mcpartools Documentation

Release 0.6.2.post0.dev

Author

Aug 14, 2023

Contents

Contents:

CHAPTER 1

Getting Started with mcpartools

Brief overview of mcpartools and how to install it.

1.1 Introduction

mcpartools is a software simplifying time consuming simulation of particle transport using Monte Carlo codes (Fluka, SHIELDHIT12A). We assume user has access to a computing cluster with batch processing software installed (i.e. slurm, torque) and wants to parallelize simulation by running it simultaneously on many computing nodes. **mcpartools** simplifies this process by generating necessary directory structures and scripts for starting calculations and collecting the results.

mcpartools provides a command line application called `generatemc` which works under Linux operating system (interpreter of Python programming language has to be also installed). No programming knowledge is required from user, but basic skills in working with terminal console are needed.

1.2 Quick installation guide

First be sure to have Python framework installed, then type:

```
pip install mcpartools
```

This command will automatically download and install **mcpartools** for all users in your system. In case you don't have administrator rights, add `--user` flag to `pip` command. In this situation converter will be probably installed in `~/.local/bin` directory.

For more detailed instruction, see installation guide.

1.3 Using generatemc command

Let us start with simple simulation of 10^6 of particles using Fluka MC code. Such simulation would probably take few hours when running on single CPU. It can be however faster, when you submit 100 parallel jobs, each running simulation of 10^4 particles. We assume that:

- you are logged in to the computing cluster, all commands are executed there
- **mcpartools** is installed on the cluster
- cluster has working `slurm` batch job software
- Fluka in installed on the cluster and `rfluka` command is available
- an example Fluka input file is located in `$HOME/sample.inp`

First step is to generate necessary scripts and directory structure. To accomplish this, type in terminal:

```
generatemc --jobs_no 100 --particle_no 100000 $HOME/sample.inp
```

New directory with a name similar to `$HOME/run_20160913_084601` will be created. To start simulation, we call appropriate script:

```
$HOME/run_20160913_084601/submit.sh
```

After the simulation is done (it may take few minutes), run following script to gather the results in a single directory:

```
$HOME/run_20160913_084601/collect.sh
```

Output files from 100 parallel jobs will be saved in `$HOME/run_20160913_084601/output` directory, ready to be analyzed or merged. In case the output is not satisfactory, new workspace can be created and whole process repeated from scratch.

More documentation dealing with advanced options can be found here https://mcpartools.readthedocs.io/

1.4 Features

- user-friendly parallelism of particle transport simulations
- output collected in single directory
- workspace with logs and input files saved for bookkeeping
- Monte-Carlo codes support: SHIELD-HIT12A and Fluka
- cluster batch software support: slurm
- python2 and python3 compatible
- no external libraries needed

1.5 Support

Bugs can be submitted through the issue tracker. Besides the example directory, cookbook recipes are encouraged to be posted on the wiki page

1.6 License

mcpartools is licensed under GPLv3.

CHAPTER 2

User's Guide

2.1 Directory layout and basic workflow

generatemc will create following directory structure for each run:

```
run_20170219_122904                # main directory, new one is created for each␣
↪generatemc call
    collect.sh                     # call to copy output files to output directory
    input                          # reference directory with copy of input files
        sample_fluka.inp           # non-modified copy of original Fluka input files
    submit.sh                      # call to submit jobs to the cluster
    workspace                      # workspace used to store modified input files and␣
↪temporary storage for output
        main_run.sh                # this script will be called by batch system and␣
↪redirect the execution to specific worker
        job_0001                   # working directory of worker no 1
            sample_fluka.inp       # copy of input file, with adapted RNG seed and␣
↪output file path
            run.sh                 # run script executed by worker no 1
        job_0002
            sample_fluka.inp
            run.sh
        job_0003
            sample_fluka.inp
            run.sh
```

After executing submit.sh script, output files will be created in the workspace directory. Each parallel job will store its output in separate directory:

```
run_20170219_122904
   collect.sh
   input
       sample_fluka.inp
   submit.sh
```

(continues on next page)

(continued from previous page)

```
   workspace
       main_run.sh
       job_0001
           sample_fluka.inp
           run.sh
           TODO                    # TODO
       job_0002
           sample_fluka.inp
           run.sh
           TODO                    # TODO
       job_0003
           sample_fluka.inp
           run.sh
           TODO                    # TODO
```

In order to collect all files in a single place, run *collect.sh* script. This will result in following new files:

```
run_20170219_122904
    collect.sh
    input
        sample_fluka.inp
    submit.sh
    output                          # TODO
        TODO                        # TODO
    workspace
        main_run.sh
        job_0001
            sample_fluka.inp
            run.sh
        job_0002
            sample_fluka.inp
            run.sh
        job_0003
            sample_fluka.inp
            run.sh
```

2.2 Advanced options

There are several advanced options in the generator, customising the workflow.

After executing *generatemc* command a directory will be created (i.e. run_20170219_122904), by default in the same location as the input configuration files. In order to change the location of generated directory, use the *–workspace* option. For example after typing:

```
generatemc.py -p 10000 -j 20 tests/res/sample_fluka.inp
```

A directory *tests/res/run_20170717_195410* will be created. Now providing a workspace option:

```
generatemc.py -p 10000 -j 20 tests/res/sample_fluka.inp --workspace mydir
```

will result in new directory *mydir/run_20170717_195557*

Another useful option is the ability to provide additional options for scheduler and for Monte-Carlo binary. The first one can be used i.e. to specify directly the walltime for job execution:

```
generatemc.py -p 10000 -j 20 tests/res/sample_fluka.inp --scheduler_options "[--
↪time=2:00:00]"
```

Note additional square brackets added to distinguish between *generatemc* and scheduler options.

There is also a possibility to do automatic collection of data after calculation. User can also specify desired format of collected data:

```
generatemc.py -p 10000 -j 20 tests/res/sample_fluka.inp -c image
```

Data will be collected automatically after calculation (in this example to the images) so there is no need to run additional *./collect.sh* script. Right now available options are *mv, cp, image, plotdata* and *custom*. User could also provide his own script for collecting the data. In such case the program can be run in following way:

```
CUSTOM_COLLECT=usercollect.sh generatemc.py -p 10000 -j 20 tests/res/sample_fluka.inp␣
↪-c custom
```

One could also specify additional options to Monte-Carlo binary files. For example to add an user-defined particle source in Fluka one can use its *-e* option. If the *flukadpm3_sobp* file is not present in the PATH environmental variable, then its location needs to be known. This may be achieved by a mechanism of creating a link to an external file. Such links can be created by using *-x* switch, here we provide an example in which an external source is enabled by *-e* switch and two external files are linked (*sobp.dat* and *flukadpm3_sobp*):

```
generatemc.py -p 10000 -j 20 tests/res/sample_fluka.inp --mc_engine_options "[-e␣
↪flukadpm3_sobp]" -x ./sobp.dat ./flukadpm3_sobp
```

When using *-x* option you may also set the absolute paths to the linked files.

CHAPTER 3

Detailed Installation Guide

Installation guide is divided in two phases: checking the prerequisites and main package installation.

mcpartools works under Linux operating system.

If you are familiar with python and pip tool, simply type following command to install the package:

```
$ pip install mcpartools
```

If your are a less advanced user, read the rest of the page.

Installation guide is divided in two phases: prerequisites (mainly Python installation) and main package installation.

There are two groups of users: administrators and regular ones. For regular users we assume that they can write files to their home directory, but not necessary elsewhere. We assume that administrators can log in as root user, execute commands via `sudo` or have some other way to create files system-wide.

We expect that **mcpartools** will mostly be used by regular users on computing clusters. This is the reason why installation instruction is lengthy and detailed, explaining how to deal with lack of administrator rights.

3.1 Prerequisites - python interpreter

First we need to check if Python interpreter is installed. Try if one of following commands (printing Python version) works:

```
$ python --version
$ python3 --version
```

At the time of writing Python language interpreter has two popular versions: 2.x (Python 2) and 3.x (Python 3) families. Command `python` invokes either Python 2 or 3, while `python3` can invoke only Python 3.

mcpartools supports most of the modern Python versions, mainly: 2.7, 3.2, 3.4, 3.5 and 3.6. Check if your interpreter version is supported.

If none of `python` and `python3` commands are present, then Python interpreter has to be installed.

We suggest to use the newest version available for your Linux distribution (from 3.x family).

3.1.1 User installation

In case Python is missing and you are regular user, the best would be contact somebody with administrator rights and ask to install Python interpreter in the system.

Installation of Python without administrator rights is however possible, but in case something goes wrong it will require expert knowledge.

As an user you will need to download Python interpreter source code (written in C language) and compile it. For that purpose you will need a C language compiler (i.e. *gcc*) and some other tools (i.e. *make*). These tools are usually installed by somebody with administrator rights. Python installer might not complain about missing SSL libraries (i.e. *libssl-dev*) and will compile successfully, but we recommend to install it (SSL libraries) before, to have easier installation of pip package manager in the next steps.

When installing as user we advice to unpack downloaded source code in *$HOME/tmp* directory and keep it there. It may be needed for upgrade or deinstallation purpose.

Let us install Python 2.7 into *$HOME/usr/py27* directory. First let us create *$HOME/tmp* directory and step into it:

```
$ mkdir -p $HOME/tmp
$ cd $HOME/tmp
```

Now its time to download and unpack source code package. We show an example with 2.7.12 version, but newer one can be problably found on https://www.python.org/downloads/source/

```
$ wget https://www.python.org/ftp/python/2.7.12/Python-2.7.12.tgz
$ tar -zxf Python-2.7.12.tgz
$ cd Python-2.7.12
```

Finally let us start compilation process (it might take couple of minutes). This process might be interrupted by some error message. Do not hesitate to find a professional help to fix it:

```
$ ./configure --prefix=$HOME/usr/py27
$ make
$ make install
```

Python2.7 is now installed into *$HOME/usr/py27* directory. In order to execute python interpreter, you need to provide full path to the executable file, i.e.:

```
$ $HOME/usr/py27/bin/python --version
```

In a similar way python3.x can be installed.

3.1.2 Administrator installation

The best way is to use your package manager.

- `apt-get install python3` (python 3) or `apt-get install python` (python 2) for Debian and Ubuntu
- `dnf install python3` (python 3) or `dnf install python` (python 2) for Fedora
- `yum install python3` (python 3) or `yum install python` (python 2) for CentOS and SLC

3.2 Single file distribution

mcpartools is shipped as a single file executable. It can be downloaded from https://github.com/DataMedSci/mcpartools/releases webpage using the web browser or via command line (here an example with 0.1.4 version is found, newer versions should also be available):

```
$ wget https://github.com/DataMedSci/mcpartools/releases/download/v0.1.4/generatemc.
↪pyz -O generatemc
```

After downloading the file, make sure it has executable bits set:

```
$ chmod ugo+x generatemc
```

When new version if released, replace downloaded file with newer one.

As mcpartools doesn't have any mechanism of automatic updates, we recommend to use installation using **pip** tool, described below. It makes easy upgrade and uninstallation procedure.

3.3 Prerequisites - pip tool

pip is a tool for installing and managing Python packages. It downloads the packages from central Internet repository and installs them in a similar way as apps are downloaded on your smartphone by Google Play or Apple Store.

Try the following commands (printing pip version):

```
$ pip --version
$ pip3 --version
```

In a similar way to python interpreter pip is a tool for Python 2 or 3, while pip3 works exclusively for Python 3. If none of these commands are present, then pip has to be installed.

3.3.1 User installation

Follow the instruction from here https://pip.pypa.io/en/stable/installing/, mainly - download installation script using your web browser, or by typing in the terminal:

```
$ wget https://bootstrap.pypa.io/get-pip.py
```

openssl for python2

Now use your python interpreter to execute downloaded script. It will install pip in your home directory:

```
$ python get-pip.py --user
```

Try if pip command is available by typing:

```
$ $HOME/.local/bin/pip --version
```

If this method fails you can also try to use a *ensurepip* approach. It works with Python versions: 2.7 (starting from 2.7.9), 3.4 and newer. To install pip, simply type:

```
$ python -m ensurepip
```

3.3.2 Administrator installation

Follow the package installation for your system. On some systems instructions mentioned below have to be prefixed with *sudo* command.

- `apt-get install python3-pip` (python 3) or `apt-get install python-pip` (python 2) for Debian and Ubuntu
- `dnf install python3-pip` (python 3) or `dnf install python-pip` (python 2) for Fedora
- `yum install python3-pip` (python 3) or `yum install python-pip` (python 2) for CentOS and SLC

3.4 pip package installation

Now it is time to install **mcpartools** package. It consists of executable file called *generatemc* and bunch of necessary code files.

3.4.1 User installation

User installation will put the **mcpartools** under hidden directory *$HOME/.local*.

To install the package, type in the terminal:

```
$ pip install mcpartools --user
```

If *pip* command is missing on your system, replace *pip* with *pip3* in abovementioned instruction.

To upgrade the **mcpartools** to newer version, simply type:

```
$ pip install --upgrade mcpartools --user
```

To completely remove **mcpartools** from your system, use following command:

```
$ pip uninstall mcpartools
```

In most of modern systems all executables found in *$HOME/.local/bin* directory (*generatemc* executable will be saved there) can be called like normal Linux commands (i.e. *ls*, *cd*). It means that after installation you should be able to simply type in terminal: *generatemc* to use this package

```
$ generatemc --help
```

If this is not the case, please prefix the command with *$HOME/.local/bin* and call it in the following way:

```
$ $HOME/.local/bin/generatemc --help
```

3.4.2 Administrator installation

Administrator installation is very simple, but requires to save some files in system-wide directories (i.e. */usr*). On some systems commands mentioned below have to be prefixed with *sudo* command:

```
$ pip install mcpartools
```

To upgrade the **mcpartools** to newer version, simply type:

```
$ pip install --upgrade mcpartools
```

To completely remove **mcpartools** from your system, use following command:

```
$ pip uninstall mcpartools
```

Now *generatemc* script should be installed for all users and can be invoked by typing:

```
$ generatemc --help
```

CHAPTER 4

Developer documentation

4.1 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/DataMedSci/mcpartools/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs or Implement Features

Look through the GitHub issues for bugs or features. Anything tagged with "bug" or "feature" is open to whoever wants to implement it.

Write Documentation

mcpartools could always use more documentation, whether as part of the official *mcpartools* docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/DataMedSci/mcpartools/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.1.2 Get Started for GIT-aware developers

Ready to contribute? Here's how to set up *mcpartools* for local development. We assume you are familiar with GIT source control system. If not you will other instruction at the end of this page.

1. Fork the *mcpartools* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/mcpartools.git
```

3. If you are not familiar with GIT, proceed to step 5, otherwise create a branch for local development:

```
$ cd mcpartools
$ git checkout -b feature/issue_number-name_of_your_bugfix_or_feature
```

4. Now you can make your changes locally.

As the software is prepared to be shipped as pip package, some modifications of PYTHONPATH variables are needed to run the code. Let us assume you are now in the same directory as `setup.py` file.

The standard way to execute Python scripts WILL NOT WORK:

```
$ python mcpartools/generatemc.py --version
Traceback (most recent call last):
  File "mcpartools/generatemc.py", line 5, in <module>
    from mcpartools.generator import Generator
ImportError: No module named mcpartools.generator
```

To have the code working, two things are needed:

- installation of `versioneer` package (needed to set proper version number)
- adjustment of PYTHONPATH variable.

First let us install `versioneer` package and generate necessary files:

```
$ pip install versioneer
$ versioneer install
```

Now code can be run by typing:

```
$ PYTHONPATH=. python mcpartools/generatemc.py --version
0.1.3.post.dev2
```

5. Make local changes to fix the bug or to implement a feature.
6. When you're done making changes, check that your changes comply with PEP8 code quality standars (flake8 tests) and test against other Python versions with tox:

```
$ flake8 mcpartools tests
$ tox
```

To get flake8 and tox, just pip install them.

7. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
```

8. Repeat points 4-6 until the work is done. Now its time to push the changes to remote repository:

```
$ git push origin feature/issue_number-name_of_your_bugfix_or_feature
```

9. Submit a pull request through the GitHub website to the master branch of `git@github.com:DataMedSci/mcpartools.git` repository.
10. Check the status of automatic tests ran by Travis system.

You can find them on the pull request webpage https://travis-ci.org/DataMedSci/mcpartools/pull_requests. In case some of the tests fails, fix the problem. Then commit and push your changes (steps 5-8).

4.1.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.2, 3.4, 3.5 and 3.6. Check https://travis-ci.org/DataMedSci/mcpartools/pull_requests and make sure that the tests pass for all supported Python versions.

4.1.4 Get Started for non-GIT developers

1. Fetch the code from remote GIT repository to your local directory:

```
$ git clone git@github.com:DataMedSci/mcpartools.git
```

2. Follow steps 4-6 from the instruction for GIT-aware developers. Install versioneer:

```
$ pip install versioneer
$ versioneer install
```

To run code locally, prefix usual calls with `PYTHONPATH=.`:

```
$ PYTHONPATH=. python mcpartools/generatemc.py --version
0.1.3.post.dev2
```

Make your changes and check that they comply with PEP8 code quality standards (flake8 tests) and test against other Python versions with tox:

```
$ flake8 mcpartools tests
$ tox
```

3. Compress your working directory and send it to us by email (see authors), describing your changes.

4.1.5 Tips

To run full tests type:

```
$ tox
```

To run only a single test type:

```
$ PYTHONPATH=. python tests/test_file_to_run.py
```

4.2 Technical documentation

TODO

4.3 Badges and links

docs	
tests	
package	

4.4 Credits

4.4.1 Development

- Leszek Grzanka - IFJ-PAN, Poland <leszek.grzanka@gmail.com>
- ant6

4.4.2 Contributors

None yet. Why not be the first?

Indices and tables

- genindex
- modindex
- search