Pythia 8: Physics and usage

Saariselkä Midsummer School 2024

Ilkka Helenius

June 27, 2024

JYVÄSKYLÄN YLIOPISTO
UNIVERSITY OF JYVÄSKYLÄ

Research Council of Finland

Centre of Excellence in Quark Matter

Preparations

- Download the Docker image:

```
$ docker pull hepstore/rivet-pythia
```

- Start the container and set up the current directory as a "host" directory (This way you can open and modify files as they would be local files):

```
$ docker run -v $PWD:/host -it --rm hepstore/rivet-pythia
```

- Copy an example to your `/host` folder:

```
$ cd /host
$ cp ../usr/local/share/Pythia8/examples/main01.cc .
$ cp ../usr/local/share/Pythia8/examples/Makefile* .
```

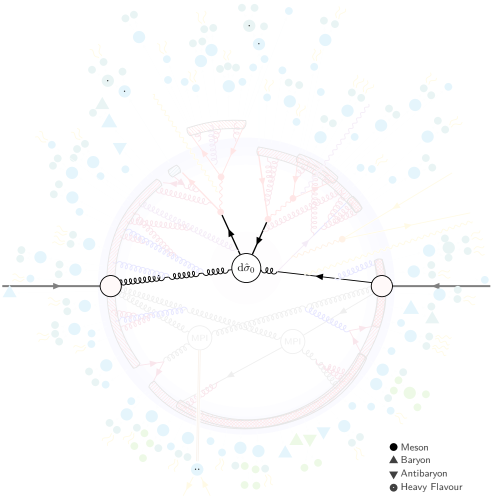
- Compile and run the example:

```
$ make main01 && ./main01
```

Physics modelled within Pythia 8 event generator

Classify event generation in terms of "hardness"

1. Hard Process (here $t\bar{t}$)

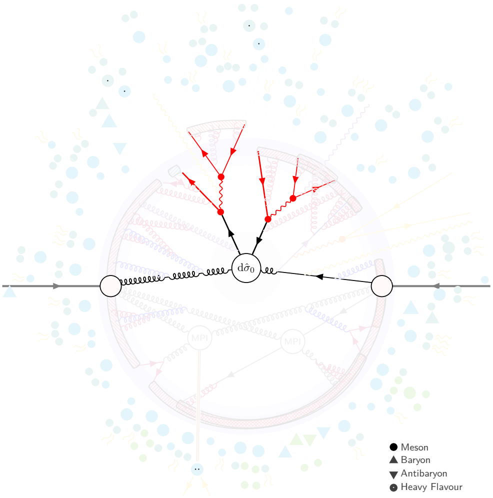


[figure by P. Skands]

Physics modelled within Pythia 8 event generator

Classify event generation in terms of "hardness"

1. Hard Process (here $t\bar{t}$)
2. Resonance decays ($t, Z, \ldots$)

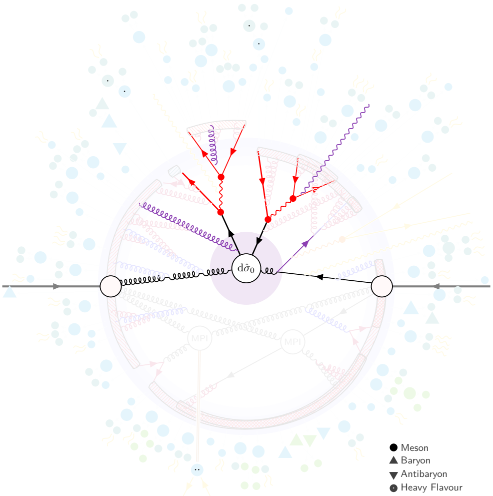


[figure by P. Skands]

Physics modelled within Pythia 8 event generator

Classify event generation in terms of "hardness"

1. Hard Process (here $t\bar{t}$)
2. Resonance decays (t, Z, . . .)
3. Matching, Merging and matrix-element corrections

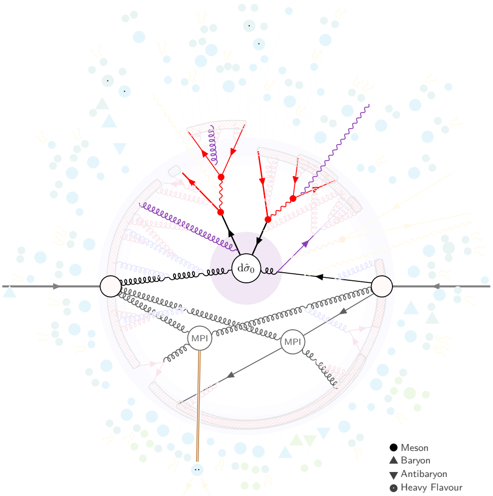


[figure by P. Skands]

Physics modelled within Pythia 8 event generator

Classify event generation in terms of "hardness"

1. Hard Process (here $t\bar{t}$)
2. Resonance decays ($t, Z, \ldots$)
3. Matching, Merging and matrix-element corrections
4. Multiparton interactions

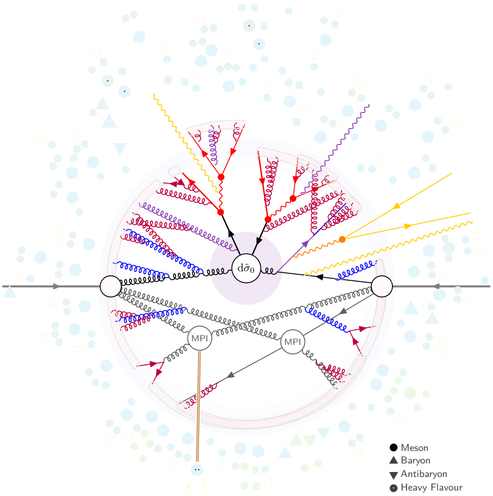


[figure by P. Skands]

Physics modelled within Pythia 8 event generator

Classify event generation in terms of "hardness"

1. Hard Process (here $t\bar{t}$)
2. Resonance decays ($t, Z, \ldots$)
3. Matching, Merging and matrix-element corrections
4. Multiparton interactions
5. Parton showers: ISR, FSR, QED, Weak

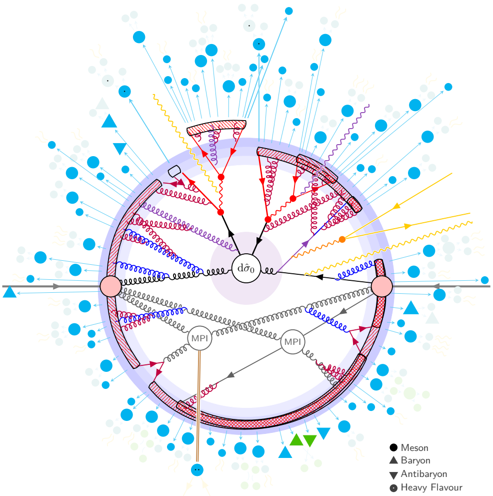


[figure by P. Skands]

Physics modelled within Pythia 8 event generator

Classify event generation in terms of "hardness"

1. Hard Process (here $t\bar{t}$)
2. Resonance decays ($t, Z, \ldots$)
3. Matching, Merging and matrix-element corrections
4. Multiparton interactions
5. Parton showers: ISR, FSR, QED, Weak
6. Hadronization, Beam remnants

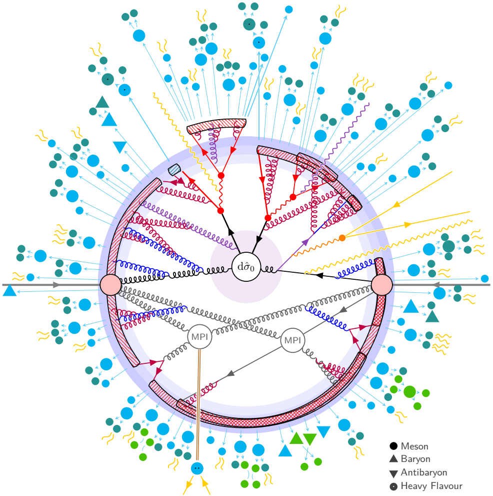


[figure by P. Skands]

Physics modelled within Pythia 8 event generator

Classify event generation in terms of "hardness"

1. Hard Process (here $t\bar{t}$)
2. Resonance decays ($t, Z, \ldots$)
3. Matching, Merging and matrix-element corrections
4. Multiparton interactions
5. Parton showers: ISR, FSR, QED, Weak
6. Hadronization, Beam remnants
7. Decays, Rescattering



[figure by P. Skands]

Outline

Lecture 1: `ProcessLevel`

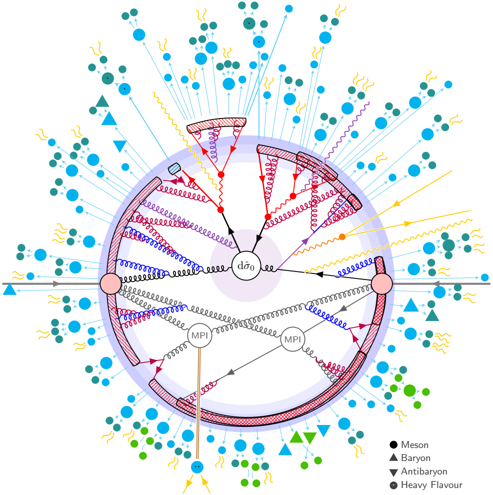
- History of Pythia
- Monte Carlo techniques
- Hard-process sampling

Lecture 2: `PartonLevel`

- Multiparton interactions
- Parton showers

Lecture 3: `HadronLevel`

- Hadronization
- Beam configurations

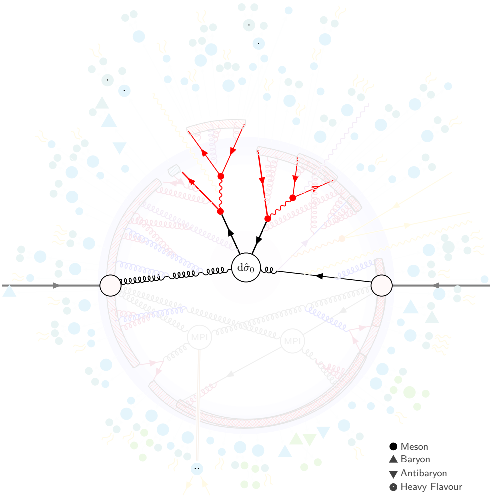


[figure by P. Skands]

Outline

Lecture 1: `ProcessLevel`

- History of Pythia
- Monte Carlo techniques
- Hard-process sampling



[figure by P. Skands]

History of Pythia

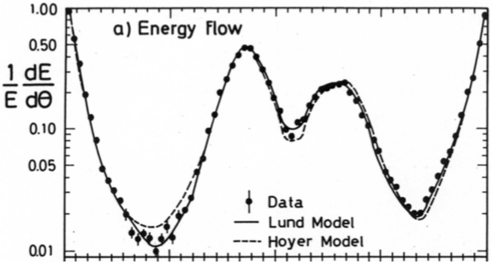
…a local woman, the Pythia, would sit on a tripod and inhale the vapours. Her more-or-less incoherent screams would be interpreted by the local priesthood, and often presented as poems in perfect hexameter. Some of these became famous for their ambiguity, and the disastrous consequences of a misinterpretation.

Similarly the PYTHIA code is intended to provide you with answers to many questions you may have about high-energy collisions, but it is then up to you to use sane judgement when you interpret these answers.

Long history of Pythia (in short)

The PYTHIA Event Generator: Past, Present and Future by Torbjörn Sjöstrand, Comput. Phys. Comm. 246 (2020) 106910 arXiv:1907.09874v1 [hep-ph]

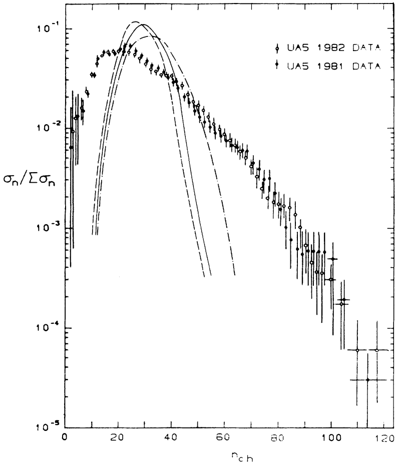
- 1978: Jetset string hadronization
- 1982: Pythia for p-p collisions
- 1984: Final-state parton shower
- 1985: Initial-state parton shower
- 1985: Multi-parton interactions
- 1997: Jetset merged into Pythia 6
- 2005: Pythia 8.1 in `C++`
- 2014: Pythia 8.2 (drop 6.4 support)
- 2019: Pythia 8.3 with `C++11`



Long history of Pythia (in short)

The PYTHIA Event Generator: Past, Present and Future by Torbjörn Sjöstrand, Comput. Phys. Comm. 246 (2020) 106910 arXiv:1907.09874v1 [hep-ph]

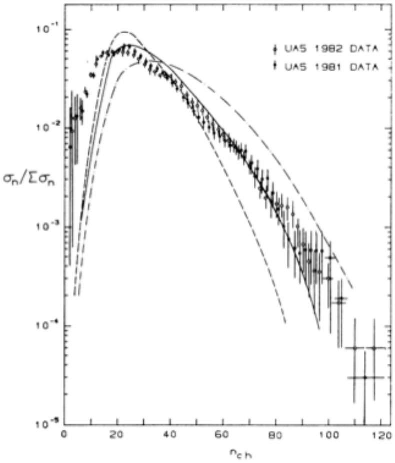
- 1978: Jetset string hadronization
- 1982: Pythia for p-p collisions
- 1984: Final-state parton shower
- 1985: Initial-state parton shower
- 1985: Multi-parton interactions
- 1997: Jetset merged into Pythia 6
- 2005: Pythia 8.1 in `C++`
- 2014: Pythia 8.2 (drop 6.4 support)
- 2019: Pythia 8.3 with `C++11`



Long history of Pythia (in short)

The PYTHIA Event Generator: Past, Present and Future by Torbjörn Sjöstrand, Comput. Phys. Comm. 246 (2020) 106910 arXiv:1907.09874v1 [hep-ph]

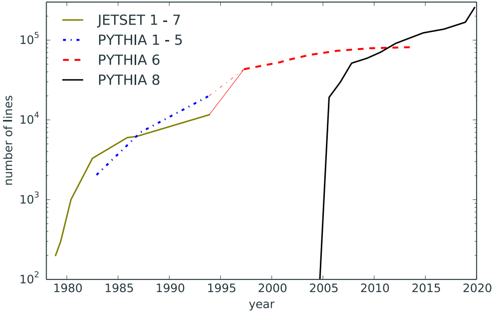
- 1978: Jetset string hadronization
- 1982: Pythia for p-p collisions
- 1984: Final-state parton shower
- 1985: Initial-state parton shower
- 1985: Multi-parton interactions
- 1997: Jetset merged into Pythia 6
- 2005: Pythia 8.1 in `C++`
- 2014: Pythia 8.2 (drop 6.4 support)
- 2019: Pythia 8.3 with `C++11`



Long history of Pythia (in short)

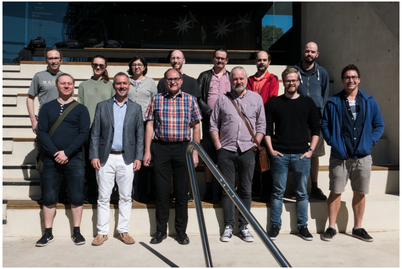
The PYTHIA Event Generator: Past, Present and Future by Torbjörn Sjöstrand, Comput. Phys. Comm. 246 (2020) 106910 arXiv:1907.09874v1 [hep-ph]

- 1978: Jetset string hadronization
- 1982: Pythia for p-p collisions
- 1984: Final-state parton shower
- 1985: Initial-state parton shower
- 1985: Multi-parton interactions
- 1997: Jetset merged into Pythia 6
- 2005: Pythia 8.1 in `C++`
- 2014: Pythia 8.2 (drop 6.4 support)
- 2019: Pythia 8.3 with `C++11`



Pythia Collaboration

- Javira Altmann (Monash University)
- Christian Bierlich (Lund University)
- Naomi Cooke (University of Glasgow)
- Nishita Desai (TIFR, Mumbai)
- Leif Gellersen (Lund University)
- Ilkka Helenius (University of Jyväskylä)
- Philip Ilten (University of Cincinnati)
- Leif Lönnblad (Lund University)
- Stephen Mrenna (Fermilab)
- Christian Preuss (University of Wuppertal)
- Torbjörn Sjöstrand (Lund University)
- Peter Skands (Monash University/Oxford)
- Marius Utheim (University of Jyväskylä)
- Rob Verheyen (University College London)



[Pythia meeting in Monash 2019]

Latest release 8.312 (May 2024)

[SciPost Phys. Codebases 8-r8.3 (2022)]

`https://pythia.org`

`https://gitlab.com/Pythia8/releases`

`authors@pythia.org`

Monte Carlo methods

Monte Carlo simulations

Method

- Based on numerical modelling and statistics
- Sample “events” from known distributions using (pseudo-)random numbers

Useful when

- Distributions are difficult to handle on pen and paper
- Multi-dimensional distributions

Monte Carlo techniques I

Analytical solution

- $f(x)$ a one-dimensional distribution
- When $x_{\text{min}} < x < x_{\text{max}}$ we have $0 < R < 1$ such that

$$\int_{x_{\text{min}}}^{x} f(x')\mathrm{d}x' = R \int_{x_{\text{min}}}^{x_{\text{max}}} f(x')\mathrm{d}x'$$

- If integral of f ($F(x)$) is known and invertible ($F^{-1}(x)$)

$$x = F^{-1}(F(x_{\text{min}}) + R(F(x_{\text{max}}) - F(x_{\text{min}})))$$

R is a random number $\in [0, 1[$

Analytical solution

- $f(x)$ a one-dimensional distribution
- When $x_{\text{min}} < x < x_{\text{max}}$ we have $0 < R < 1$ such that

$$\int_{x_{\text{min}}}^{x} f(x')\mathrm{d}x' = R \int_{x_{\text{min}}}^{x_{\text{max}}} f(x')\mathrm{d}x'$$

- If integral of f ($F(x)$) is known and invertible ($F^{-1}(x)$)

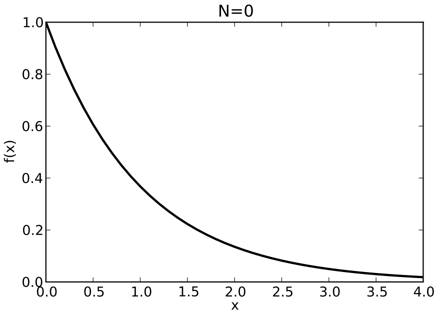
$$x = F^{-1}(F(x_{\text{min}}) + R(F(x_{\text{max}}) - F(x_{\text{min}})))$$

R is a random number $\in [0, 1[$

Example: $f(x) = e^{-x}, \quad 0 < x < \infty$

- $F(x) = 1 - e^{-x}$
- $F^{-1}(x) = -\log(1 - x)$

$\Rightarrow$ $x = -\log(R)$



Monte Carlo techniques I

Analytical solution

- $f(x)$ a one-dimensional distribution
- When $x_{\text{min}} < x < x_{\text{max}}$ we have $0 < R < 1$ such that

$$\int_{x_{\text{min}}}^{x} f(x')dx' = R \int_{x_{\text{min}}}^{x_{\text{max}}} f(x')dx'$$

- If integral of f ($F(x)$) is known and invertible ($F^{-1}(x)$)

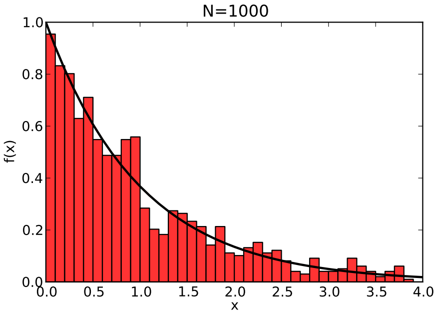
$$x = F^{-1}(F(x_{\text{min}}) + R(F(x_{\text{max}}) - F(x_{\text{min}})))$$

R is a random number $\in [0, 1[$

Example: $f(x) = e^{-x}, \quad 0 < x < \infty$

- $F(x) = 1 - e^{-x}$
- $F^{-1}(x) = -\log(1 - x)$

$\Rightarrow$ $x = -\log(R)$



Monte Carlo techniques I

Analytical solution

- $f(x)$ a one-dimensional distribution
- When $x_{\text{min}} < x < x_{\text{max}}$ we have $0 < R < 1$ such that

$$\int_{x_{\text{min}}}^{x} f(x')dx' = R \int_{x_{\text{min}}}^{x_{\text{max}}} f(x')dx'$$

- If integral of f ($F(x)$) is known and invertible ($F^{-1}(x)$)

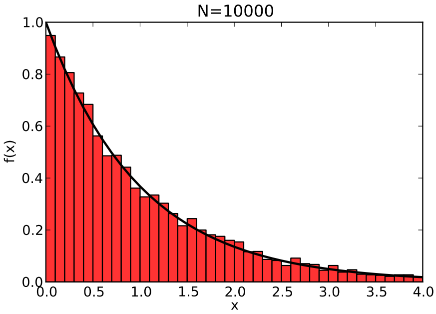
$$x = F^{-1}(F(x_{\text{min}}) + R(F(x_{\text{max}}) - F(x_{\text{min}})))$$

R is a random number $\in [0, 1[$

Example: $f(x) = e^{-x}, \quad 0 < x < \infty$

- $F(x) = 1 - e^{-x}$
- $F^{-1}(x) = -\log(1 - x)$

$\Rightarrow$ $x = -\log(R)$



Monte Carlo techniques I

Analytical solution

- $f(x)$ a one-dimensional distribution
- When $x_{\text{min}} < x < x_{\text{max}}$ we have $0 < R < 1$ such that

$$\int_{x_{\text{min}}}^{x} f(x')\text{d}x' = R \int_{x_{\text{min}}}^{x_{\text{max}}} f(x')\text{d}x'$$

- If integral of f ($F(x)$) is known and invertible ($F^{-1}(x)$)

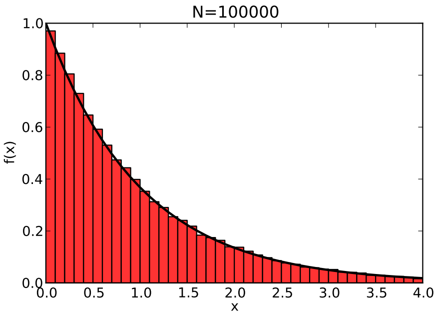
$$x = F^{-1}(F(x_{\text{min}}) + R(F(x_{\text{max}}) - F(x_{\text{min}})))$$

R is a random number $\in [0, 1[$

Example: $f(x) = e^{-x}, \quad 0 < x < \infty$

- $F(x) = 1 - e^{-x}$
- $F^{-1}(x) = -\log(1 - x)$

$\Rightarrow$ $x = -\log(R)$



Monte Carlo techniques II

Hit-and-miss

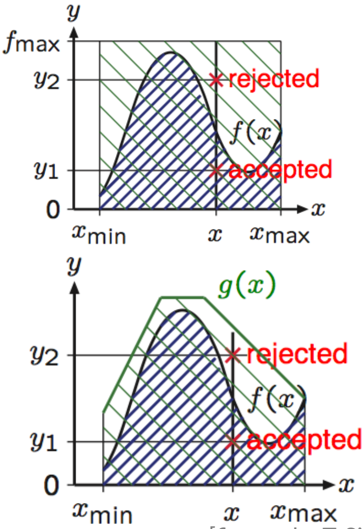
- Let $f(x) \leq f_{\max}$ in $x \in [x_{\min}, x_{\max}[$

1. Sample $x = x_{\min} + R_1\,(x_{\max} x_{\min})$
2. Sample $y = R_2\, f_{\max}$
3. while $y > f(x)$ cycle to 1.

Importance sampling

- Pick $g(x)$ such that $f(x) < g(x)$ in $x \in [x_{\min}, x_{\max}[$, integral of $g(x)$ $(G(x))$ known and $G(x)^{-1}$ simple

1. Sample x from $g(x)$ (analytic)
2. Sample $y = R\,g(x)$
3. while $y > f(x)$ cycle to 1.

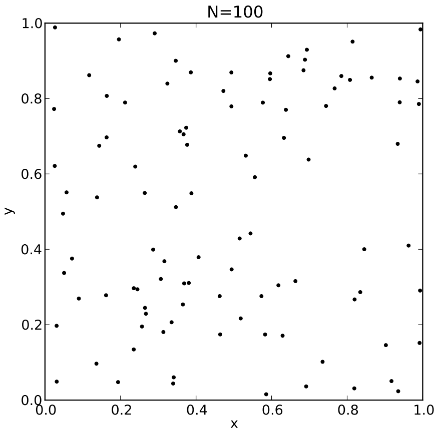


[figures by T. Sjöstrand]

Hit and miss example: Calculate value for π

Monte Carlo algorithm

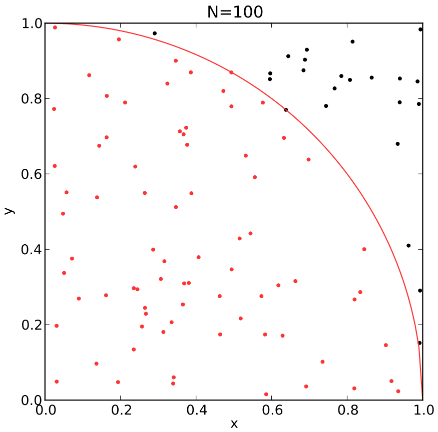
1. Sample N pairs of random numbers $(x, y), x, y \in [0, 1[$



Hit and miss example: Calculate value for π

Monte Carlo algorithm

1. Sample N pairs of random numbers $(x, y), x, y \in [0, 1[$
2. Calculate number of points within a quarter of circle $x^2 + y^2 < 1$

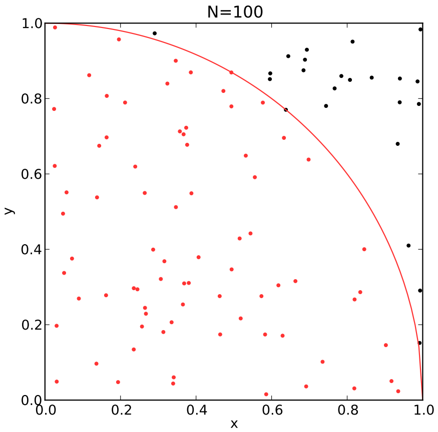


Hit and miss example: Calculate value for π

Monte Carlo algorithm

1. Sample N pairs of random numbers $(x, y), x, y \in [0, 1[$
2. Calculate number of points within a quarter of circle $x^2 + y^2 < 1$
3. Approximated values $= 4 * N_{\text{inside}} / N_{\text{tries}}$

N	value	error
100	3.08	0.0616

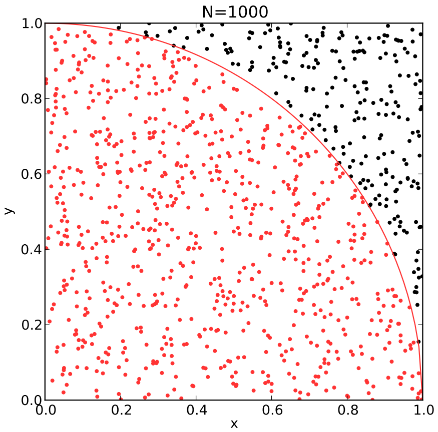


Hit and miss example: Calculate value for π

Monte Carlo algorithm

1. Sample N pairs of random numbers $(x, y), x, y \in [0, 1[$
2. Calculate number of points within a quarter of circle $x^2 + y^2 < 1$
3. Approximated values $= 4 * N_{\text{inside}}/N_{\text{tries}}$

N	value	error
100	3.08	0.0616
1000	3.092	0.0496

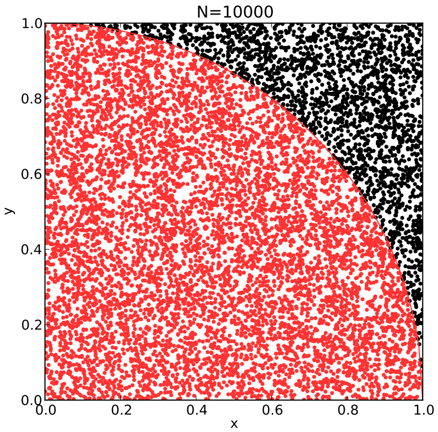


Hit and miss example: Calculate value for π

Monte Carlo algorithm

1. Sample N pairs of random numbers $(x, y), x, y \in [0, 1[$
2. Calculate number of points within a quarter of circle $x^2 + y^2 < 1$
3. Approximated values $= 4 * N_{\text{inside}} / N_{\text{tries}}$

N	value	error
100	3.08	0.0616
1000	3.092	0.0496
10000	3.118	0.0236

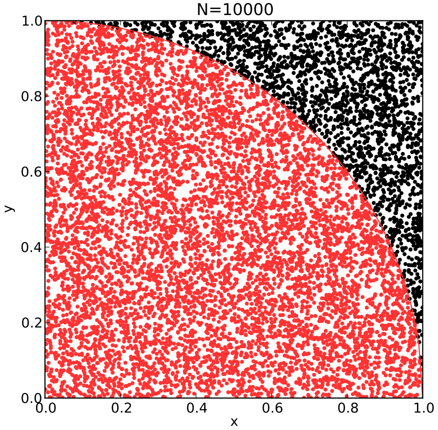


Hit and miss example: Calculate value for π

Monte Carlo algorithm

1. Sample N pairs of random numbers $(x, y), x, y \in [0, 1[$
2. Calculate number of points within a quarter of circle $x^2 + y^2 < 1$
3. Approximated values $= 4 * N_{\text{inside}} / N_{\text{tries}}$

N	value	error
100	3.08	0.0616
1000	3.092	0.0496
10000	3.118	0.0236
100000	3.14752	0.00593

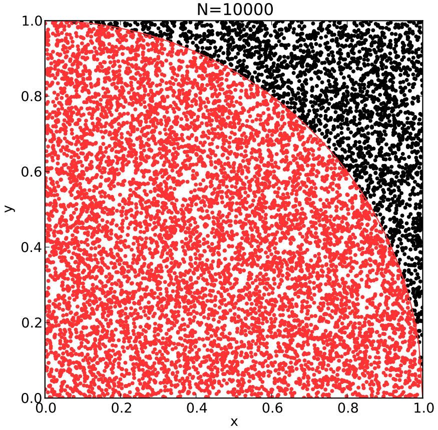


Hit and miss example: Calculate value for π

Monte Carlo algorithm

1. Sample N pairs of random numbers $(x, y), x, y \in [0, 1[$
2. Calculate number of points within a quarter of circle $x^2 + y^2 < 1$
3. Approximated values $= 4 * N_{\text{inside}}/N_{\text{tries}}$

N	value	error
100	3.08	0.0616
1000	3.092	0.0496
10000	3.118	0.0236
100000	3.14752	0.00593
1000000	3.14208	0.000487



- Statistical uncertainty $\propto \frac{1}{\sqrt{N_{\text{events}}}}$

Monte Carlo techniques III: The veto algorithm

Radioactive decays

- Probability $P(t)$ proportional to the number of remaining nuclei $N(t)$:

$$P(t) = -\frac{\mathrm{d}N(t)}{\mathrm{d}t} = c\,N(t) \quad \Rightarrow \quad N(t) = \exp(-ct)$$

- What if c time dependent: $P(t) = f(t)\,N(t)$? Need to solve

$$\frac{\mathrm{d}N(t)}{\mathrm{d}t} = -f(t)\,N(t) \quad \Rightarrow \quad N(t) = \exp\left(-\int_0^t \mathrm{d}t' f(t')\right) = \exp\left(-(F(t) - F(0))\right)$$

  Assuming $N(0) = 1$ and $F(\infty) = \infty$ we can sample decay time t from

$$t = F^{-1}(F(0) - \log(R))$$

- What if $f(t)$ does not have F^{-1} (or even simple F)? Simple hit and miss with $f(t)/g(t)$ would give

$$P(t) = f(t)\exp\left(-\int_0^t \mathrm{d}t' g(t')\right)$$

Monte Carlo techniques III: The veto algorithm

- Assume that we have $g(x)$ such that $f(x) < g(x)$, G simple and invertible

1. Start with $i = 0$ and $t_0 = 0$
2. Take the next step: $i = i + 1$
3. Sample $t_i = G^{-1}(G(t_{i-1}) - \log(R))$ (start from the previous point!)
4. Sample $y = R\,g(t_i)$
5. while $y > f(t)$ cycle to 2., otherwise accept t_i

Winner takes it all

- Have multiple possible decay channels

$$P(t) = -\frac{\mathrm{d}N(t)}{\mathrm{d}t} = f_1(t)N(t) + f_2(t)N(t)$$

Go ahead by combining $f(t) = f_1(t) + f_2(t)$ and pick channel from $f_1(t) : f_2(t)$, Or:

1. Sample t_1 from $P_1(t_1) = f_1(t_1)N(t_1)$
2. Sample t_2 from $P_2(t_2) = f_2(t_2)N(t_2)$
3. Pick channel with smaller t, continue from this

Hard process generation

Internally defined hard processes

QCD processes

- Hard $2 \to 2$ partonic scatterings (some $2 \to 3$)
- Heavy-quark production

Electroweak processes

- Prompt photon production
- EW boson production and exchange
- Deep inelastic scattering
- Photon collisions

Onia production

- Charmonium, Bottomonium with different spin states

Top production

- $t\bar{t}$ pairs and single top

Higgs production

- Standard-Model Higgs, also in association with other particles
- Beyond-the-Standard-Model Higgs

Beyond the Standard Model

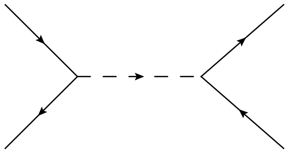
- Supersymmetry
- Dark Matter
- Leptoquarks
- …

Phase-space sampling

- Factorized cross section for $2 \to 2$ scattering

$$\sigma_{ab} = \int \frac{d\tau}{\tau} dy\, d\hat{t}\, x_1 f_a(x_1, Q^2)\, x_2 f_b(x_2, Q^2)\, \frac{d\hat{\sigma}(\hat{s}, \hat{t}, Q^2)}{d\hat{t}},$$

where $\tau = x_1 x_2$ and $y = 0.5 \log(x_1/x_2)$ is the rapidity of the outgoing particles



- $\hat{s}$, $\hat{t}$ and $\hat{u}$ are the (partonic) Mandelstam variables
$\hat{p}_\mathrm{T}^2 = \frac{\hat{t}\hat{u}}{\hat{s}}$ transverse momentum of the outgoing partons
- Phase space parametrized with: τ, y and z, where $z = \cos\hat{\theta}$ (instead of $\hat{t}$)

$\Rightarrow$ Can define cuts for $\hat{s}_\mathrm{min}$, $\hat{s}_\mathrm{max}$ $\hat{p}_\mathrm{T,min}$ and $\hat{p}_\mathrm{T,max}$

$$\frac{\hat{s}_\mathrm{min}}{s} < \tau < \frac{\hat{s}_\mathrm{max}}{s}$$

$$-\frac{1}{2}|\log\tau| < y(\tau) < \frac{1}{2}|\log\tau|$$

$$\sqrt{1 - \frac{4\hat{p}_\mathrm{T,max}^2}{\tau s}} < |z(\tau)| < \sqrt{1 - \frac{4\hat{p}_\mathrm{T,min}^2}{\tau s}}$$

Phase-space biasing

- Phase-space sampling according to the cross section
  - $\Rightarrow$ More events in regions where cross section large
  - $\Rightarrow$ Difficult to populate all parts of the phase space

Example: dijet events

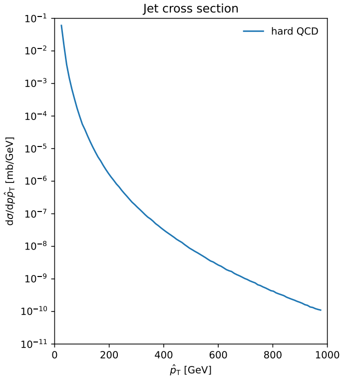
- Cross section of QCD $2 \to 2$ processes behave as

$$\frac{\mathrm{d}\sigma}{\mathrm{d}\hat{p}_\mathrm{T}} \propto \frac{1}{\hat{p}_\mathrm{T}^n}$$

  where $n \approx 4 - 6$ depending on collision energy

- How to fullfill phase space with large $\hat{p}_\mathrm{T}$ span?

1. Generate events in smaller $\hat{p}_\mathrm{T}$ slices
2. Bias event sampling, compensate by assigning a weight for each event



Running Pythia

`main01.cc`

```
#include "Pythia8/Pythia.h"
using namespace Pythia8;
int main() {
  // Generator. Process selection. LHC initialization. Histogram.
  Pythia pythia;
  pythia.readString("Beams:eCM = 8000.");
  pythia.readString("HardQCD:all = on");
  pythia.readString("PhaseSpace:pTHatMin = 20.");
  pythia.init();
  Hist mult("charged multiplicity", 100, -0.5, 799.5);
  // Begin event loop. Generate event. Skip if error. List first one.
  for (int iEvent = 0; iEvent < 100; ++iEvent) {
    if (!pythia.next()) continue;
    // Find number of all final charged particles and fill histogram.
    int nCharged = 0;
    for (int i = 0; i < pythia.event.size(); ++i)
      if (pythia.event[i].isFinal() && pythia.event[i].isCharged())
        ++nCharged;
    mult.fill( nCharged );
  // End of event loop. Statistics. Histogram. Done.
  }
  pythia.stat();
  cout << mult;
  return 0;
```

- Run as "main" programs
- Set up beams (default LHC@14TeV)
- Pick hard process(es) and set phase-space cuts
- Generate an event
- Analyse outgoing particles
- Repeat *n* times
- Collect results

Preparations

- Download the Docker image:

```
$ docker pull hepstore/rivet-pythia
```

- Start the container and set up the current directory as a “host” directory (This way you can open and modify files as they would be local files):

```
$ docker run -v $PWD:/host -it --rm hepstore/rivet-pythia
```

- Copy an example to your `/host` folder:

```
$ cd /host
$ cp ../usr/local/share/Pythia8/examples/main01.cc .
$ cp ../usr/local/share/Pythia8/examples/Makefile* .
```

- Compile and run the example:

```
$ make main01 && ./main01
```

Excercise I: Phase-space biasing

- Start from the `main01.cc` example in `/host`

```
$ cp main01.cc mymain01.cc
```

- Open mymain01.cc with a text-editor, modify such that

```
pythia.readString("PhaseSpace:pTHatMin=50.");
pythia.readString("PartonLevel:all=off");
pythia.readString("HadronLevel:all=off");
Hist pTevent("Hard-process-pT", 100, 0., 1000.);
```

- Within the event loop, save $\hat{p}_T$ and fill histogram

```
double pTnow=pythia.info.pTHat();
pTevent.fill(pTnow);
```

- Increase the number of events to `10k`, print histogram

```
cout << pTevent;
```

Exercise I: Phase-space biasing

```
2024-06-25 10:43        Hard-process pT

      8.10*10^ 2        9
      7.80*10^ 2        X
      7.50*10^ 2        X
      7.20*10^ 2        X
      6.90*10^ 2        X
      6.60*10^ 2        X
      6.30*10^ 2        X
      6.00*10^ 2        X
      5.70*10^ 2        X
      5.40*10^ 2        X
      5.10*10^ 2        X
      4.80*10^ 2        X
      4.50*10^ 2        X
      4.20*10^ 2        X
      3.90*10^ 2        X
      3.60*10^ 2        X
      3.30*10^ 2        XX
      3.00*10^ 2        XX
      2.70*10^ 2        XX
      2.40*10^ 2        XX
      2.10*10^ 2        XX
      1.80*10^ 2        XX5
      1.50*10^ 2        XXX
      1.20*10^ 2        XXX
      0.90*10^ 2        XXX6
      0.60*10^ 2        XXXX5
      0.30*10^ 2        XXXXX96322111

        Contents
          *10^ 2  0000083100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
          *10^ 1  0000026742110000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
          *10^ 0  0000068564770643211000000000000000000000000000000000000000000000000000000000000000000000000000000000
          *10^-1  0000086387642471251965432211111100000000000000000000000000000000000000000000000000000000000000000000

        Low edge
          *10^ 2  0000000000111111111122222222223333333333444444444455555555556666666666777777777788888888889999999999
          *10^ 1  0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
          *10^ 0  0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

 Entries  =     10000   xMin =     0.000   Underflow =     0.000   Mean   =    65.884   RMS  =    19.263
 SumW(in) =  1500.296   xMax =  1000.000   Overflow  =     0.000   Median =    59.298   nEff =  2940.378
```

Exercise I: Phase-space biasing

- Enable phase-space biasing:

```
pythia.readString("PhaseSpace:bias2Selection=on");
pythia.readString("PhaseSpace:bias2SelectionPow=4.");
pythia.readString("PhaseSpace:bias2SelectionRef=50.");
```

- Add another histogram

```
Hist pTweighted("Weighted-hard-process-pT", 100, 0.,
   1000.);
```

- Within the event loop, fill the new histogram including event weight

```
double weight = pythia.info.weight();
pTweighted.fill(pTnow,weight);
```

- Print histograms

```
cout << pTevent << pTweighted;
```

Exercise I: Phase-space biasing

Modify `bias2SelectionPow` to get roughly a flat $\hat{p}_T$ dependence, histogram with weights should remain as before

```
2024-06-25 10:45       Hard-process pT

      1.16*10^ 3       5
      1.12*10^ 3       X
      1.08*10^ 3       X
      1.04*10^ 3       X
      1.00*10^ 3       X
      0.96*10^ 3       X
      0.92*10^ 3       X9
      0.88*10^ 3       XX
      0.84*10^ 3       XX5
      0.80*10^ 3       XXX
      0.76*10^ 3       XXX
      0.72*10^ 3       XXX
      0.68*10^ 3       XXX
      0.64*10^ 3       XXX9
      0.60*10^ 3       XXXX5
      0.56*10^ 3       XXXXX2
      0.52*10^ 3       XXXXXX1
      0.48*10^ 3       XXXXXXX
      0.44*10^ 3       XXXXXXX
      0.40*10^ 3       XXXXXXX9
      0.36*10^ 3       XXXXXXXX54
      0.32*10^ 3       XXXXXXXXXX0
      0.28*10^ 3       XXXXXXXXXXX5
      0.24*10^ 3       XXXXXXXXXXXX712
      0.20*10^ 3       XXXXXXXXXXXXXXX41
      0.16*10^ 3       XXXXXXXXXXXXXXXXX665
      0.12*10^ 3       XXXXXXXXXXXXXXXXXXXX97353 4
      0.08*10^ 3       XXXXXXXXXXXXXXXXXXXXXXXXXXX57471321
      0.04*10^ 3       XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX9X9X7773668456554535332333333112313111111 111 111 111  11111

        Contents
          *10^ 3  0000010000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
          *10^ 2  0000019865543332222211111110100000000000000000000000000000000000000000000000000000000000000000000000
          *10^ 1  0000041137289338620076443109098957564554333222122311212121111011111100010100000000000000000000000000
          *10^ 0  0000017859746842193874459592110790765304484989945437848050381382203315592324552441523123313321122434

        Low edge
          *10^ 2  0000000000111111111122222222223333333333444444444455555555556666666666777777777788888888889999999999
          *10^ 1  0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
          *10^ 0  0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

 Entries  =      10000  xMin =     0.000  Underflow =     0.000  Mean    =    168.18  RMS  =    139.88
 SumW(in) =   9959.000  xMax =  1000.000  Overflow  =    41.000  Median  =    117.49  nEff =  9959.000
```

Exercise II: Inclusive jet production at the LHC

- Copy an example configuration to run with `main93`

```
$ cp ../usr/local/share/Pythia8/examples/main93.cmnd .
```

- Include hard process and phase-space cuts from Excercise 1, remove `SoftQCD:all = on`

```
Beams:eCM = 8000.
HardQCD:all = on
PhaseSpace:pTHatMin = 50.
```

- Include Rivet analysis for inclusive jets by ATLAS, remove others

```
Main:analyses = ATLAS_2017_I1604271
```

- Run generation for 10 000 events $\mathcal{O}(1\text{ min})$

```
$ pythia8-main93 -c main93.cmnd -o pp8TeV-flat
```

Exercise II: Inclusive jet production at the LHC

- Enable phase-space biasing from Excercise I in `main93.cmnd`
- Generate 10 000 events $\mathcal{O}$(3 min)

```
$ pythia8-main93 -c main93.cmnd -o pp8TeV-bias
```

- There are now two `.yoda` files, compare to data and plot

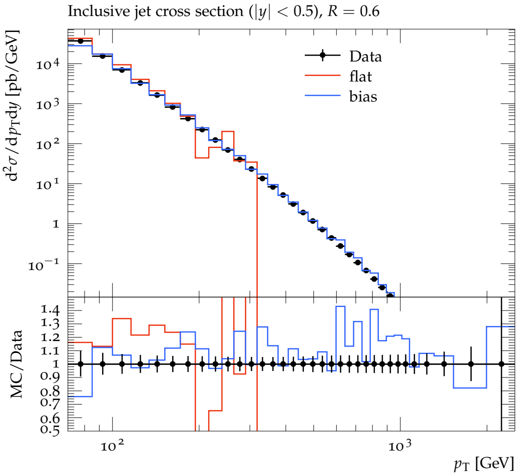
```
rivet-mkhtml pp8TeV-flat.yoda:flat pp8TeV-bias.yoda:bias
    -o html-jets
```

- Open the resulting `html-jets/index.html` with a browser outside the container

Why Rivet? https://rivet.hepforge.org

- Allow for straightforward data comparison and plotting!

Exercise II: Inclusive jet production at the LHC



Final result

- Run out of statistics at large p_T with flat phase-space sampling
- Biasing the phase space with p_T^n and compensating with event weight can help a lot
- Fair agreement with the data over several orders of magnitude

Backup slides

Class structure of Pythia