EasyScan_HEP: a tool for connecting programs to scan parameter space of physics models

Liangliang Shang [*a] and Yang Zhang [†b]

[a] *Department of Physics, Henan Normal University, Xinxiang 453007, China*
[b] *ARC Centre of Excellence for Particle Physics at the Tera-scale, School of Physics and Astronomy, Monash University, Melbourne, Victoria 3800, Australia*

June 28, 2022

Abstract

We present an application, EasyScan_HEP, for connecting programs to scan parameter space of high energy physics models using various sampling algorithms. We develop EasyScan_HEP according to the principle of flexibility and usability. It allows one to set parameters for scan algorithm, connect different programs that calculate physical quantities, and apply constraints in one human-readable configuration file. All programs executed through command line(s) can be connected to EasyScan_HEP with flexible ways to set input parameters and read output parameters of the programs. The current version offers random, grid, Markov Chain Monte Carlo and MultiNest samplers. We also provide functions for resuming, rescan and quick analyses of result. In this manual, we briefly describe the structure of EasyScan_HEP, provide a demonstration of how to use it and some examples for reference.

The first line of the output file is devoted to the legend thatspecifies the entries of the various columns.

[*] shangliangliang@htu.edu.cn
[†] yang.zhang@monash.edu

Contents

1 Introduction

In high energy physics (HEP), scan of parameter space is unavoidable for the numerical studies of physics models. During the scan, many HEP packages may be involved, such as spectrum generators [1, 2], calculation of various observations [3, 4], and imposition of constraints [5, 6]. Data transmission between them can be processed manually or through script. Either way may take a lot of time and effort, especially with large data sets. When using sophisticated sampling methods, like Markov Chain Monte Carlo (MCMC) [7, 8] or MultiNest [9], the connection to HEP packages becomes more complicated.

`EasyScan_HEP` is developed to make the scan easier. Different to the sophisticated scan packages, e.g., `GAMBIT` [10], `MasterCode` [11], `SuperBayeS` [12] and `Fittino` [13], `EasyScan_HEP` is not confined to any specific model. Therefore, there is no calculation of any physical quantities in `EasyScan_HEP`. The disadvantage is that user needs to prepare all the calculation packages, while the advantage is that one can connect any package[1] to `EasyScan_HEP`. `EasyScan_HEP` offers several flexible ways to set input parameters and read output parameters for the external packages. Thus user hardly needs to modify the external packages to fit `EasyScan_HEP`.

`EasyScan_HEP` can rescue us from the tedious and inefficient scan framework in the follow aspects:

- Connection of calculation packages. User only needs to provide the position, execution command(s), input parameters and output parameters of the calculation packages. `EasyScan_HEP` will call the packages as needed in the scan loop.

- Using and setting different sampling algorithms. We reorganize the setting of different sampling algorithms, so that they have one common interface. It is rather convenient to switch sampling algorithm in `EasyScan_HEP`. The current version offers random, grid, MCMC and MultiNest samplers. More sampling algorithms will be involved in the future versions.

- Analyses of the results. Once the scan finishes, `EasyScan_HEP` immediately provides the overall features of the result, including statistical information, histograms, scatter plots, contour plots.

Furthermore, we provide some useful functions, such as break-point resume and re-scan. User can complete above settings just in one human-readable configuration file. All the scan results are saved in one folder in an organised way.

During the development of `EasyScan_HEP`, it has been used in several works [14–19]. The configuration file of Ref. [14] is provided as an example of `EasyScan_HEP`, see Sec. 5, with `makefile` to install the relevant observations calculation packages. From this aspect, `EasyScan_HEP` can help people to repeat the work that uses `EasyScan_HEP` organizing their scan.

The outline of the paper is as follows. We describe the installation instruction in Sec. 2 and the structure of `EasyScan_HEP` in Sec. 3. Definitions and conventions of items in the configuration file are given in Sec. 4. Sec. 5 provides some complex but practical examples of configuration files. The summary is given in Sec. 6.

[1] The package must be able to be executed through command line in the terminal.

2 Quick start

EasyScan_HEP is a Python3 code with dependencies on the numpy [20], scipy [21] and ConfigParser libraries. The optional plot functions and the MultiNest sampler further require matplotlib [23], pandas [22] and pymultinest libraries, respectively. The dependencies can be installed via pip:

```
$ sudo apt install python3-pip python3-tk
$ sudo pip3 install numpy scipy matplotlib ConfigParser pandas pymultinest
```

The repository for EasyScan_HEP is available at

https://github.com/phyzhangyang/EasyScan_HEP.

To download and uncompress EasyScan_HEP, run the following commands at terminal:

```
$ wget https://codeload.github.com/phyzhangyang/EasyScan_HEP/zip/master
$ tar -xf EasyScan_HEP-1.0.tar.gz
```

The "easyscan.py" in folder "bin" is the main program, which is executed with configuration file through the command line,

```
$ ./bin/easyscan.py templates/example_random.ini
```

Here "example_random.ini" is an example configuration file provided in EasyScan_HEP. It performs a scan on a simplified model,

$$f(x, y) = \sin^2 x + \cos^2 y, \tag{1}$$

using random sampler, where x and y are input parameters in range $[0, \pi]$ and $[-\pi, \pi]$, respectively, and f is output parameter.

If the example scan runs successfully, a folder named "example_random" will be generated and there will be a PNG image "fig1.png" in "example_random/Figures" folder similar to the upper left panel of Fig. 1 with less points.

In the "templates" folder, we provide three other instructive configuration files, "example_grid.ini", "example_mcmc.ini" and "example_multinest.ini", that scan the simplified model with different samplers and generate plots similar to the rest panels of Fig. 1. The likelihood function for "example_mcmc.ini" and "example_multinest.ini" is

$$\mathcal{L} = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(f-\mu)^2}{2\sigma^2}\right), \tag{2}$$

where the mean $\mu = 1$ and the standard deviation $\sigma = 0.2$.

All the settings for the scan are done by the human-readable configuration files. Here we explain the configuration file example_random.ini to briefly show the usage of EasyScan_HEP. For full usage and detailed rules of configuration file, see Sec. 4.

Listing 1: Contents of configuration file example_random.ini

```
1 [scan]
2 Result folder name:example_random
3 Sampler:           random
4 #                  VarID  Prior   Min      MAX
5 Input parameters:  x,     flat,   0,       3.14
```

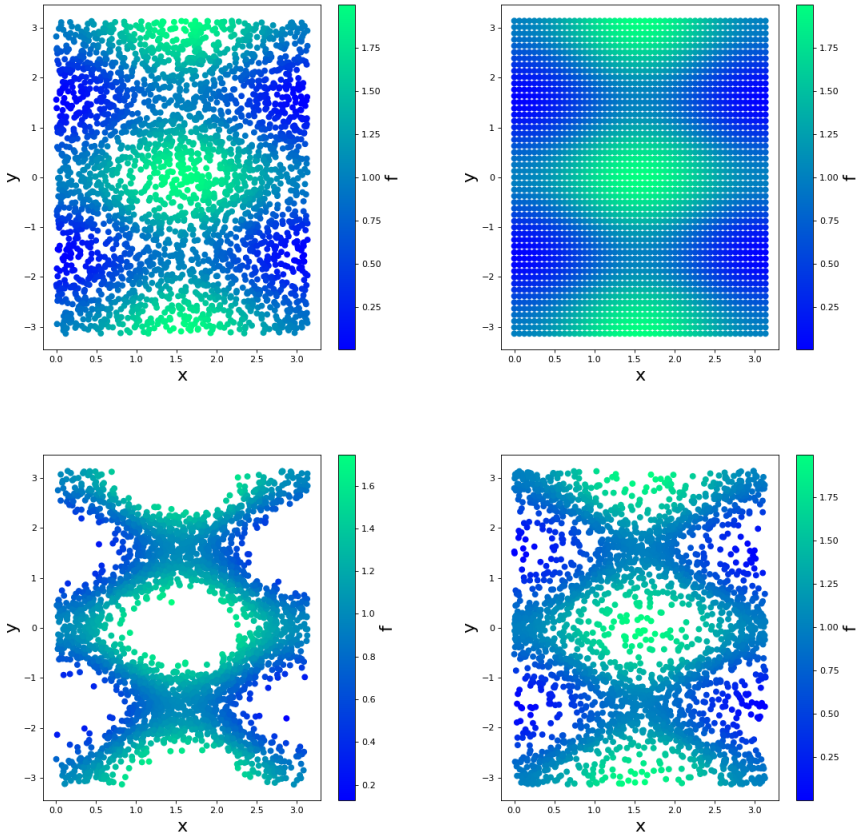

Figure 1: The auto-generated scatter plots for the example configuration files of `EasyScan_HEP`, `example_random.ini` (upper left), `example_grid.ini` (upper right), `example_mcmc.ini` (lower left) and `example_multinest.ini` (lower left).

```
                    y,      flat,    -3.14,    3.14
Points number:  300
[program1]
Execute command: ./TestFuction.py
Command path:    utils/
#                FileID   FileName
Input file:      1,       utils/TestFuction_input.dat
#                VarID    FileID  Method    Row Column
Input variable:  x,       1,      Replace, ID1
                 y,       1,      Replace, ID2
Output file:     1, utils/TestFuction_output.dat
#                VarID    FileID  Method    Row Column
Output variable: f,       1,      Position, 1,  1
[constraint]
#           VarID   Mean   StandardDeviation
Gaussian:   f,      1,     0.2
[plot]
#       X-axis    Y-axis      Color     FigureName
Color: x,        y,          f,        fig1
```

List. 1 shows contents of the configuration file `example_random.ini`. The configuration file consists of sections, each of which contains keys with value entries, following the rules of the `ConfigParser` module[2]. There are four sections in the configuration file `example_random.ini`:

[scan] (`line 1` $\sim$ `line 7`) setting up parameters related to sampler, including following keys

- `"Result folder name"` –name of the folder saving all scan outputs. The value entry for this key in List. 1 is "example_random". When the scan starts, a folder named "example_random" will be generated in the path runing `EasyScan_HEP`. *A unique name has to be set to define the current run and to set up the output folder accordingly.*
- `"Sampler"` –choice of sampler. Here it uses random sampler.
- `"Input parameters"` –input parameters for the scan. Each line of the value entry describes one input parameter. Line 5 of List. 1 stands for x in Eq. (1), while line 6 indicates y in Eq. (1). Each line is comprised of four keyword argument separated by comma: name of the parameter used in the following of configuration file, prior distribution of the parameter, lower limit and upper limit of the parameter. The name of parameter is case insensitive.
- `"Points number"` –total number of the scan points for random sampler. It has different meaning for other sampler, see Sec. 4.1.

[program1] (`line 8` $\sim$ `line 18`) controlling external programs calculating model predictions and other quantities.

As for `example_random.ini`, we use `"EasyScan_HEP /utils/TestFuction.py"` to simulate external program that calculate $f(x, y)$ in Eq. (1). It can be executed through the command line

```
$ ./TestFuction.py
```

The input file of `"TestFuction.py"` is `"TestFuction_input.dat"` shown in List. 2, where the first floating number is x and the second floating number is y.

Listing 2: Contents of input file `"TestFuction_input.dat"`

```
1.5845887764980207      2.95977735836697
```

The output file of `"TestFuction.py"` is saved as `"TestFuction_output.dat`. List. 3 shows the content, which is a floating number, i.e. f.

Listing 3: Contents of output file `"TestFuction_output.dat"`

```
1.96711562785
```

To use `"TestFuction.py"` in `EasyScan_HEP`, the section `[program1]` includes

- `"Execute command"` –command(s) used to execute the program through command line. Thus here the corresponding value entry is "./TestFuction.py".

[2]See https://docs.python.org/3/library/configparser.html for the rules of the `ConfigParser` module.

- `"Command path"` –relative/absolute path where above command should be executed. In List. 1 we use the relative path to the path executing `EasyScan_HEP`.
- `"Input file"` –location of input file. The first element of the value entry is a number identifying the input file, which is "1" in List. 1. The second element is the relative/absolute location path of the input file.
- `"Input variable"` –setting up input variable. Each line of the value entry takes charge of setting one input parameter. The first element of each line is the name of parameter, the second element is the file ID defined in `"Input file"`, and the rest elements decide how to set the parameter. For example, line 15 of List. 1 means that x is located in the place labeled with "ID1". Beside `"Replace"` method, there are several other methods to locate input parameters in input files, descried in Sec. 4.2.
- `"Output file"` –location of output file, which is similar to `"Input file"`.
- `"Output variable"` –reading output variable. Each line of the value entry takes charge of getting information from `"Output file"`, whose rule is similar to `"Input variable"` but without `"Replace"` method. In this example, line 19 of List. 1 means that f is getting value from the first row and the first column.

[**constraint**] (`line 19` $\sim$ `line 21`) applying constraints during the scan, including

- `"Gaussian"` –one dimension Gaussian distributed constraint. The likelihood function Eq. (2) is implemented by setting variable to f, mean to 1 and standard deviation to 0.2. It is not used to guide the scan for random and grid samplers, but is critical for sophisticated samplers, i.e. MCMC and `MultiNest` samplers.

[**plot**] (`line 22` $\sim$ `line 24`) generating plots for the scan results, inculding

- `"Color"` –colorful scatter plot. In `example_random.ini`, X-axis and Y-axis are x and y, colors indicates f, and figure name is `"fig1"`. After scan, a PNG image named `"fig1.png"` in `"example_random/Figures"` will be generated.

After the scan finish, the outputs are stored in the output folder named "example_random" in [scan] section. For random sampler, it contains

– `"ScanInf.txt"` explain the data files of the scan results. List. 4 presents the contents of `"ScanInf.txt"` for `example_random.ini`. The first line is the absolute path of the output folder and the name of data file. The rest of lines has two columns: the first column is the name of variables including `"Input parameters"` in [scan] section, `"Input variable"` and `"Output variable"` in [program*] section, equivalent χ^2 for each constraint (Here named "Gaussian_f") in [constraint] section and total equivalent χ^2 named `"Chi2"`; the second column indicate the zero-based column numbering for corresponding variable in `"RandomData.txt"`.

Listing 4: Contents of `"ScanInf.txt"` for `example_random.ini`

```
1 /home/*/EasyScan_HEP/example_random     RandomData.txt
2 y             0
3 x             1
```

```
4 f           2
5 Chi2        3
6 Gaussian_f  4
```

- "RandomData.txt" storing samples that execute external program(s) successfully. Each line stands for one sample and the meaning of columns are indicated in "ScanInf.txt".
- "All_RandomData.txt" only for "MCMC" method.
- "Figures" saving figure(s) generated in [plot] section, which is "fig1.png" for the example.
- "SavedFile" saving output file(s) of external program(s) for each point, which is empty for "example_random.ini".

The output folders for other samplers are explained in Sec. 4.1.

Besides normal mode, EasyScan_HEP can be executed in debug mode by

```
$ ./bin/easyscan.py templates/example_random.ini -d
```

with more screen outputs for debugging, and in resume mode by

```
$ ./bin/easyscan.py templates/example_random.ini -r
```

in case previous scan is aborted incidentally. For further functions and rules for configuration file in Sec. 4 and complex examples in Sec. 5.

3 Structure

Fig. 2 outlines the directory structure of the package.

EasyScan_HEP has the five blocks shown in Fig. ??. In the Input block, EasyScan_HEP needs input information involves Scan information, Program information, Constrain information and Plot information, which have been shown in Sec. ??. Scan information decide to use which scan method in the block Scan Tools. Then Scan Tools returns the scanning result to the main block of EasyScan_HEP, the main block of EasyScan_HEP put the values of scanning result in the input files of programs. The main block transmits these input files to External Tools and call the program to run. When running the programs if you choose "MCMC" scanning method, EasyScan_HEP still call the block "Scan Tools" to decide the direction of scanning. In the process of running EasyScan_HEP, EasyScan_HEP saves the results and puts the output variables in a file. Finally, EasyScan_HEP will plot the figures.

Now we show the functions of folders in EasyScan_HEP as following,

- Folder of bin

  There stored the main executable program named easyscan.py and the output file named by the option Result file name in [scan] in the configuration file.

- Folder of src

  There stored the internal functions of EasyScan_HEP. mainfun.py is used for connecting programs defined in External Tools and reading and outputting file. readin.py

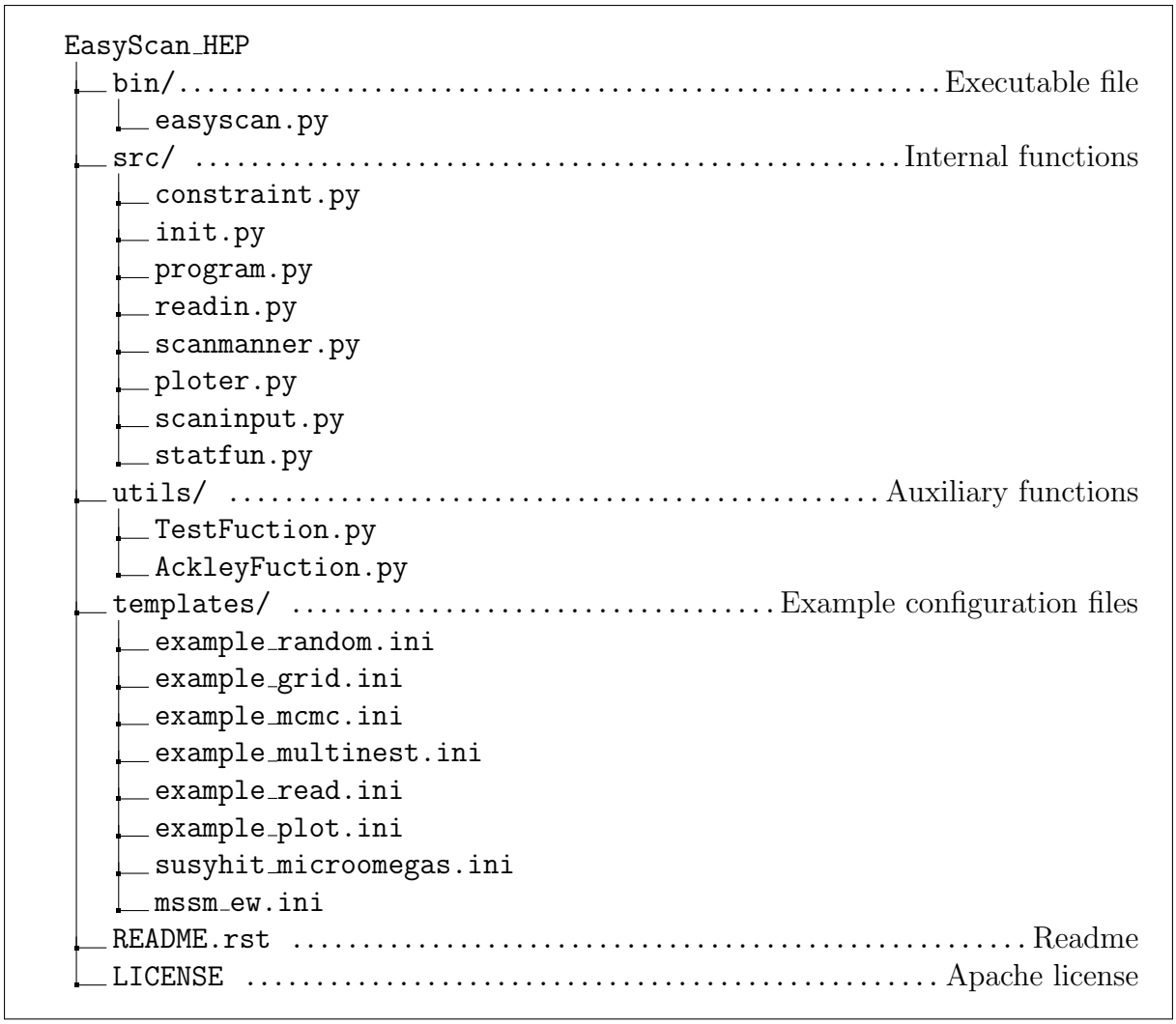

Figure 2: Directory structure of `EasyScan_HEP`.

is used for parsing the configuration file. `scanmanner.py` involves different scanning methods. `statfun.py` involves some statistical functions. `subfun.py` involves some auxiliary functions.

- Folder of `SAMPLE`

  In this folder, there are some configuration files for users to refer.

- Folder of `Externals`

  This folder stores external programs, such as `SPheno` and `Micromegas`. But this is not necessary because external programs could be placed anywhere if only you set their pathes in the configuration file.

- Folder of `Internals`

  There is the testing program `zytest` in this folder now. In updated version, we will give more packages in this folder those are often used in high energy physics. Meanwhile we give the corresponding configuration files in the folder `SAMPLES`.

4 Configuration file

We use `ConfigParser` module to read configuration file. Thus a configuration file consists of sections, each led by a `[section]` header, followed by key/value entries separated by a colon. Lines prefixed by `"#"` are comments. Sections and keys are fixed in `EasyScan_HEP`, and one writes value entries to set up scan. In following, we list all sections and keys in `EasyScan_HEP`, and describe rules for writing value entries.

4.1 `[scan]`

Table 1: Rules for `[scan]` section

Key	Type of value	Mandatory	Multi-line	Component
Result folder name	string	Yes	No	1
Sampler	**Random** **Grid** **MCMC** **MultiNest**	Yes	No	1
Input parameters	string,**Flat**,float,float,∗ string,**Log**,float,float, ∗	Yes	Yes	≥ 4
Points number	integer	Yes †	No	1

† Not mandatory for grid sampler.

This part sets the basic information of scanning. Each information involves "name" and "parameter". The part before colon is "name", this is used for show functions of this information and could not be modified. But some of them could be deleted. After the colon, it is "parameter", and for different parameters, their contents are different.

- `Result folder name`

  This folder stored the names of output data and results. After running of `easyscan.py`, it will generate the folder of the name in the path of running of `easyscan.py`. If the folder have existed, it will show the message,

  ```
  $ * The Result folder [test] already exists.
  $ Choose: (r)replace , (b)backup , (s)stop
  ```

  Inputting "r" stands for delete the old folder and generate the new one. Inputting "b" stands for backup the old folder by moving it in the folder `Backup` and generate the new one. Inputting "s" stands for stoping `EasyScan_HEP`.

- `Scan method`

  Scanning method must be chosen from the following four modes, which are case-insensitive.

  - `Random`: random scanning.
  - `Grid`: grid scanning, i.e., taking equal intervals for each parameter.
  - `MCMC`: Metropolis-Hastings algorithm. For details, see in App. A.

- `Read`: Read the parameters gotten before, and recalculate output and plot. In this mode, there need to have output parameters gotten before such as those in MCMCData.txt and copy ScanInf.txt file as ScanInfINPUT.txt. Data of file in the first line of ScanInfINPUT.txt would be read.
- `PLOT`: Plot new figures with the parameters gotten before. In this mode, the output parameters before must be existing.

- `Input parameters`: Input parameters in `EasyScan_HEP` and each of them in a line, in each line it have the following contents according to the scan method.
  - "Random" mode: each input parameter contains information that has a variable name (must begin with a letter, cannot have duplicate names, case-sensitive), distribution functions (currently only "Flat" and "Log" are available for selection, case-insensitive), scan range minimum and maximum (must be real numbers). These four items are the most basic, and each scanning mode should include them. These contents are fixed in order separated by commas, for example

```
#                  VarID  Distribution Min Max
Input parameters: tanb, Flat, 2.0, 60.0
```

  - "Grid": the above basic four items + the number of intervals (if not provided, the default value is 30). For example, for $\tan\beta$ (in the range of $[2, 60]$), the number of intervals of 10 means a total of $10 + 1$ values which starts at 2 and go to 60 at the interval (60-2)/10. The corresponding expression is

```
#                  VarID Distribution Min Max NumberOfIntervals
Input parameters: tanb, Flat, 2.0, 60.0, 10
```

  - "MCMC": the above basic four items + the number of intervals (if not provided, the default value is 30) + the initial value, for example

```
#                VarID  Distribution Min Max NumberOfIntervals
   Initial
Input parameters: tanb, Flat, 2.0, 60.0, 10, 8.1
```

    If the initial value is not provided, the default value is the intermediate value, (max-min)/2. It is better to set the initial value not to close the min or max value.
  - "Read": Variable name only. Because the pattern does not scan the parameter space, it reads the existing parameters only, there is no need to provide more information except the above basic four items.

- `Points number`: It stands for the number of scanned sample points and it must be a positive integer. For the "Random" mode, it is the total number of scanned samples. For the "MCMC" mode, it is the number of reserved samples according to the Metropolis-Hastings algorithm [7]. While for "Grid" and "Read" modes, it is unnecessary and could be reserved, deleted or commented.

- `Interval of print`: It stands for the interval of screen output samples and must be a positive integer. To prevent the brush screen, you can output information of samples after some points. If all elements defined in "Output variable" are read by easyscan successfully, easyscan would show information as following every interval.

- `Random seed`: It stands for the random number of seeds. If "-1" is taken or it is not given, the system's time when running `./easyscan` is used as the random number seed.

- `Acceptance rate`: It stands for the acceptance rate of the MCMC algorithm and should be real number between 0 to 1, whose function is to adjust step size. If you do not resize the step, delete or comment the line.

4.2 [program]

Table 2: Rules for `[program]` section

Key	Type of value	Mandatory	Multi-line	Component
Execute command	string	Yes	Yes	1
Command path	string	Yes	No	1
Input file	integer,string	No	Yes	2
Input variable	string,integer,**Position**,integer,integer string,integer,**Label**,string,integer string,integer,**Replace**,string string,integer,**SLHA**, BLOCK, $*$ string,integer,**SLHA**, DECAY, $*$	No	Yes	≥ 4
Output file	integer, string	No	Yes	2
Output variable	string,integer,**Position**,integer,integer string,integer,**Label**,string,integer string,integer,**SLHA**, BLOCK, $*$ string,integer,**SLHA**, DECAY, $*$ string,integer,**File**, SAVE	No	Yes	≥ 4
Bound	string, $*$	No	Yes	≥ 3

```
[program1]
Program name:    softsusy-3.7.3
Execute command: ./softsusy-3.7.3/ softpoint.x leshouches < softsusy-3.7.3/
    inOutFiles/lesHouchesInput > LesHouchesOutput.txt
Command path:    ../Externals/
Input file:      1, ../Externals/softsusy-3.7.3/inOutFiles/LesHouchesInput.txt
#                VarID FileID   Way     others
Input variable:   m0IN,  1, Position,      14, 2
                 m12IN,  1,    Label,     m12, 2
                tanbIN,  1,  Replace, ES_tanb
```

```
                  AOIN,  1,     SLHA,   BLOCK, MINPAR, 5
Output file:     1, ../Externals/lesHouchesOutput
#                VarID FileID Way others
Output variable:   mh,   1,  SLHA, BLOCK, MASS, 25
                   mn1,   1,  SLHA, BLOCK, MASS, 1000022
                  spec,   1,  File, SAVE
#                VarID  LowLimit UpLimit
Bound:              mh,   120,     130

[program2]
Program name:    micromegas_4.3.1
Execute command: ./micromegas_4.3.1/MSSM/main LesHouchesOutput.txt >
   micromrgas.out
Command path:    ../Externals/
Input file:
Input variable:
Output file:     1, ../Externals/micromrgas.out
#                VarID FileID Way others
Output variable: omega, 1, label, Omega=, 3
Time limit minute: 10
```

Listing 5: `[programX]` section of the example configuration file

This section provides information about external programs that can contain multiple external programs, each of which takes up a section. The name of the section is `[programX]`, where "X" is a positive integer. For different sections, the number cannot be the same. In order of the size of the number "X", but not in the order in which the section is written, each section is running[3]. Each section contains information about

- `Program name`: It is external software name and can be represented in any string, but cannot be wrapped.
- `Execute command`: It is the commands of running the external softwares. You can write more than one command, but one command in one line. Note there would be no comma in Execute command because of comma interpreted as the separator.
- `Command path`: The directory where the above commands are running. It can be an absolute path or a relative path relative to the path of running `./easyscan.py`.
- `Input file`: It is input file names of external softwares involving paths which are absolute paths or relative paths relative to the path of running `./easyscan.py`. It could contain multiple input files and one file per row. Each input file corresponds to two quantities, separated by comma: The first quantity is a file number which is a positive integer that represents the different input file, and the second quantity is the input file with its path.
- `Input variable`: It is used to describe which of the parameters in input files of the external softwares are varying, i.e., need to be scanned. In the scan, the values of

[3]These numbers need not to be start from 1 and also could be discrete, for examples, "[program1],[program2],[program3]" or "[program2], [program3], [program5]" are both proper.

input parameters are placed in the right positions of input files according to this description here. Input parameters can be parameters defined in `Input parameters` in `[scan]`, or the output parameters of external programs ordered before this section. Each input parameter contains the case-sensitive variable name, the file number, the way about how to arrange the input variables in input files, which involves "Position", "Replace", "Label" and "SLHA" now, and supplementary information for the way. Assuming that the contents of input file "LesHouchesInput.txt" is (note that its file ID is 1)

```
Block MINPAR
1 1.250000000e+02 # m0
2 9.000000000e+02 # m12
3 1.000000000e+01 # tan beta at MZ
4 1.000000000e+00 # sign(mu)
5 0.000000000e+00 # A0
```

We give an example that how to replace the value $1.250000000e+02$ of $m0$ in the file "LesHouchesInput.txt" by the value of $m0IN$ in `[scan]` by the four ways.

- "Position": Determines the position where the value of input parameter place based on the order of row and column, including two arguments for the number of row and column. The column is divided by space or tab.

  ```
  #              VarID FileID Way  Row Column
  Input variable: m0IN, 1, Position, 2, 2
  ```

- `Label`: The value to be replaced is in an specific line in which it involves an identifier that only appear once in the input file. Two information is required: the identifier (supporting regular expression) and the order of column (split by one or more blank space or tabs and being counting from one) in the line. The order of column are 1 (-1) and 2 (-2) for the first (last) and second (last but one) element respectively, and so on.

  ```
  #              VarID FileID Way Identifier Column
  Input variable: m0IN,   1, Label, m0,  2
  ```

- `SLHA`: This can be used if the input file is a SLHA [24] format file or one that has a similar structure. The necessary information includes: (A) The choice is "block" or "decay"; (B) The name of "block" [4] or the PDG code of the decaying particle; (C) Key of variable.

  ```
  #               VarID FileID Way Section SectionName Keys
  Input varialbe: m0IN,  1,  SLHA, BLOCK,  MINPAR,  2
  ```

  If the file has the following contents (assuming the file has ID as 2),

  ```
  BLOCK UMIX # Chargino Mixing Matrix U
  1 1 -9.69283723E-01 # U_11
  1 2 2.45945247E-01  # U_12
  ```

[4] The name of "block" is case sensitive. And there could not be two same blocks or decays in the giving SLHA format file.

```
2 1 2.45945247E-01   # U_21
2 2 9.69283723E-01   # U_22
#
DECAY 1000023 1.22776237E-04 # neutralino2
8.88469593E-04 2 1000022 22    #BR(~ chi_20 -> ~chi_10 gam)
1.15050111E-01 3 1000022 -2 2 #BR(~ chi_20 -> ~chi_10 ub u)
```

where values of U_{11}, $\Gamma_{\tilde{\chi}_2^0}$ and $BR(\tilde{\chi}_2^0) \to \tilde{\chi}_1^0\gamma$ could be replaced as following[5],

```
#                VarID       FileID Way Section SectionName Keys
Input variable: U11IN,        2, SLHA, BLOCK, UMIX, 1, 1
                Gamma_n2IN,   2, SLHA, DECAY, 1000023, 0
                BRn2_n1GaIN, 2, SLHA, DECAY, 1000023, 2, 1000022, 22
```

– `Replace`: If the input file is not generated by the previous external softwares, you can change the value who you want to replace by the value of input variable, to an identifier which is unique in the input file. For example, modify the file "LesHouchesInput.txt" as following,

```
Block MINPAR # Input parameters
1 ES_m0 # m0
2 9.000000000e+02 # m12
3 1.000000000e+01 # tan beta at MZ
4 1.000000000e+00 # sign(mu)
5 0.000000000e+00 # A0
```

Then we could use "Replace" method as following,

```
#                VarID FileID Method Identifier
Input variable: m0IN, 1, Replace, ES_m0
```

Before the scan begins, `EasyScan_HEP` backs up the input file involving the identifier (i.e., adding the suffix ".ESBackup" after the name of file) and deletes the backup file when `EasyScan_HEP` finishes successfully. If the program is accidentally or abnormally interrupted, the backup file needs to be restored (i.e., remove the suffix ".ESbackup") before the script `./easyscan` reruns.

– Finally, in the `Input variable`, we could use calculation expression for each replacing method[6]. For example,

```
Input variable: m0IN+m12IN,        1, Position, 2, 2
                m0IN+pow(m12IN;2), 1, Position, 2, 2
```

In the expression, it could involve python build-in calculating functions [7] and functions from math module in python [8].[9] And using this method, we could

[5] $BR(\tilde{\chi}_2^0) \to \tilde{\chi}_1^0\bar{u}u$ could also be replaced by setting `1000023, 3, 1000022, -2, 2`

[6] This function also could be used in "Bound" in [programX] and "Gaussian" in [constraint].

[7] It involves: abs, complex, divmod, float, int, long, pow, range, round, sum, oct, hex, chr, bin, bool.

[8] It involves: ceil, copysign, fabs, factorial, floor, fmod, frexp, fsum, isinf, isnan, ldexp, modf, trunc, exp, expml, log, log1p, log10, pow, sqrt, acos, asin, atan, atan2, cos, hypot, sin, tan, degrees, radians, acosh, asinh, atanh, cosh, sinh, tanh, erf, erfc, gamma, lgamma, pi, e.

[9] If the function in math module have the same name with build-in calculation function, we use the one in math module. Also note that in these function, we use ";" other than ",".

fix parameter, for example, we want to set $\tan\beta = 10$, we can do

```
Input variable: sqrt(100),        1,  SLHA, block, EXTPAR, 25
```

- `Output file`: Its usage is similar to `Input file`. Note that if a scanning sample cannot produce desired output files during the scan, or an output variable cannot be read correctly from the output file, the sample will be set to a non-physical sample.

- `Output variable`[10]: Its usage is like that of "Input variables", but it doesn't support the way "Replace" and add two new ways "File"[11] and "Calculate". The "File, SAVE" mode saves the output file to the output folder and note that in this mode the name can be arbitrary[12]. For example,

```
#                 VarID   FileID Way
Output variable: spectrum, 1, File, SAVE
```

  ~~"Calculate" mode~~ (replaced by math support) does simple calculation with input variables involving the output variables generated by the previous external softwares. For example, we calculate the $(m0)^2$ and save the result in the variable "m0sq",

```
#                 VarID FileID Way      Formular
Output variable: m0sq, 1, Calculate, var[m0]**2
```

  where the file ID `1` is not used but there must be a positive integer placed. The variables in the formula are labeled as "var[X]", and X represents the variable name.

- `Bound`: It support the following formats,

```
Bound:      mh, 120, 130
            mh, <=, 130
            mh, sigmaSI, upper, ../External/limit.txt
```

  ~~The first line stands for mass of Higgs should be in the range between 120 GeV and 130 GeV,~~ the second line stands for the mass of Higgs should be "<="[13] 130 GeV [14] . It also support bound from external file just as the third line shown. The file contains data used for interpolation. For example, there are two columns in limit.txt, the first one stands for Higgs mass, the second one stands for spin-independent cross section. Then we get interpolating value of sigmaSI according to mh and compared sigmaSI with the interpolationg value according to option "upper" or "lower". For

[10] If `Input file` and `Input variable` or `Output file` and `Output variable` are not used, their contents keep empty as shown in `[program2]`. Variable is assigned to `NaN` if searching mode could not be matching besides 0 for `SLHA, DECAY`. This is because one decay mode is not be written when its branching ratio is zero for many particle decay calculator.

[11] Corresponding values are set as zero in "XData.txt" or "All_Xdata.txt" file, where "X" represents the scanning mode.

[12] The information about method "File, SAVE" in data file such as GridDATA.txt is numbered with points numbering ID.

[13] It also support ">=", "<=", ">", "<", "==" and "!="

[14] It also support variable name, such as mh, mn1 and etc.

upper (lower) bound, it is allowed when sigmaSI less (greater) than interpolating value. [15]

- `Time limit minute`: It can be used to limit the running time of program, such as 5 minutes shown in [program2] in the example.

4.3 [constraint]

Table 3: Rules for `[constraint]` section

Key	Type of value	Mandatory	Multi-line	Component
Gaussian	string, float, float, $*$	Yes †	Yes	≥ 3
FreeFormChi2	string	Yes †	Yes	1

† Not mandatory for grid sampler or random sampler. For MCMC sampler or MultiNest sampler, one of Gaussian and FreeFormChi2 is mandatory at least.

```
[constraint]
Gaussian: mh,   125.0,     2.0
          omg, 0.1199, 0.00012, upper
FreeFormChi2: pow(mh-125;2)/2
Limit:    mn1,  sigma, externals/LUX.txt, lower
Chi^2:    chisq_mh, 1
```

Listing 6: `[constraint]` section of the example configuration file

This section is designed to add experimental limits to the scan. For these scanning method "Random", "Grid" and "Read", this part can be existed, deleted or commented. But for the method "MCMC", this part must be existed. If this part exists, the output will be divided into two categories, one of which is to meet each experiment limits at 95% confidence level stored in the file named "XData.txt" file, where "X" represents the scanning mode and the other one is all physical samples stored in the file "All_Xdata.txt". The limitations currently contain the followings,

- `Gaussian`: The Gaussian distribution. The basic parameters are the variable name, the center value of the Gaussian distribution and the standard deviation. The optional parameters are one chosen from "upper" or "lower", representing the half of the Gaussian distribution and name of chi square. Options "upper" ("lower") means upper (lower) bound and so it is allowed in the case whose value is less (greater) than the upper (lower) bound.[16]

- `FreeFormChi2`: This is free form chi square distribution, where user could give arbitrary expression of chi square. The basic parameter is expression of chi square. The optional parameter is name of chi square. Through this method, it is easy to add chi square from external programs such as HiggsSignal into total chi square of `EasyScan_HEP` scanning.

[15] We properly deal with cases outside range of data when interpolation. For example, if mh is out of range of first line in limit.dat, we set interpolating value of sigmaSI as infinity (minus infinity) when upper (lower) is set. In other words, it is allowed for cases outside range of data whatever.

[16] Technically, we set chi2 equal to zero when value is less (greater) than the upper (lower) bound, or set chi2 normally.

- ~~`Limit`~~ `replaced by Bound`: It is used to add upper or lower limits. It contains four parameters: the first variable name (X), the second variable name (Y), the data file and the upper or lower limit selection. [17] The contents of the data file is two-dimensional arrays of two columns and many rows. The meaning of the first column is corresponding to that of the parameter (X) and its range should involving values of (X) and the second column is the corresponding upper or lower limit. If the point (X,Y) is not in the data file, we will use the interpolation. So the first column should be ascending or descending ordered and the second column's dependencies on the first column should be monotonous.

- ~~`Chi2`~~ replaced by FreeFormChi2: This part should contain two parameters: the first parameter is the output variable name of external softwares, whose purpose is to construct χ^2 with this parameter; the second one is the degree of freedom of χ^2 distribution and must be a positive integer. When implemented the scan, `EasyScan_HEP` could calculate the confidence based on the degree of χ^2 to determine whether the sample points are excluded or not.

4.4 [plot]

Table 4: Rules for `[plot]` section

Key	Type of value	Mandatory	Multi-line	Component
Histogram	string, $*$	No	Yes	≥ 1
Scatter	string, string, $*$	No	Yes	≥ 2
Color	string, string, string, $*$	No	Yes	≥ 3
Contour	string, string, string, $*$	No	Yes	≥ 3

```
[plot]
Histogram: mh
           m1
Scatter:   mn1, sigma
            m0, m12
Color:      m0, m12, mh
Contour:    m0, m12, mh
```

Listing 7: `[plot]` section of the example configuration file

This section is to show how to plot. Currently it supports the following graphics,

- `Histogram`: It need one parameter and will plot histogram for this parameter. It could generate many histograms for many parameters with each of them in a new line.

- `Scatter`: It need two parameters which are x-axis and y-axis of the two dimension graph respectively.

[17] In high energy physics, some limits are on a plane, such as limits on dark matter-nucleon scattering cross section in the dark matter direct searching experiments. These limits depend on the mass of dark matter. Here the variable (X) is the mass of the dark matter and for each of them, there is the upper bound on the scatter cross section. And `EasyScan_HEP` will compare the variable (Y) to the upper bound to decide whether accept this point or not.

- `Color`: This is scatter plot with colorbar and needs three parameters which are x-axis, y-axis and colorbar.
- `Contour`: This is contour map and needs three parameters which are x-axis, y-axis and the height of contours.

These pictures are saved in the folder "Figures" in the output folder, named as "X_Y_1_Y_2..." where X is the graphic category, such as "Histogram", and Y_i represents the variable names used for the drawing.

5 Examples

ss

6 Summary

This paper introduces the workflow and usage of `EasyScan_HEP`. Essentially, `EasyScan_HEP` provides an easy-to-use platform with a rich input/output interface for connecting different high-energy computing software. Then we can easily use the built-in scanning algorithm to study the parameter space of different models. It also provides an automated visualization module that can quickly understand the results at the end of the scan.

To keep track of developments, report any bugs or ask for help, please see https://github.com/michaelhb/superplot/issues.

Acknowledgement

This work is supported in part by the National Science Foundation of China (11705048). We thank Kun Wang for fixing a bug in `Bound` function.

A Statistical functions

In the process of scanning, we use Markov Chain Monte Carlo (MCMC) based on Metropolis-Hastings algorithm. By this method, a sample chain with a node density proportional to the posterior probability density function (PDF) can be obtained. The posterior probability density function represents the cognition of the parameter η after the known experimental data d, and the formula is expressed as,

$$p(\eta|d) = \frac{p(d|\epsilon(\eta))\pi(\eta)}{p(d)} \tag{3}$$

where $\epsilon(\eta)$ is the experimental observable, the likelihood function $p(d|\epsilon(\eta)) \equiv \mathcal{L}(\eta)$ represents the probability density function of the data d from the experimental measurement ϵ, and the priori probability density function $\pi(\eta)$ represents the cognition of parameters before the measurement result is used. The most common use of MCMC is the integration of complex functions, but we mainly use them to get as many samples as possible to meet the experimental limit in limited time and computational resources here.

Metropolis-Hastings algorithm is the most basic MCMC method. Metropolis et al. first put forward Metropolis algorithm [7], where the proposed distribution is symmetric,

that is $q(x|y) = q(y|x)$ which is that the probability of transferring from x to y is equal to the probability of transferring from y to x. Later Hastings improved the above algorithm to allow to use the asymmetric proposed distribution. In simple terms, the basic idea of MCMC method is to start from the initial point x^0 and to get the Markov chain iteratively. The process from the ith node x^i to the $(i+1)$th node x^{i+1} is,

- According to the proposed distribution function $q(x^i|x^j)$, a test point x^j is obtained;
- Calculate acceptance probability

$$\rho = \min\left(1, \frac{\pi(x^j)}{q(x^j|x^i)} \frac{q(x^i|x^j)}{\pi(x^i)}\right) \tag{4}$$

  where $\pi(\eta) = p(\eta|d)$ is obtained by comparing the physical observables to their corresponding experimental results.

- Produces a uniformly distributed $\mathcal{U}(0,1)$ random number u. If $u < \rho$, accept the test point, make $x^{i+1} = x^j$; if $u > \rho$, reject the test point, go back to the first step.

References

[1] W. Porod and F. Staub, Comput. Phys. Commun. **183**, 2458 (2012) doi:10.1016/j.cpc.2012.05.021 [arXiv:1104.1573 [hep-ph]].

[2] P. Athron, M. Bach, D. Harries, T. Kwasnitza, J. h. Park, D. Stöckinger, A. Voigt and J. Ziebell, Comput. Phys. Commun. **230**, 145 (2018) doi:10.1016/j.cpc.2018.04.016 [arXiv:1710.03760 [hep-ph]].

[3] G. Bélanger, F. Boudjema, A. Goudelis, A. Pukhov and B. Zaldivar, Comput. Phys. Commun. **231**, 173 (2018) doi:10.1016/j.cpc.2018.04.027 [arXiv:1801.03509 [hep-ph]].

[4] F. Mahmoudi, Comput. Phys. Commun. **180**, 1579 (2009) doi:10.1016/j.cpc.2009.02.017 [arXiv:0808.3144 [hep-ph]].

[5] P. Bechtle, O. Brein, S. Heinemeyer, O. Stål, T. Stefaniak, G. Weiglein and K. E. Williams, Eur. Phys. J. C **74**, no. 3, 2693 (2014) doi:10.1140/epjc/s10052-013-2693-2 [arXiv:1311.0055 [hep-ph]].

[6] D. Dercks, N. Desai, J. S. Kim, K. Rolbiecki, J. Tattersall and T. Weber, Comput. Phys. Commun. **221**, 383 (2017) doi:10.1016/j.cpc.2017.08.021 [arXiv:1611.09856 [hep-ph]].

[7] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, E. Teller, "Equations of State Calculations by Fast Computing Machines," *J. Chem. Phys.* 21, 1087 (1953).

[8] W.K. Hastings, "Monte Carlo sampling methods using Markov chains and their applications," *Biometrika*, 1970 57(1):97.

[9] F. Feroz, M. P. Hobson and M. Bridges, Mon. Not. Roy. Astron. Soc. **398**, 1601 (2009) doi:10.1111/j.1365-2966.2009.14548.x [arXiv:0809.3437 [astro-ph]].

[10] P. Athron *et al.* [GAMBIT Collaboration], "GAMBIT: The Global and Modular Beyond-the-Standard-Model Inference Tool," Eur. Phys. J. C **77**, no. 11, 784 (2017) Addendum: [Eur. Phys. J. C **78**, no. 2, 98 (2018)] [arXiv:1705.07908 [hep-ph]].

[11] O. Buchmueller *et al.*, JHEP **0809**, 117 (2008) doi:10.1088/1126-6708/2008/09/117 [arXiv:0808.4128 [hep-ph]].

[12] F. Feroz, K. Cranmer, M. Hobson, R. Ruiz de Austri and R. Trotta, JHEP **1106**, 042 (2011) doi:10.1007/JHEP06(2011)042 [arXiv:1101.3296 [hep-ph]].

[13] P. Bechtle, K. Desch and P. Wienemann, Comput. Phys. Commun. **174**, 47 (2006) doi:10.1016/j.cpc.2005.09.002 [hep-ph/0412012].

[14] G. Pozzo and Y. Zhang, Phys. Lett. B **789**, 582 (2019) doi:10.1016/j.physletb.2018.12.062 [arXiv:1807.01476 [hep-ph]].

[15] P. Athron, C. Balazs, A. Fowlie and Y. Zhang, JHEP **1802**, 121 (2018) doi:10.1007/JHEP02(2018)121 [arXiv:1711.11376 [hep-ph]].

[16] J. Cao, J. Li, Y. Pan, L. Shang, Y. Yue and D. Zhang, Phys. Rev. D **99**, no. 11, 115033 (2019) doi:10.1103/PhysRevD.99.115033 [arXiv:1807.03762 [hep-ph]].

[17] J. Cao, L. Feng, X. Guo, L. Shang, F. Wang and P. Wu, Phys. Rev. D **97**, no. 9, 095011 (2018) doi:10.1103/PhysRevD.97.095011 [arXiv:1711.11452 [hep-ph]].

[18] J. Cao, L. Feng, X. Guo, L. Shang, F. Wang, P. Wu and L. Zu, Eur. Phys. J. C **78**, no. 3, 198 (2018) doi:10.1140/epjc/s10052-018-5678-3 [arXiv:1712.01244 [hep-ph]].

[19] J. Cao, X. Guo, Y. He, L. Shang and Y. Yue, JHEP **1710**, 044 (2017) doi:10.1007/JHEP10(2017)044 [arXiv:1707.09626 [hep-ph]].

[20] S. v. d. Walt, S. C. Colbert, and G. Varoquaux, "The numpy array: A structure for efficient numerical computation," *Computing in Science & Engineering* **13** no. 2, (2011) 22–30.

[21] E. Jones, T. Oliphant, P. Peterson, *et al.*, "SciPy: Open source scientific tools for Python," 2001. http://www.scipy.org/.

[22] W. McKinney, "Data structures for statistical computing in Python," in *Proceedings of the 9th Python in Science Conference*, S. van der Walt and J. Millman, eds., pp. 51 –56. 2010.

[23] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing In Science & Engineering* **9** no. 3, (2007) 90–95.

[24] P. Z. Skands *et al.*, "SUSY Les Houches accord: Interfacing SUSY spectrum calculators, decay packages, and event generators," JHEP **0407** (2004) 036 [hep-ph/0311123].