file where all the data would be found in Mathematica format. If you use makemath in the frame of ICalc please keep it *Iall.m* in every case.
- **variables** – here you should list the variables of your integral what can be found in the table formatted .res files. The structure of the table is *variables*, *results & errors* for each pole. The name of the variables can be chosed freely and they have to be separated with commas (and no spaces).

- **outvars** – here you should list the name of variables (the names should match the ones given at *variables*) which you want to written into your Mathematica function definiton.

According to this description you may able to write your own config files for your own integrals.

2.3 Run `ICalc` – example

Now we have enough information to run `ICalc`. Before doing it, I suggest to set the parameter **cpumod** to *moderate* in `sdtest.run`. After that go to the **ICalc** folder and type `python run.py Test sdtest.run` into your terminal. The general syntax of the this command can be guess easily from it: `python run.py` *'name of the project directory' 'name of the run file'*.

After hitting enter the program starts running. First it scans the **SecDec-source** directory for input files and distribute them equally between the available threads (their number depend on your *cpumod* parameter). It writes shellscripts automaticly to run every task sequentially at each thread and run the scripts parallel at the same time. The output of SecDec's are written into log files named `secdec_sdtest.log` at the SecDec directories. When all the integrations are done the results will be copied to **Test/SDtest/results**. If the *collect* parameter in `.run` file is *yes*, then the `ICalc` runs the `collectres` and the `makemath` programs. The first one collect all the results belonging to one integral into single `_all.res` files in table format. The `makemath` then convert these tables into a Mathematica file where all the integrals are defined as functions at every point where they have been computed. In our example this will be `Test_SDtest_all.m` for results and the same with an `err` preword for errors both at **Test/SDtest**. At the end of the running the program also prints the date of starting, date of ending and the consumed time in minutes.

2.4 Run `ICalc` – generally

After finishing our short tutorial now you are able to run `ICalc` completely generally with *secdec* runtype. The steps are the same as before, just maybe you will have more integrals and/or run the program in different *cpumod*. Remember `ICalc` is not completely user safe so I ask you to follow the all instructions and keep the input formats.

Notes: 1.: in case of an already existing `.m` and `_all.res` files the new results will append to the end; 2.: `makemath` streams data with 10 digits of

fixed `C++` stream precision! If you need more you can set it easily in the sourcecode.

3 Changes in version 1.2 (since v1.0)

In **install.py**

- Number of installing SecDecs can be set by the user during the installation process up to the number of available processor cores.

In **run.py**

- The number of threads can be set explicitly with an integer as **cpumod** option.
- New options have been added to **runmode**: *secdec_single* for running only the `launch` command, and *secdec_multi* for only the `mymultinumerics.pl` command. *secdec* alone stands for both!
- Error emerging during the scanning over the content of **SecDecsource** directory is fixed.

SecDec patch

- Multikernel decomposition is switched off. In the file `SDroutine.m` `LaunchKernels[]` (which runs as default as many kernels as many cores are available) is replaced to `LaunchKernels[1]` to avoid memory and kernel overloading which can occur if one use more simultaneously SecDecs on multicore computers.

collectres

- The issue coming from missing poles in the SecDec output `.res` files is fixed.

4 Changes in version 1.3

SecDec patch

- In SecDec there is a feature with one could set the desired accuracy of the poles independently, however it was buggy. The bug has been fixed by Zoltan Trócsányi, modifying the following scripts: `getinfo.pm`, `makeint.pm`, `polenumerics.pl`, `preparenumerics.pl`. The patch has been added to the SecDec patch folder and is applied during installation.

In **install.py**

- Patching of SecDec has been updated.

In **run.py**

- The `CUBACORES` enviromental variable is set to be 1 in the beginning of running the script to have control on the parallelization of CUBA.

5 Additional programs

As I mentioned before `ICalc` contains some programs and scripts which can be used standalone. In this section you can find some brief descriptions about them

pythonscripts

In the directory **pythonscripts** there are three scripts written in python to set some parameters in the input files. The scripts `change_accuracy_secdec.py` and `change_accuracy_mb.py` are for changing the relative accuracy of the integration in the input files. `change_maxpoints_mb.py` does pretty the same but changes the parameter *maxpts* in the MB type `.m` files. Each scripts can use bash type pattern matching but the string must be within single quotation marks! The run command follows the same structure for all three: `python script.py '`*your path with patterns within single quotation*`' new value`. They can be very useful if you want to set the same parameter in onehundred files.

collectres

If you have forgot to set the parameter *collect* to yes or one of your results are not good, you may want to collect your results later. `collectres` is for collecting integral results into table formatted `.res files`. To do this type `shscripts/collectres-all.sh` *"projectname" "runname"* into your terminal and hit enter. The script will run `collectres.exe` for every single *integral_somevalues_full.res* in **projectname**/**runname**/**results** and collect them into *integral_all.res* at **projectname**/**runname**/**collected**.

Note: if there are still existing *_all.res* files, the new results will be append to the of the file.

makemath

After you have collected your data you might want to use them in Mathematica not only simple data tables but well definied functions. The program `makemath` will do this job for you, all you need is to have the well-written *configfiles* discussed at section 2.2 before. The running syntax is pretty much the same as for `collectres`: `shscripts/makemath-all.sh` *"projectname" "runname"*. After running all your results from **projectname/runname/collected** will end in a file named *projectname_runname_all.m* (and errors in a file named the same with a *err* preword).

Notes: 1.: in case of an already existing `.m` file the new results will append to the end; 2.: `makemath` streams data with 10 digits of fixed `C++` stream precision! If you need more you can set it easily in the sourcecode.