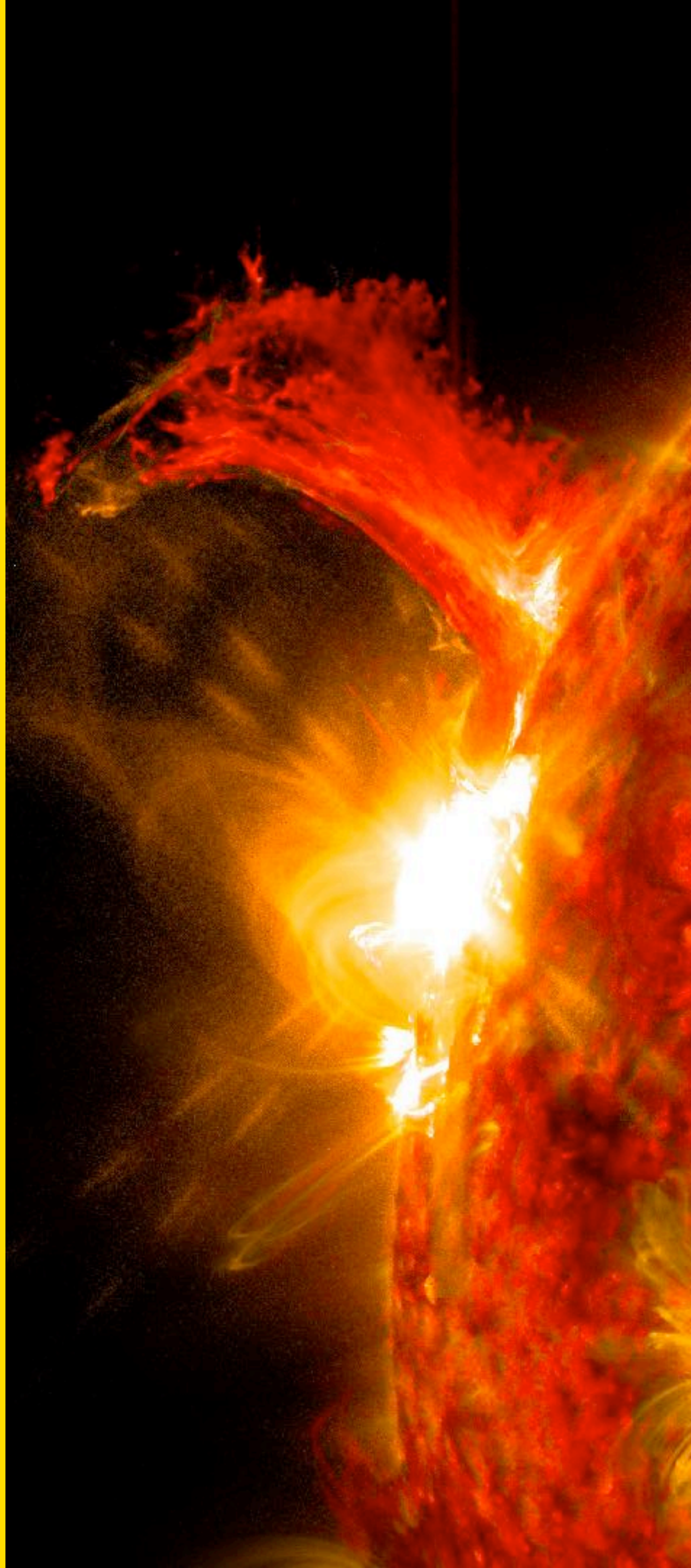
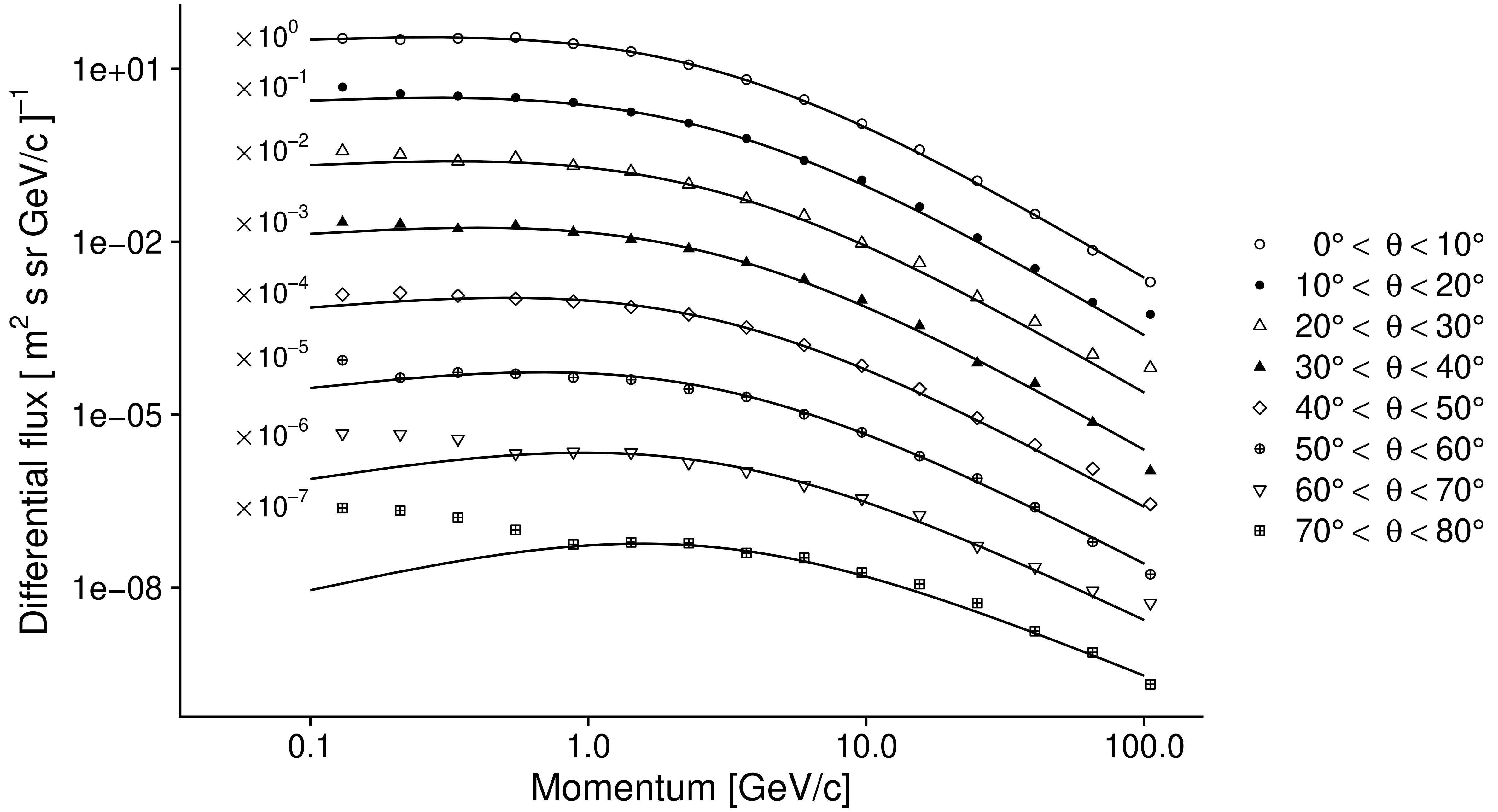
Update on EcoMug cosmic-ray muon generator

D. Pagano, G. Bonomi, A. Donzella, A. Zenoni, N. Zurlo

UNIVERSITÀ DEGLI STUDI DI BRESCIA & INFN PAVIA



What is EcoMug?



- Parametric cosmic muon generator
  - based on experimental data (Bonechi et al.)

- Differential flux $J \equiv J(t, p, \theta, \phi) = \dfrac{dN}{dt \cdot dp \cdot d\Omega \cdot dS_n}$

parametrized as

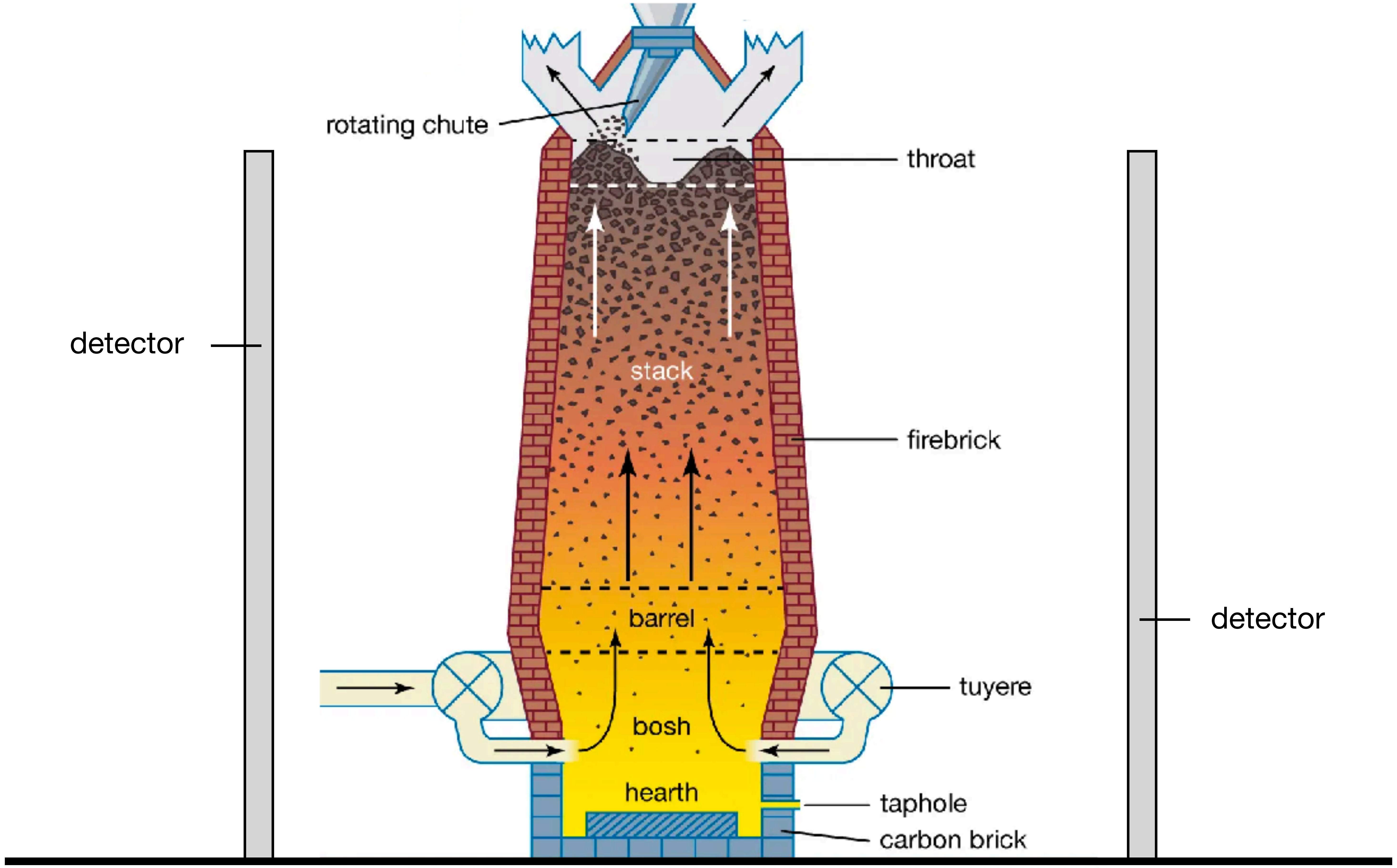
$$J = \left[1600 \cdot \left(\frac{p}{p_0} + 2.68\right)^{-3.175} \cdot \left(\frac{p}{p_0}\right)^{0.279}\right] \cdot (\cos\theta)^n \cdot \frac{1}{\text{m}^2 \cdot \text{s} \cdot \text{sr} \cdot \text{GeV/c}}$$

, with $n(p) = \max\left[0.1,\ 2.856 - 0.655 \cdot \ln\left(\frac{p}{p_0}\right)\right]$

- Several tools already available (MCEq, CRY, CMSCGEN, muTeV, ...) why a new generator?

Why EcoMug?

flat generation surface



Why EcoMug?


flat generation surface
rotating chute
throat
detector
stack
firebrick
barrel
detector
tuyere
bosh
hearth
taphole
carbon brick

Why EcoMug?

cylindrical generation surface

What is EcoMug?

- In previous study case a cylindrical surface would highly increase the generation efficiency

- For other cases a half-spherical surface could be optimal choice

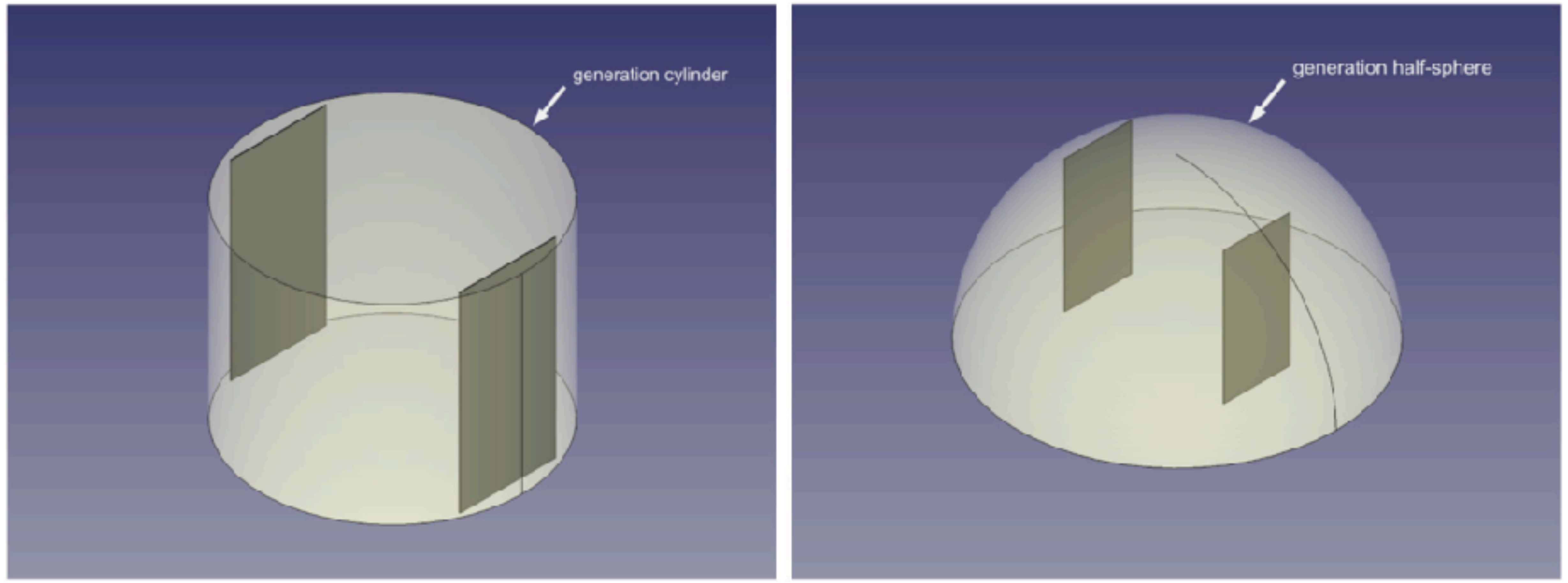
- EcoMug is a C++11 header only library which addresses this problem

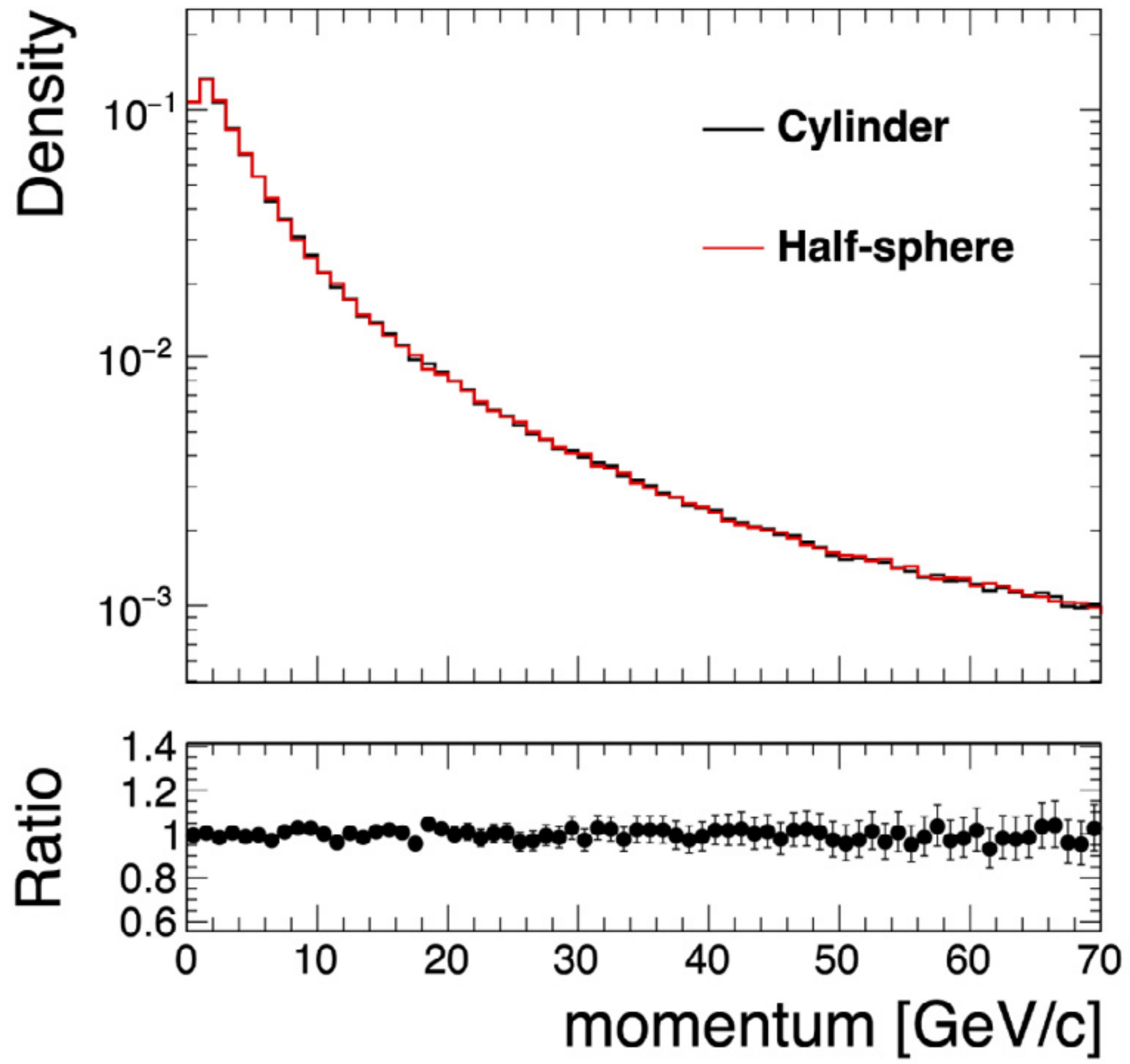
- It allows generating from different surfaces (plane, cylinder and half-sphere), while **keeping the correct angular and momentum distributions of muons**

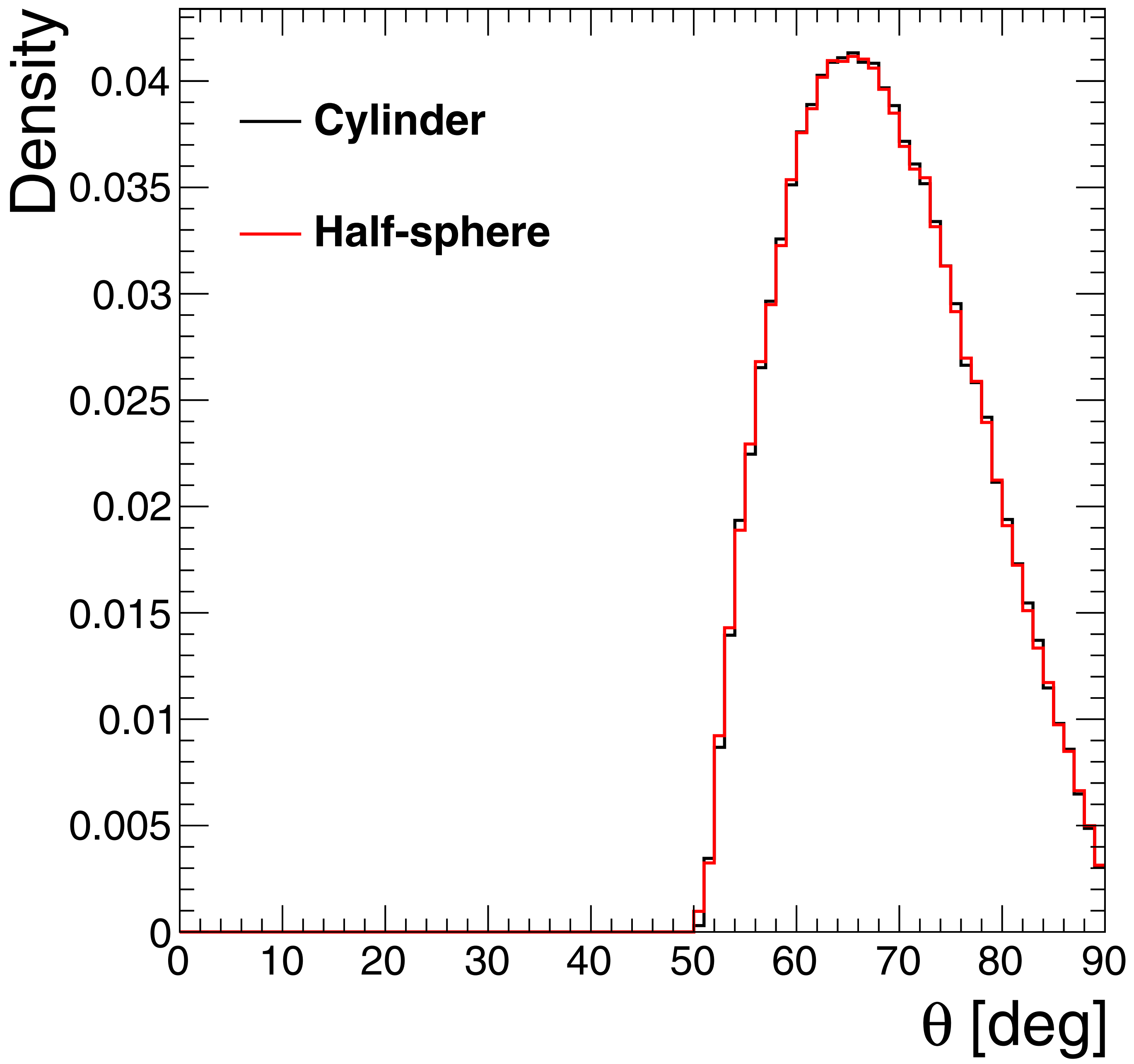
- Additionally, the user can constraint the generation (momentum, zenith angle and azimuthal angle) to further reduce the number of useless generated tracks

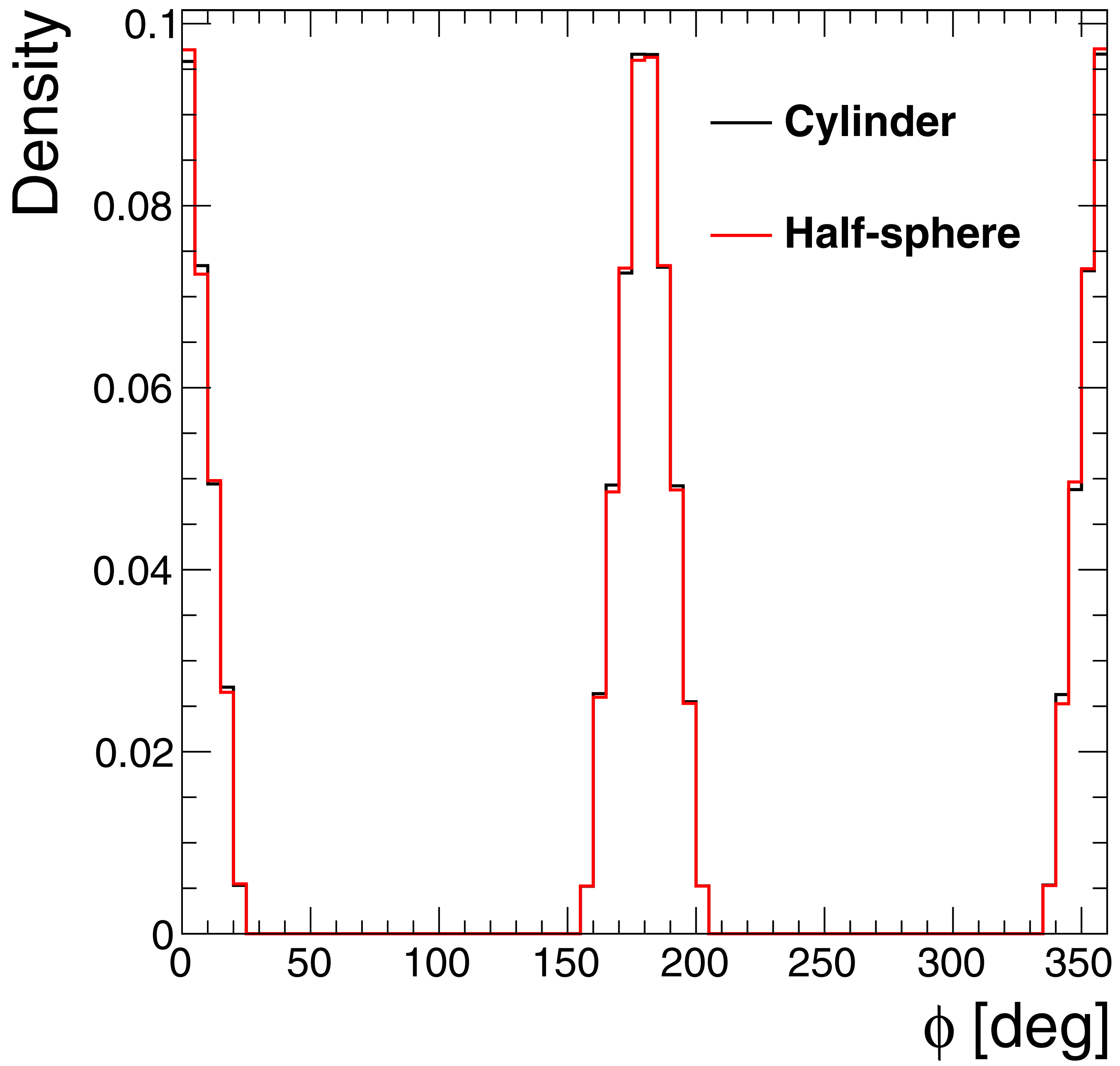
Equivalence between generation surfaces

- Generations from different surfaces are equivalent, provided the full coverage of the geometrical acceptance of the detection system is granted







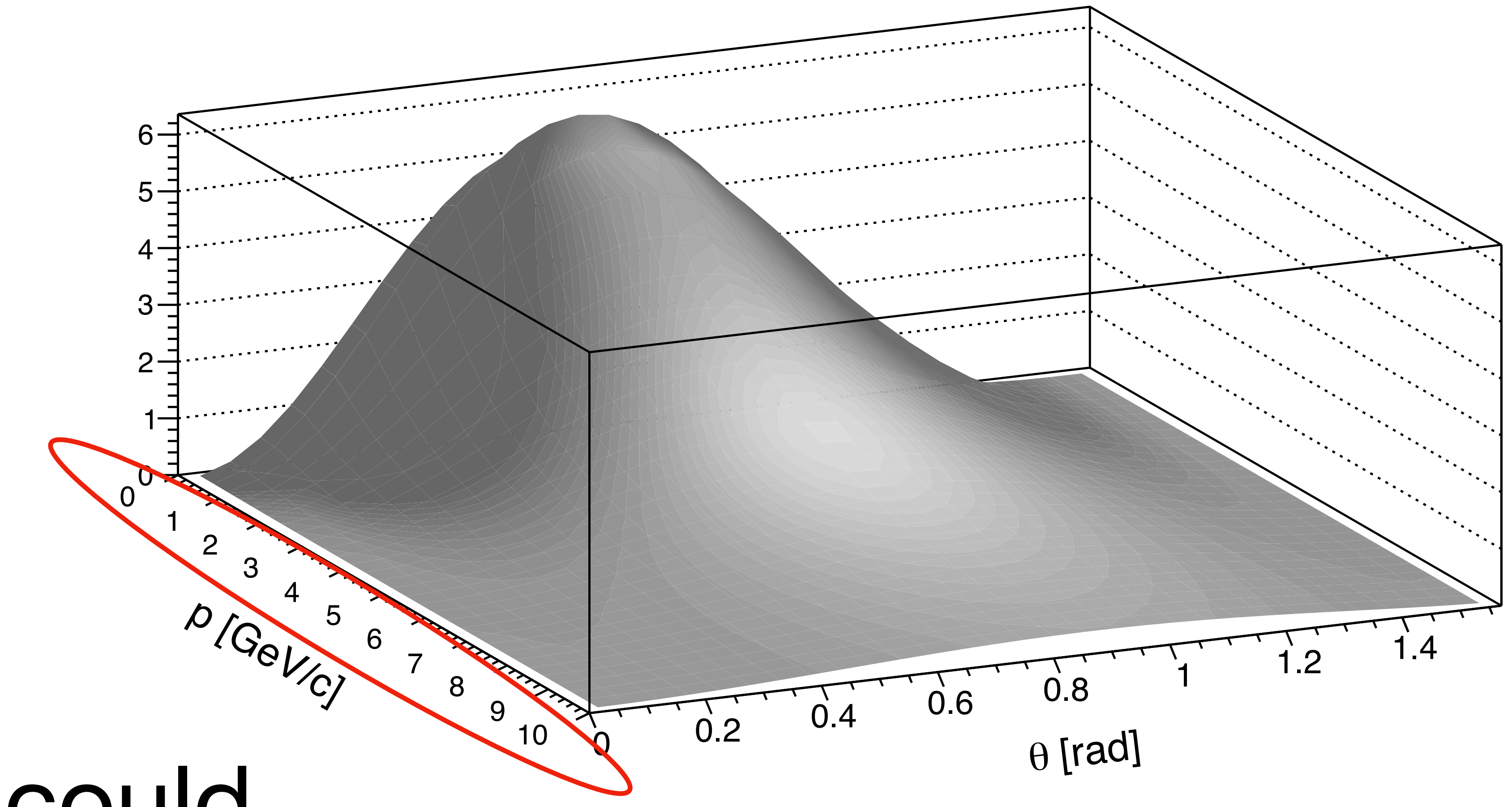


What's new in v2?

Under-the-hood improvements

Plane: $(x_0, y_0, \theta, \phi, p)$
Cylinder: $(\theta_0, z_0, \theta, \phi, p)$
Hemisphere: $(\theta_0, \phi_0, \theta, \phi, p)$

- Generation of a muon requires 5 parameters in EcoMug
- Depending on the surface, up to 4 not-independent variables
  - Acceptance-rejection method is a simple solution but extremely inefficient for J



- By properly factorizing the differential flux for all surfaces, we could use a hybrid approach based on both inverse transform and acceptance-rejection methods
  - **This was further improved in version 2**

D. Pagano and L. Sostero (2022) 10.1016/j.softx.2022.101083

- Other under-the-hood improvements include new methods for MC integration and an improved code for the metaheuristic optimization (internally used for the generation of muons)
- Also new: copy constructor for the EcoMug class, new method for retrieving the generation surface area (even when constrained), ...

Units

- EcoMug now includes a coherent system of units under the namespace EMUnits
  - default units:
    - lengths/areas: meter (m) - square meter (m2)
    - time: second (s)
    - energy/momentum: Giga electron Volt (GeV)
    - angles: radian (rad)

Example

```
EcoMug genPlane;
genPlane.SetUseSky();
genPlane.SetSkySize({{200.*EMUnits::cm, 200.*EMUnits::cm}});
genPlane.SetSkyCenterPosition({0., 0., 1.*EMUnits::mm});

double genArea = genPlane.GetGenSurfaceArea()/EMUnits::m2;
double genRate = genPlane.GetAverageGenRate()/EMUnits::hertz*EMUnits::m2;
```

```
namespace EMUnits {
  // Default units:
  // meter              (m)
  // second             (s)
  // Giga electron Volt (GeV)
  // radian             (rad)

  // Lengths and areas
  static const double m     = 1.;
  static const double cm    = 1.e-2*m;
  static const double mm    = 1.e-3*m;
  static const double km    = 1000.*m;
  static const double mm2   = mm*mm;
  static const double cm2   = cm*cm;
  static const double m2    = m*m;
  static const double km2   = km*km;

  // Angles
  static const double rad   = 1.;
  static const double mrad  = 1.e-3*rad;
  static const double deg   = (M_PI/180.0)*rad;

  // Time
  static const double s     = 1.;
  static const double ms    = 1.e-3*s;
  static const double us    = 1.e-6*s;
  static const double ns    = 1.e-9*s;
  static const double min   = 60.*s;
  static const double hour  = 60.*min;
  static const double day   = 24.*hour;
  static const double hertz = 1./s;

  // Energy/momentum
  static const double GeV = 1.;
  static const double MeV = 1.e-3*GeV;
  static const double keV = 1.e-3*MeV;
  static const double TeV = 1.e+6*MeV;
  static const double  eV = 1.e-6*MeV;
};
```

Logger

- EcoMug now includes a proper logger to handle the printout to screen
  - 4 levels of reporting `enum TLogLevel {ERROR, WARNING, INFO, DEBUG};`

Output

```
[EcoMug v2.0] [WARNING in EMMultiGen]: Expected exactly 1 instance with PID = 0, but 2 were provided.
```

version ↑ level ↑ class ↑ message ↑

- The reporting threshold can be set globally as in the example on the right
  - Default: `WARNING`

Example

```
EMLog::ReportingLevel = EMLog::TLogLevel::ERROR;
```

Time estimation

- EcoMug now allows to estimate the rate and time to collect a given number of muons
  - It also handles those cases where the user has constrained the generations of muons (for example by cutting on *p*), as well as the generation geometry

- The user can specify the average expected rate (Hz/m^2) (method: **`SetHorizontalRate`**) to take into account the effect of altitude, etc...
  - Default value is $129\ Hz/m^2$

- While the rate and time estimation also works with custom definitions of the flux, it is up to the user to define a properly normalized J
  - **`SetHorizontalRate`** does not work in this case (see example in the next slide)

Time estimation

- Example on how to use it (included in TestSuite.C)

```
EcoMug genPlane;
genPlane.SetUseSky();
genPlane.SetSkySize({{200.*EMUnits::cm, 200.*EMUnits::cm}});
genPlane.SetSkyCenterPosition({0., 0., 1.*EMUnits::mm});

EcoMug genHSphere;
genHSphere.SetUseHSphere();
genHSphere.SetHSphereRadius(200*EMUnits::cm);
genHSphere.SetHSphereCenterPosition({0., 0., 0.});

TVector3 P1 = {-50.*EMUnits::cm, -50.*EMUnits::cm, 0.};
TVector3 P2 = { 50.*EMUnits::cm, -50.*EMUnits::cm, 0.};
TVector3 P3 = { 50.*EMUnits::cm,  50.*EMUnits::cm, 0.};
PlaneDet detector(P1, P2, P3);
```

- We want to compute the time necessary to detect n events on a horizontal surface as generated from a flat surface and a half-spherical surface

```
while (n_good_events < number_of_events) {
    genPlane.Generate();

    ...

    if (!detector.IsCrossed(muon_origin, muon_p)) continue;
    n_good_events++;
}
```

```
while (n_good_events < number_of_events) {
    genHSphere.Generate();

    ...

    if (!detector.IsCrossed(muon_origin, muon_p)) continue;
    n_good_events++;
}
```

Time estimation

```
cout << "\n--- Generation from horizontal plane ---" << endl;
cout << "number of generated muons                   = " << n_gen_events << endl;
cout << "number of muons through the detector        = " << n_good_events << endl;
cout << "number of gen muons/generation surface [m2] = " << n_gen_events/(genPlane.GetGenSurfaceArea()/EMUnits::m2) << endl;
cout << "Estimated time [s]                          = " << genPlane.GetEstimatedTime(n_gen_events) << endl;
```

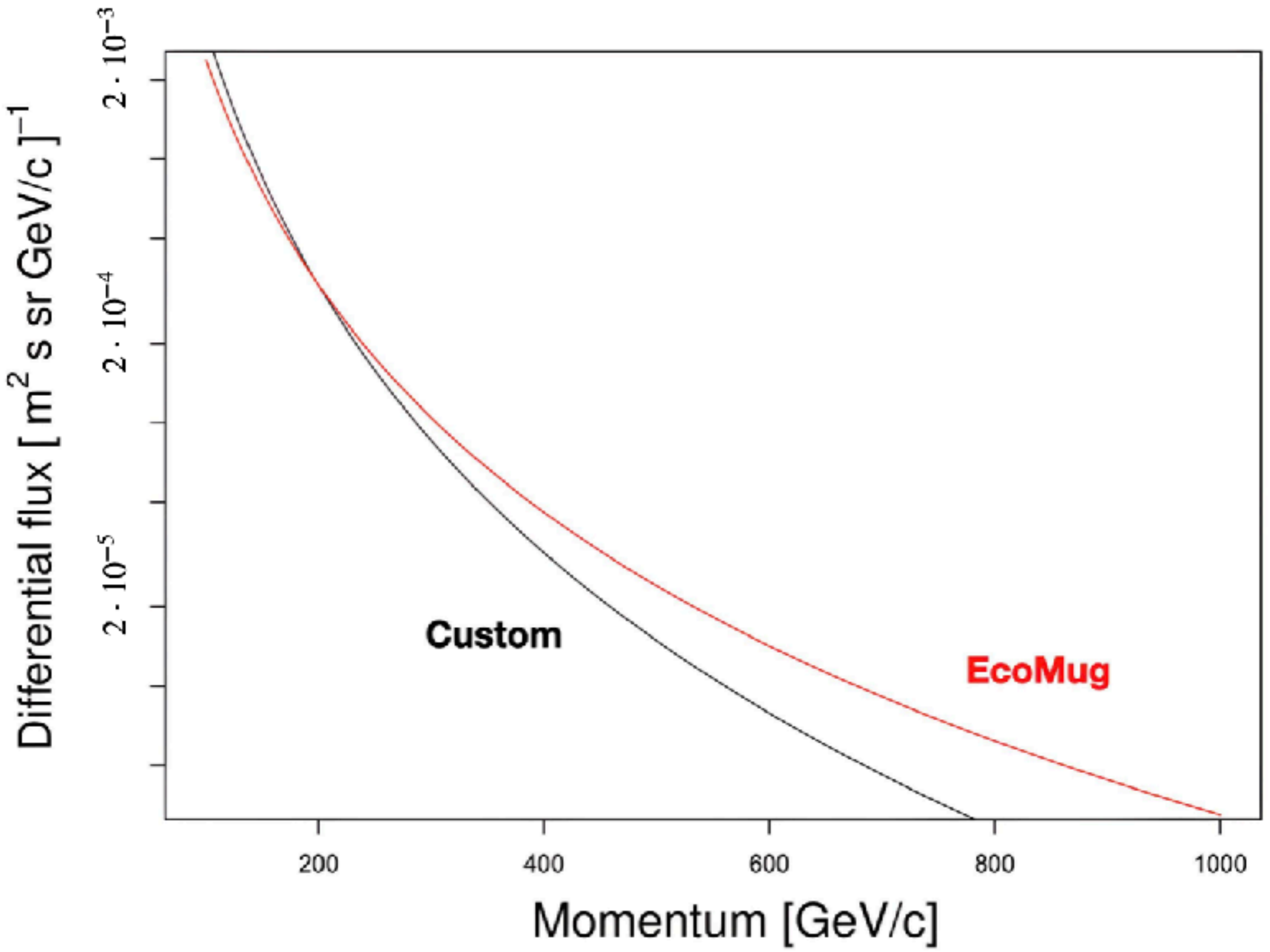
```
cout << "\n--- Generation from half-sphere ---" << endl;
cout << "number of generated muons                   = " << n_gen_events << endl;
cout << "number of muons through the detector        = " << n_good_events << endl;
cout << "number of gen muons/generation surface [m2] = " << n_gen_events/(genHSphere.GetGenSurfaceArea()/EMUnits::m2) << endl;
cout << "Estimated time [s]                          = " << genHSphere.GetEstimatedTime(n_gen_events) << endl;
```

```
% root -l TestSuite.C'(1,10000)'
Processing TestSuite.C(1,10000)...

--- Generation from horizontal plane ---
number of generated muons                   = 40351
number of muons through the detector        = 10000
number of gen muons/generation surface [m2] = 10087.8
Estimated time [s]                          = 77.77

--- Generation from half-sphere ---
number of generated muons                   = 145278
number of muons through the detector        = 10000
number of gen muons/generation surface [m2] = 5780.43
Estimated time [s]                          = 76.88

horizontal to half-spherical rate           = 1.73
```

Time estimation

Example included in TestSuite.C

```
double J(double p, double theta) {
  double A = 1400*pow(p, -2.7);
  double B = 1. / (1. + 1.1*p*cos(theta)/115.);
  double C = 0.054 / (1. + 1.1*p*cos(theta)/850.);
  return A*(B+C);
}

...

EcoMug genPlane;
genPlane.SetUseSky();
genPlane.SetSkySize({{200.*EMUnits::cm, 200.*EMUnits::cm}});
genPlane.SetMinimumMomentum(100.*EMUnits::GeV);
genPlane.SetMaximumMomentum(1000.*EMUnits::GeV);

EcoMug genCylinder(genPlane);
genCylinder.SetUseCylinder();
genCylinder.SetCylinderRadius(100.*EMUnits::cm);
genCylinder.SetCylinderHeight(10.*EMUnits::m);

EcoMug genHSphere(genPlane);
genHSphere.SetUseHSphere();
genHSphere.SetHSphereRadius(300*EMUnits::cm);

EcoMug genCustomSky(genPlane);
genCustomSky.SetDifferentialFlux(&J);

EcoMug genCustomCylinder(genCylinder);
genCustomCylinder.SetDifferentialFlux(&J);

EcoMug genCustomHSphere(genHSphere);
genCustomHSphere.SetDifferentialFlux(&J);

double rateSky, rateCyl, rateHS, rateCustomSky, rateCustomCylinder, rateCustomHSphere;
double errorSky, errorCyl, errorHS, errorCustomSky, errorCustomCylinder, errorCustomHSphere;
genPlane.GetAverageGenRateAndError(rateSky, errorSky, 1e7);
genCylinder.GetAverageGenRateAndError(rateCyl, errorCyl, 1e7);
genHSphere.GetAverageGenRateAndError(rateHS, errorHS, 1e7);
genCustomSky.GetAverageGenRateAndError(rateCustomSky, errorCustomSky, 1e7);
genCustomCylinder.GetAverageGenRateAndError(rateCustomCylinder, errorCustomCylinder, 1e7);
genCustomHSphere.GetAverageGenRateAndError(rateCustomHSphere, errorCustomHSphere, 1e7);
```




```
% root -l TestSuite.C'(2,10000)'
Processing TestSuite.C(2,10000)...

rate sky                     = 0.380 +- 0.0003
rate cylinder                = 0.178 +- 0.0003
rate half-sphere             = 0.276 +- 0.0003
rate custom J sky            = 0.551 +- 0.0005
rate custom J cylinder       = 0.341 +- 0.0006
rate custom J half-sphere = 0.461 +- 0.0007
```

Deal with background

*https://pdg.lbl.gov/2007/reviews/montecarlorpp.pdf

- EcoMug now offers a new class `EMMultiGen` to also handle background
  - Requires a `EcoMug` instance for the signal and one or more instances for the background
  - The user has to specify the differential flux (even unnormalized), the PID (Monte Carlo particle numbering scheme*) and the relative weight (w.r.t. signal) for all backgrounds

- The use of `EMMultiGen` is identical to `EcoMug`
  - The following methods to generate and access track parameters are available in both classes →
  - In addition to them, the user also access to the PID of generated track (to distinguish between the signal and different possible backgrounds) →

```
void Generate()

const std::array<double, 3>& GetGenerationPosition()

double GetGenerationMomentum()

void GetGenerationMomentum(std::array<double, 3>&)

double GetGenerationTheta()

double GetGenerationPhi()
```

```
int GetPID()
```

Deal with background

```
EcoMug muonGen;
muonGen.SetUseSky();
muonGen.SetSkySize({{200.*EMUnits::cm, 200.*EMUnits::cm}});
muonGen.SetSkyCenterPosition({0., 0., 1.*EMUnits::mm});

EcoMug electronGen(muonGen);
electronGen.SetDifferentialFlux(&J);

EcoMug positronsGen(muonGen);
electronGen.SetDifferentialFlux(&J);

EMMultiGen genSuite(muonGen, {electronGen, positronsGen});
genSuite.SetBckWeights({0.2, 0.1});
genSuite.SetBckPID({11, -11});

map<int, int> counts;
for (auto i = 0; i < number_of_events; ++i) {
  genSuite.Generate();
  counts[genSuite.GetPID()]++;
}
```

```
% root -l TestSuite.C'(4,10000)'
Processing TestSuite.C(4,10000)...

  PID   counts      ratio
  -13     4483    (0.581)
  -11      738    (0.0957)
   11     1551    (0.201)
   13     3228    (0.419)
```

ratio → **wrt to the signal** $(\mu^- + \mu^+)$

Documentation

dr4kan.github.io/EcoMug

EcoMug 2.0.0
Efficient COsmic MUon Generator

Search

- EcoMug
  - EcoMug: Efficient COsmic MUon Generator
    - Basic Usage
    - More Advanced Usage
    - Rate and time estimation
    - Deal with background
  - Classes
  - Files

EcoMug: Efficient COsmic MUon Generator

EcoMug is a header-only C++11 library for the generation of cosmic ray (CR) muons, based on a parametrization of experimental data. Unlike other tools, EcoMug gives the possibility of generating from different surfaces (plane, cylinder and half-sphere), while keeping the correct angular and momentum distribution of generated tracks. EcoMug also allows the generation of CR muons according to user-defined parametrizations of their differential flux.

If you use, or want to refer to, EcoMug please cite the following paper:

> Pagano, D., Bonomi, G., Donzella, A., Zenoni, A., Zumerle, G., & Zurlo, N. (2021). EcoMug: An Efficient COsmic MUon Generator for cosmic-ray muon applications. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 1014, 165732.

Latest release: EcoMug v2.0.0

Basic Usage

The use of the library requires the initialization of the `EcoMug` class, the choice of the generation method, and the definition of the size and position of the generation surface. Once the setup of the instance of the `EcoMug` class is done, the generation of a cosmic-ray muon can be invoked with the method `Generate()`, which will compute its position, direction, momentum, and charge. All these quantities can be accessed with the methods `GetGenerationPosition()`, `GetGenerationTheta()`, `GetGenerationPhi()`, `GetGenerationMomentum()`, and `GetCharge()`, as shown in the examples below. The charge for generated muons takes into account the excess of positive muons over negative ones, assuming a constant charge ratio (see the above mentioned paper for more details). Angles are in radians, momentum is in GeV/c, whereas the unit of measure of the position is arbitrary and depends on the choice done in the simulation code where EcoMug is used.

Plane-based generation

```
EcoMug gen; // initialization of the class
gen.SetUseSky(); // plane surface generation
gen.SetSkySize({{10., 10.}}); // x and y size of the plane
```

Conclusions

- EcoMug is a C++11 header only library to generate cosmic-ray muons from different surfaces, while keeping the correct angular and momentum distributions (based on experimental data)
    - Several applications in muography could benefit from this
- Version 2.0 offers new improvements:
    - Many under-the-hood improvements
    - A coherent system of units
    - A a proper logger to handle the printout to screen
    - Rate and time estimation for all surfaces (also in presence of user-defined cuts)
    - A new class to also handle background generation
    - A better documentation

If you have suggestions, issues, or you want to contribute go to https://github.com/dr4kan/EcoMug