A new way of η_c production study in Pythia8

A. Anufriev[1]

[1] Samara National Research University

11 April 2023
JINR, Dubna

Talk

Introduction

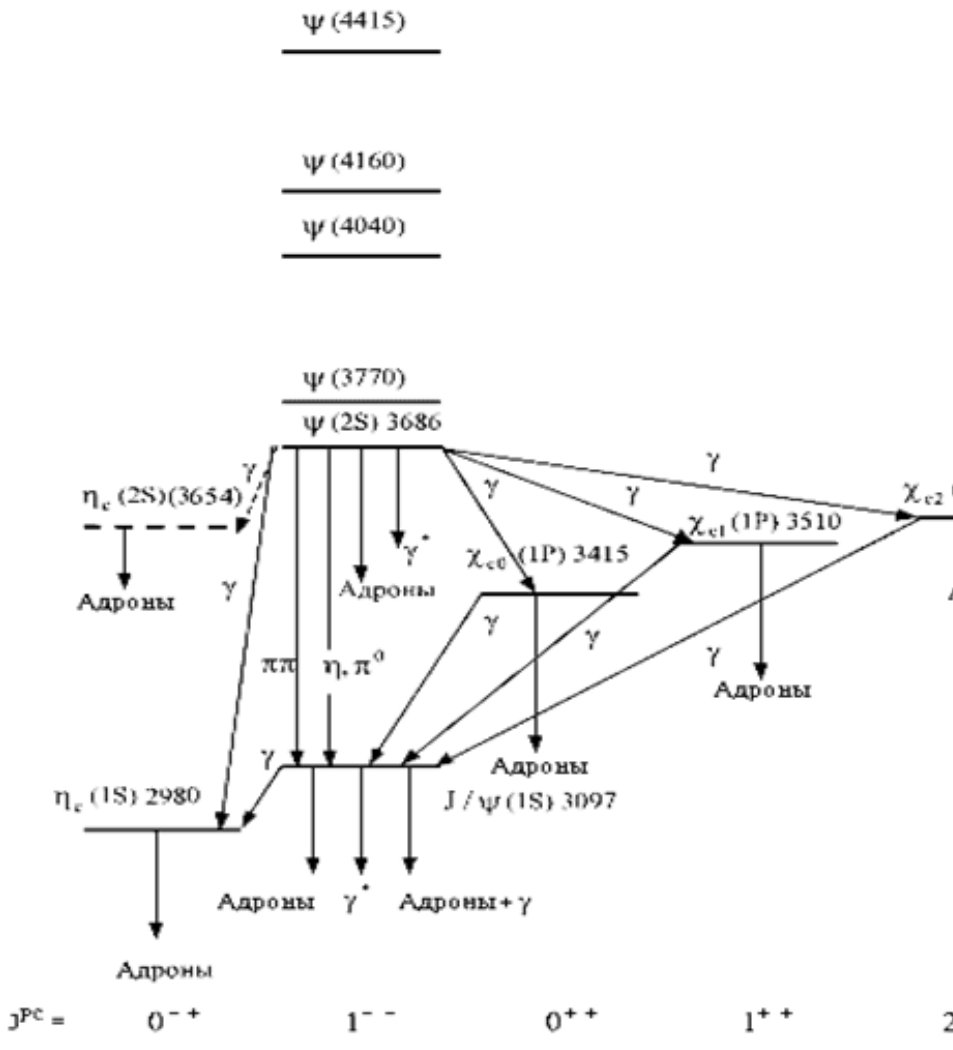
$\eta_c = c\bar{c}[^1S_0]$, $M(\eta_c) = 2.981$ GeV, $\Gamma = 29.7$ MeV

$$Br(\eta_c \to p\bar{p}) = 1.4 \times 10^{-3}$$

$$Br(\eta_c \to \Lambda\bar{\Lambda}) = 9.4 \times 10^{-4}$$

$$Br(\eta_c \to K\bar{K}\pi) = 7.2 \times 10^{-2}$$

$$\boxed{Br(\eta_c \to \gamma\gamma) = 1.78 \times 10^{-4}}$$



η_c production in theoretical calculations

There is my previous theoretical study of η_c production in

- **Collinear Parton Model (CPM)** for $g + g \to \eta_c + g$ process
- **Generalized Parton Model (GPM)** for $g + g \to \eta_c$

So, it is interesting for me find a way to calculate this in Pythia8.

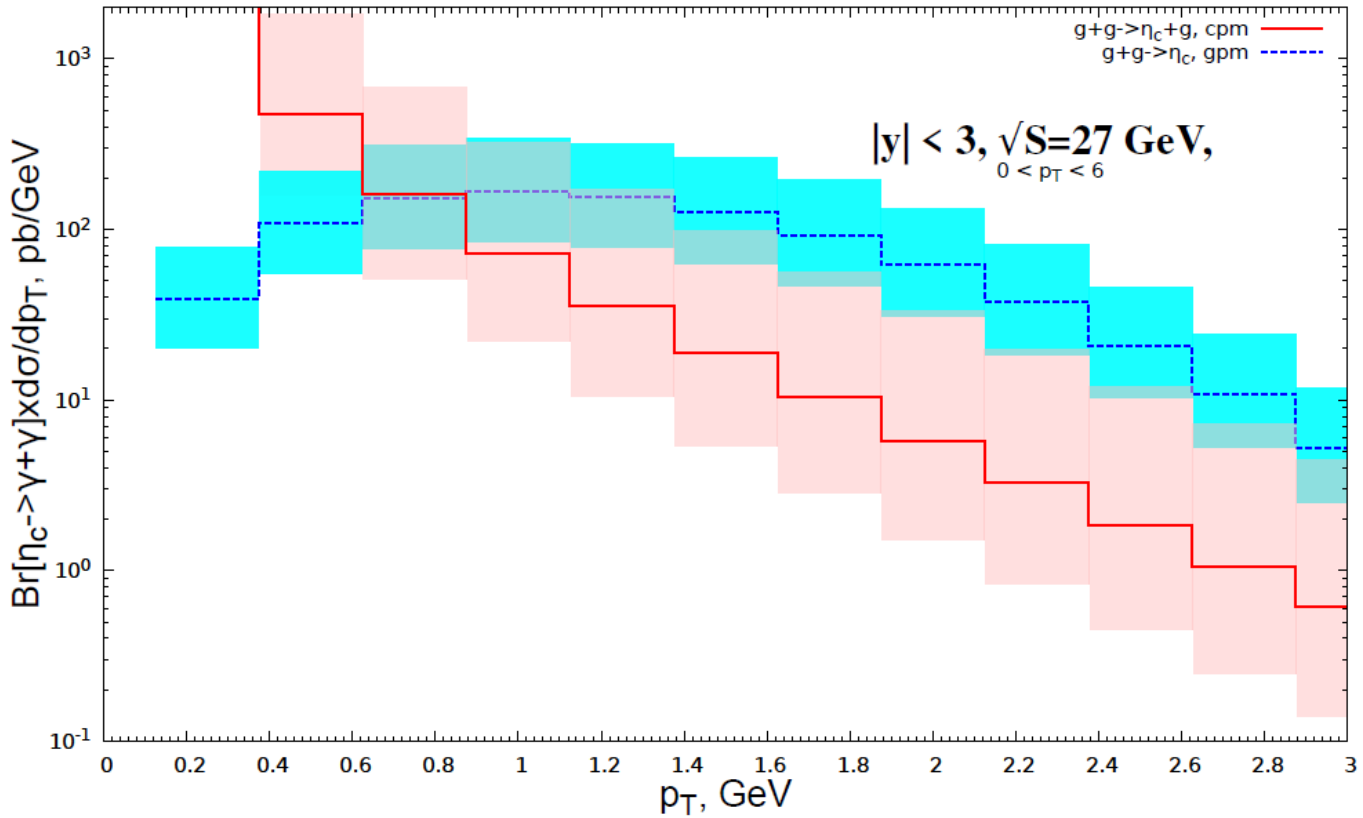


Figure: $\frac{d\sigma}{dp_T}$

η_c production from J/ψ production model

We can use command

```
"Charmonium:gg2ccbar(3S1)[3S1(1)]g=[on,off]"
```

for example, for $J/\psi + g$ production inner process of Pythia. There is also flags

```
"Charmonium:states(3S1)=[443,100443]"
"Charmonium:O(3S1)[3S1(1)] = [1.16,0.76]"
```

to define exact final charmonium from Particle Data class and Long-Distance Matrix Element (LDME) to this process.
Usually first command is used with changing of J/ψ parameters as corresponds η_c

```
"443:m0 = 2.98100"
"443:mWidth = 0.02970"
"443:oneChannel = 1 1 100 22 22"
```

I also tried to use:

```
"Charmonium:states(3S1)=[441,100443]"
"Charmonium:gg2ccbar(3S1)[3S1(1)]g=[on,off]"
"Charmonium:O(3S1)[3S1(1)] = [0.39,0.76]"
```

which is very similar.

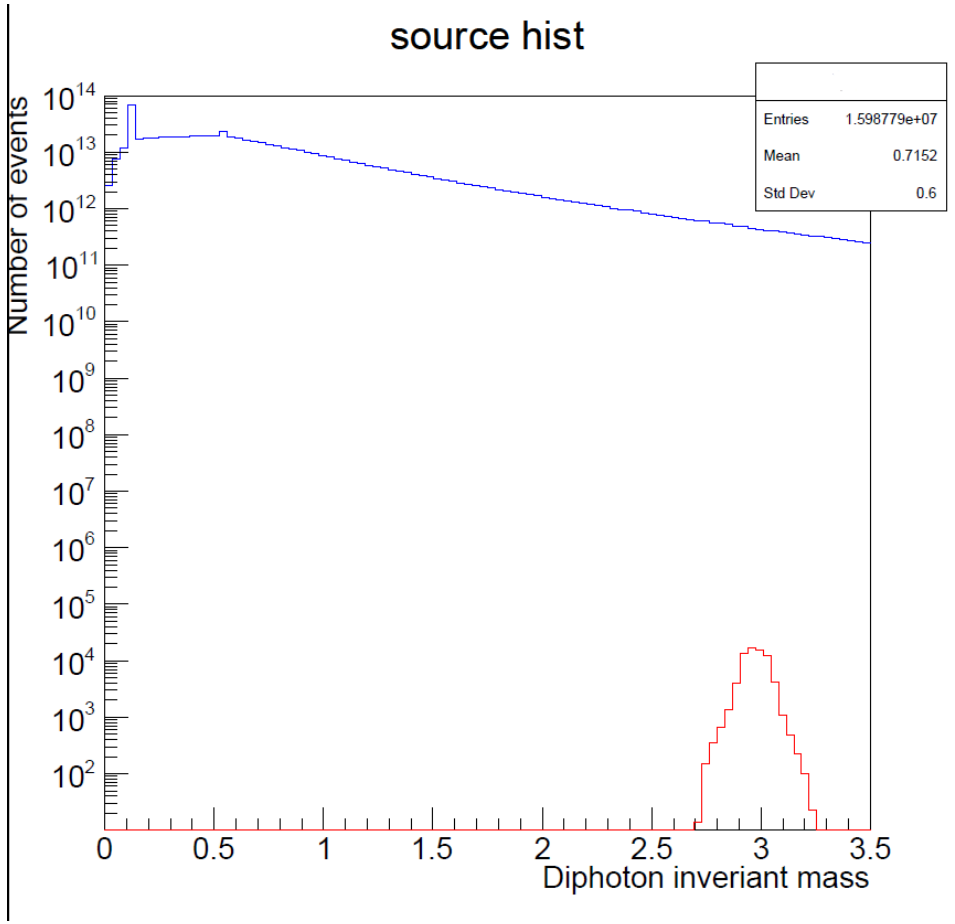


Figure: J/ψ production with η_c production distribution by diphoton invariant mass

Class-container of processes info in Pythia

There is a paragraph in the Pythia manual **"Semi-internal process"** where it is written about the advantages of using Pythia and creating a new process using the inheritance of the SigmaProcess class from all processes in Pythia.
Example **main22.cc** of using this class is in */examples* directory of Pythia.

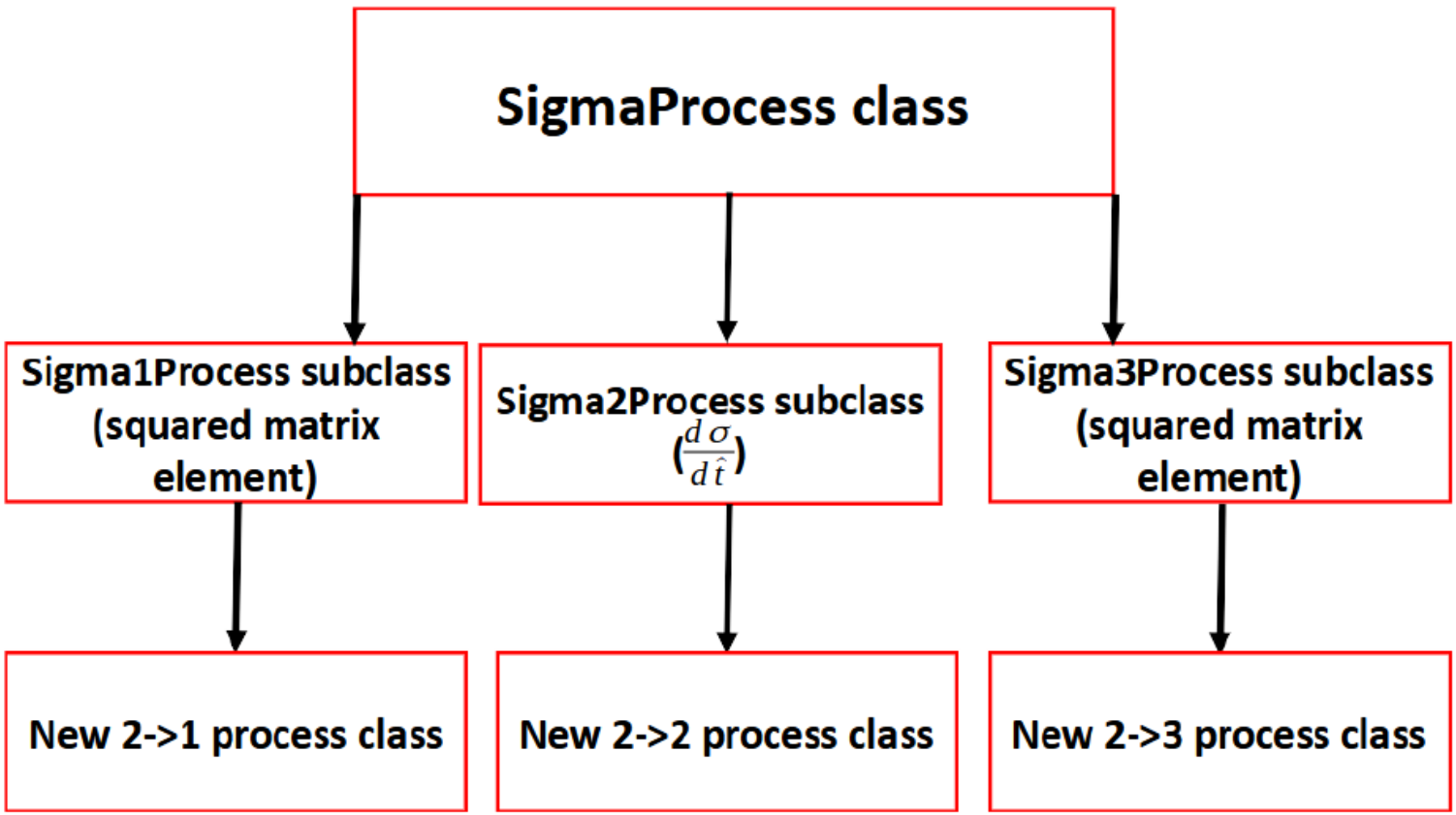


Figure: Inheritance scheme of SigmaProcess family

Methods and objects

There is set of methods that inherit all methods from SigmaClass. Most important methods is:

- **sigmaKin()**, which defines variable **sigma**. SigmaHat() method returns sigma to evaluate total cross section of process.
- **setIdColAcol()** which defines flavours, colours and anticolours of particles in process

Some number of methods must be defined to return important parameters of new process:

- string **name()** returns name of created process
- int **code()** is return **new** code of process (it is not recommended define this code as equal to Pythia inner code of some process)
- string **inFlux()** returns important variable of string type initial pair of particles. For example "gg" for gluons or "qq" for quarks (all list of pairs you can find in online manual)
- int **id3Mass()** returns id of third particle. Note that if there is no this method in new class declaration then Pythia define charmonium as zero-mass particle **even** if you define mass individually.

To switch on a new process we must create a object of a new class with commands:

```
SigmaProcess* ObjectName = new ClassName();
 pythia.setSigmaPtr(ObjectName);
```

Color flow in SigmaProcess

QCD, described by SU(3) , has rich structure.

- Quarks referred to as triplets: single color
- Gluons referred to as octets: two colors
- All others referred to as singlets: no color

```
// Select identity, colour and anticolour.

void Sigma2gg2QQbar3S11g::setIdColAcol() {

  // Flavours are trivial.
  setId( id1, id2, idHad, 21);

  // Two orientations of colour flow.
  setColAcol( 1, 2, 2, 3, 0, 0, 1, 3);
  if (rndmPtr->flat() > 0.5) swapColAcol();

}
```

There is just PDG of particles in setid(), but setColAcol() use Les Houches style colour tags , but starting with number 1. The input is grouped particle by particle, with the colour index before the anticolour one.

SigmaOnia class

Charmonium production processes is defined in SigmaOnia class which inherits from SigmaProcess main set of methods and variables.

```
void Sigma2gg2QQbar3S11g::sigmaKin() {

  // Calculate kinematics dependence.
  double stH = sH + tH;
  double tuH = tH + uH;
  double usH = uH + sH;
  double sig = (10. * M_PI / 81.) * m3 * ( pow2(sH * tuH)
    + pow2(tH * usH) + pow2(uH * stH) ) / pow2( stH * tuH * usH );

  // Answer.
  sigma = (M_PI/sH2) * pow3(alpS) * oniumME * sig;

}
```

Cross section from R. Gastmans, W. Troost, T.T. Wu, Cross-sections for gluon + gluon $\rightarrow$ heavy quarkonium + gluon// Phys. Lett. B 184 (1987) 257:

$$\frac{\mathrm{d}\sigma}{\mathrm{d}t} = \frac{5\pi\alpha_S^3 R_0^2 M}{9s^2}\left(\frac{s^2}{(t-M^2)^2(u-M^2)^2} + \frac{t^2}{(u-M^2)^2(s-M^2)^2} + \frac{u^2}{(s-M^2)^2(t-M^2)^2}\right)$$

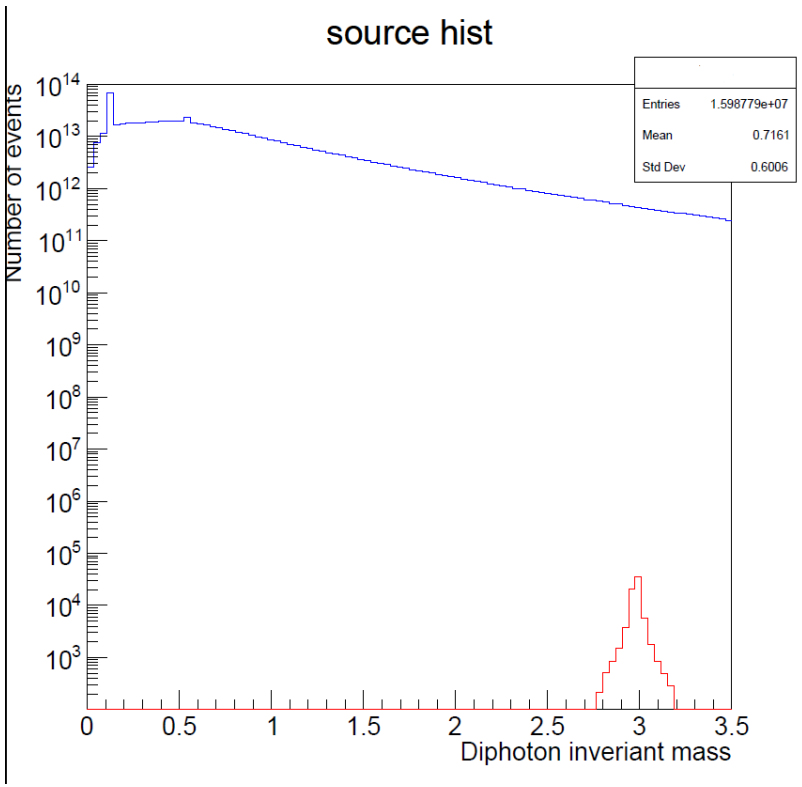
then for $g + g \rightarrow \eta_c + g$:

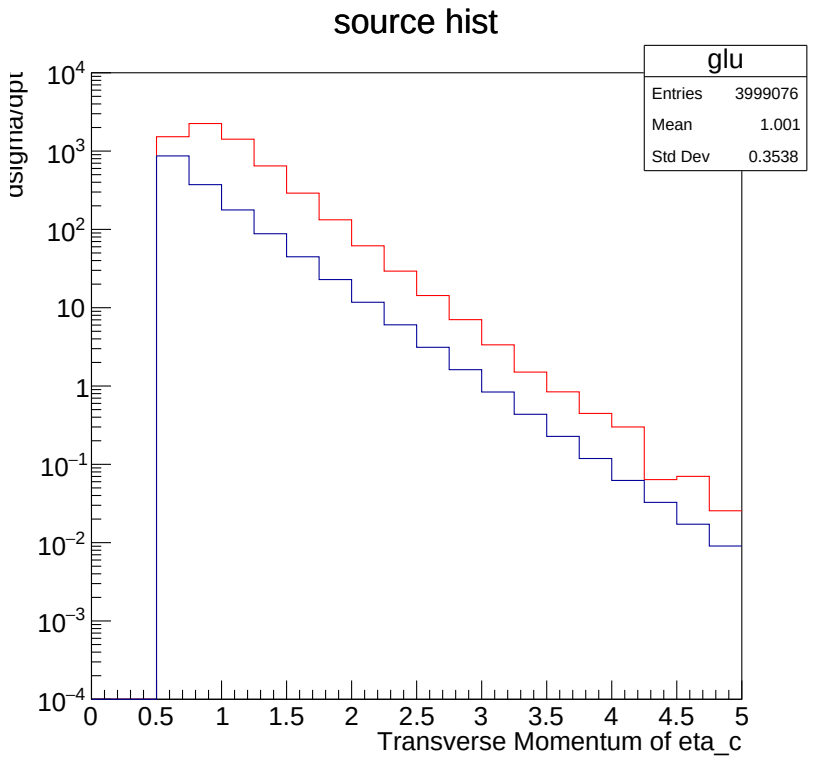
$$\frac{d\sigma}{dt} = \frac{\pi\alpha_s^3 R_0^2}{8M_{\eta_c}\hat{s}^2}\left(\frac{M_{\eta_c}^4 - \hat{s}^2 - \hat{t}^2 - \hat{u}^2}{(\hat{s}-M_{\eta_c}^2)(\hat{t}-M_{\eta_c}^2)(\hat{u}-M_{\eta_c}^2)}\right)^2 \frac{M_{\eta_c}^8 + \hat{s}^4 + \hat{t}^4 + \hat{u}^4}{\hat{s}\hat{t}\hat{u}}$$

$g + g \to \eta_c + g$

Specific cuts on p_T in Pythia

Pythia have command "pTHatMin" for usual cuts and command "pTHatMinDiverge" to avoid the divergences of some $2 \to 2$ processes. If at least one final particle have mass below pTHatMinDiverge then cut on p_T is selected from condition Min[pTHatMin, pTHatMinDiverge].
I choose both cuts as $p_T > 0.5$.





$2 \to 1$ process

sigmaHatWrap() is a wrapper which transform squared matrix element into $\hat{\sigma}$ by smearing delta function according to BV

```
double Sigma1Process::sigmaHatWrap(int id1in, int id2in) {

  id1 = id1in;
  id2 = id2in;
  double sigmaTmp = sigmaHat();
  if (convertM2()) {
    sigmaTmp /= 2. * sH;
    // Convert 2 * pi * delta(p^2 - m^2) to Breit-Wigner with same area.
    int idTmp     = resonanceA();
    double mTmp   = particleDataPtr->m0(idTmp);
    double GamTmp = particleDataPtr->mWidth(idTmp);
    sigmaTmp *= 2. * mTmp * GamTmp / ( pow2(sH - mTmp * mTmp)
      + pow2(mTmp * GamTmp) );
  }
  if (convert2mb()) sigmaTmp *= CONVERT2MB;
  return sigmaTmp;

}
```

Squared matrix element for $g + g \to \eta_c$

$$|\bar{M}|^2 = \frac{2}{9}\pi^2\alpha_s^2\frac{<^1 S_0(1) >}{M}$$

$g + g \to \eta_c$

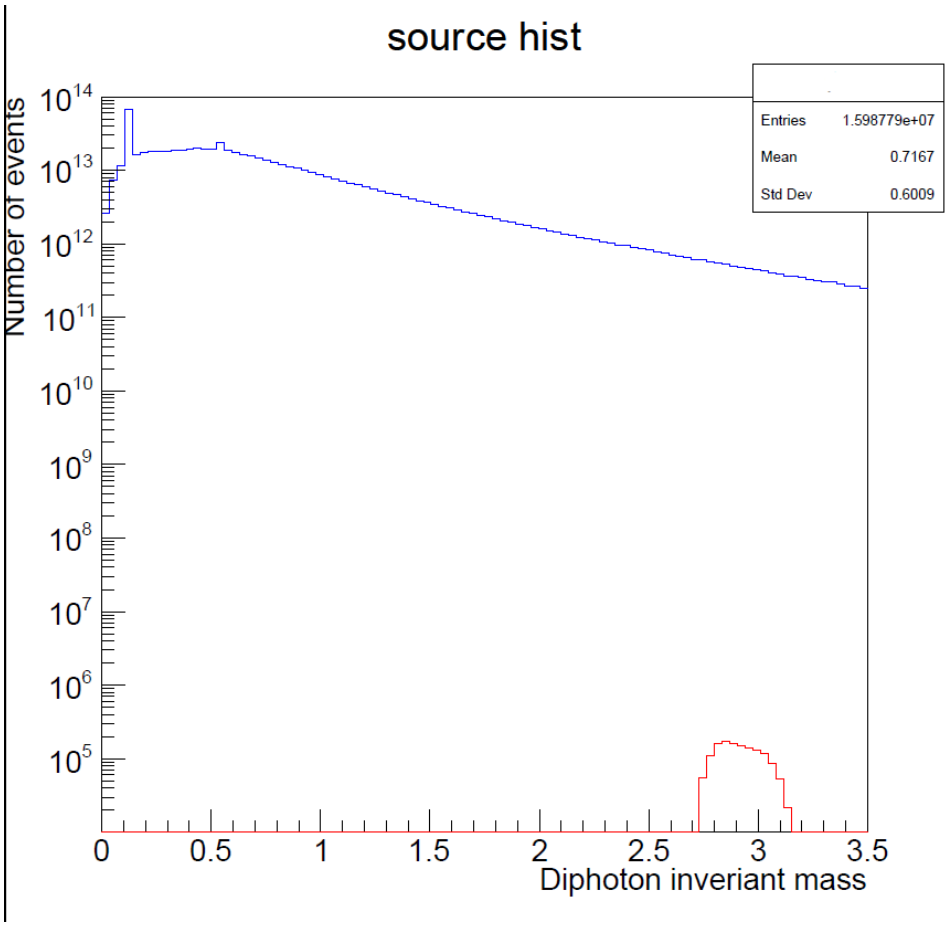


Figure: Event distribution

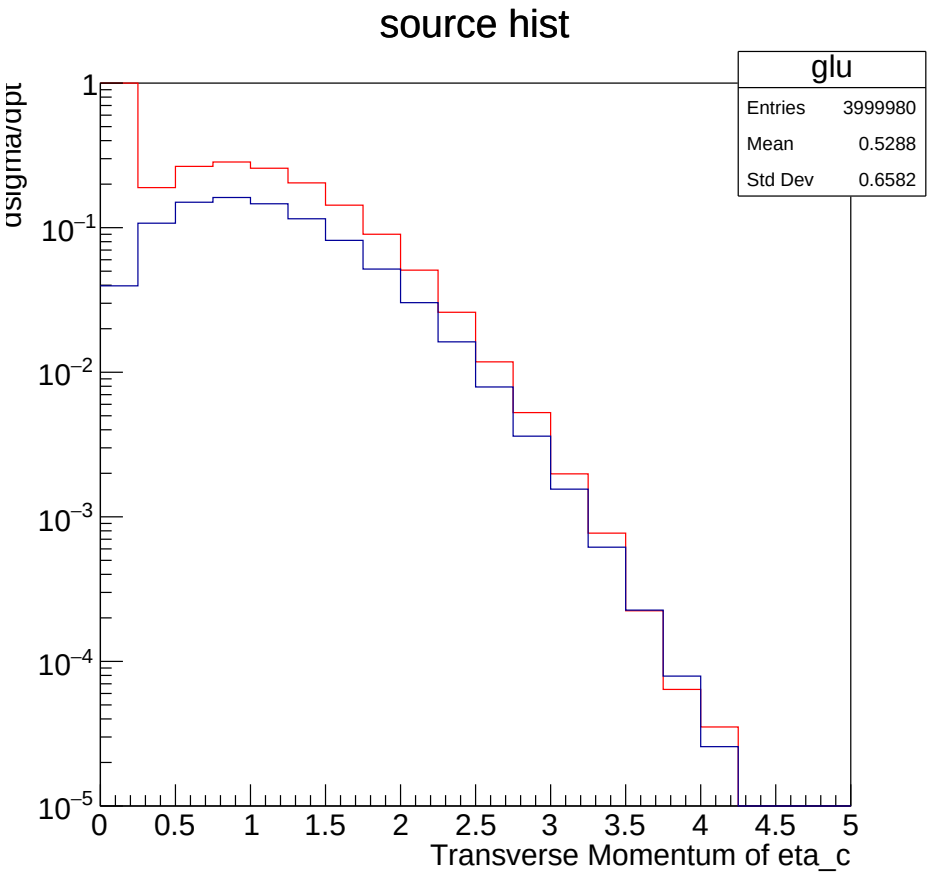
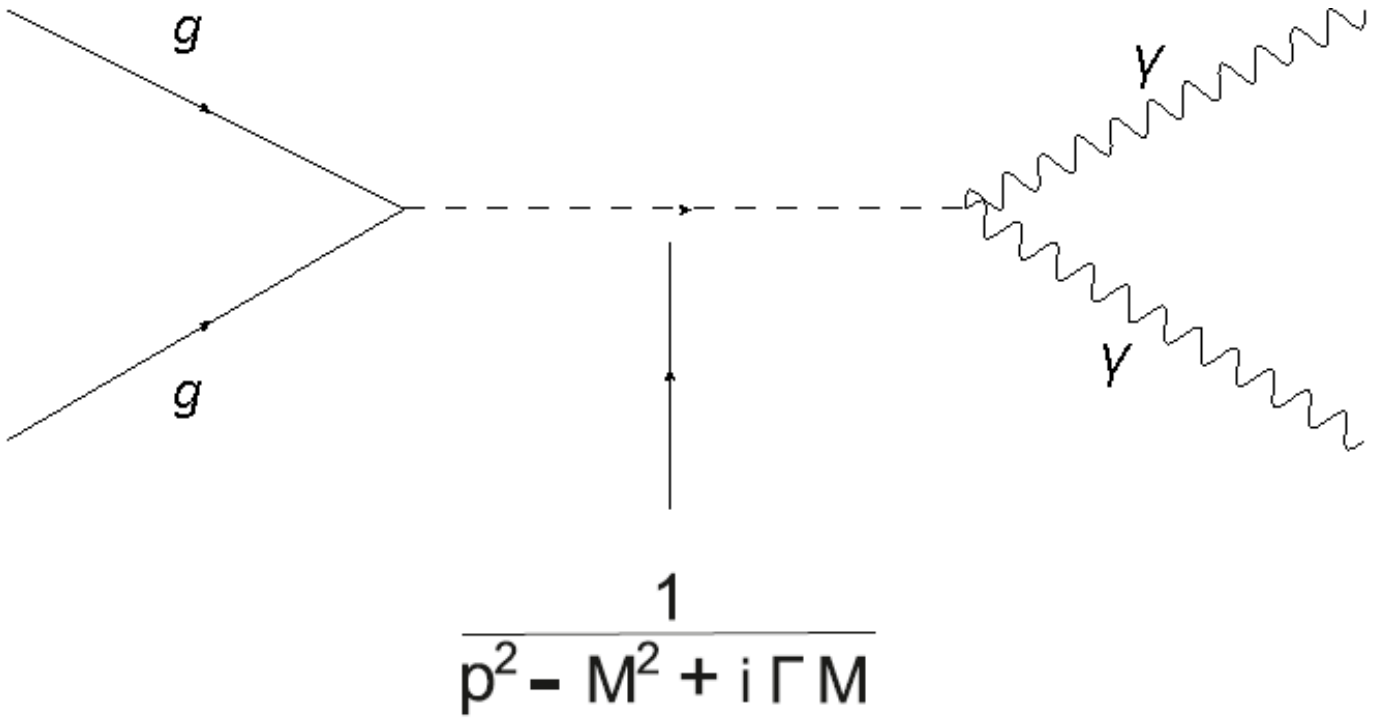


Figure: Comparison of calculations of $\frac{d\sigma}{d\hat{T}}$ in theory (blue) and Pythia (red)

$2 \to 1$ with η_c as a resonance



Formula of squared matrix elements is

$$|\bar{M}|^2 = \frac{8M_{\eta_c}^2 \Gamma_{gg}\Gamma_{\gamma\gamma}}{256((\hat{s} - M_{\eta_c}^2)^2 + M_{\eta_c}^2\Gamma^2/4)}$$

with $\Gamma_{\gamma\gamma} = 12e_c^4\alpha_{em}^2 \frac{|R(0)^2|}{M_{\eta_c}^2}$ and $\Gamma_{gg} = \frac{2}{9}\alpha_s^2 \frac{|R(0)^2|}{M_{\eta_c}^2}$

ResonanceWidths class

To create a new resonance particle we need to create a new class in addition to the class for our process with inheritance from **ResonanceWidth**. There is also an example of how to do this in **main 22.cc**.
For example, I can create a new particle Etac2 with number 663 with all parameters corresponding to η_c in the Pythia database (you can find this in the **Particle Data** paragraph in the online manual).

```
"663:new = Etac2 void 1 0 0 2.98 0.032 2.78 3.1 0."
```

and make the Etac2 resonate using the following commands:

```
"663:isResonance = true"
"663:doForceWidth = on"
ResonanceWidths* resonanceTheta = new ResonanceTheta(663);
  pythia.setResonancePtr(resonanceTheta);
```

$g + g \to \eta_c \to \gamma + \gamma$

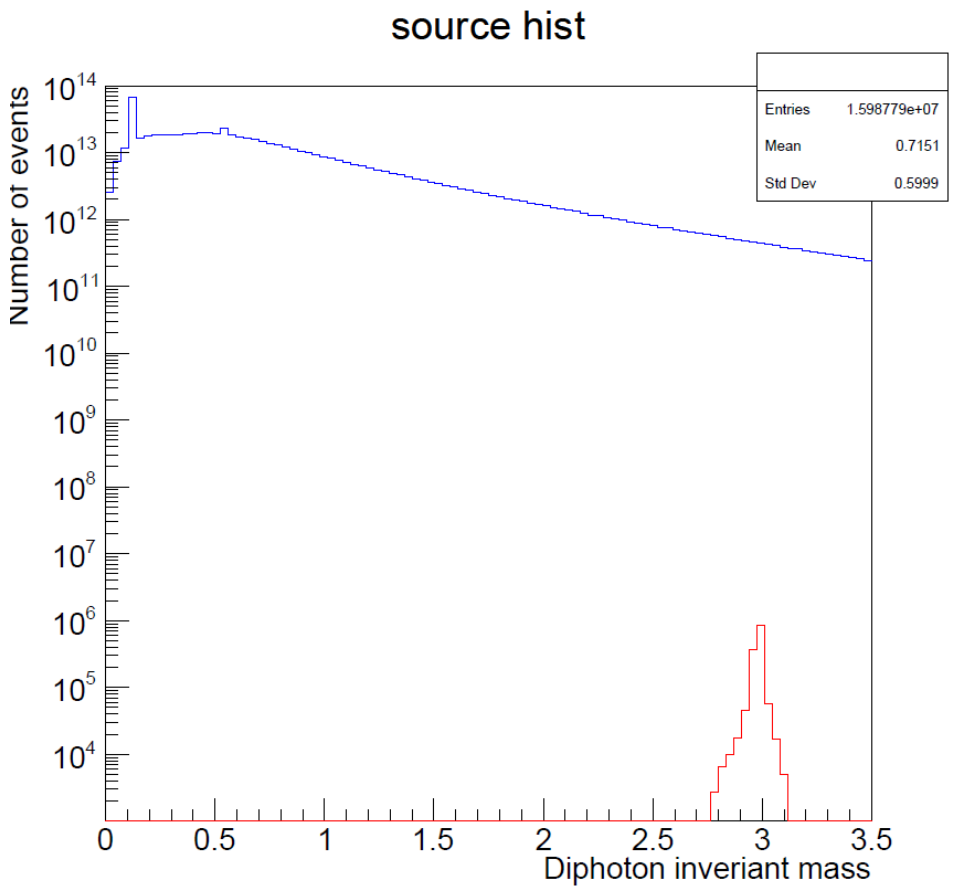


Figure: Event Distibution

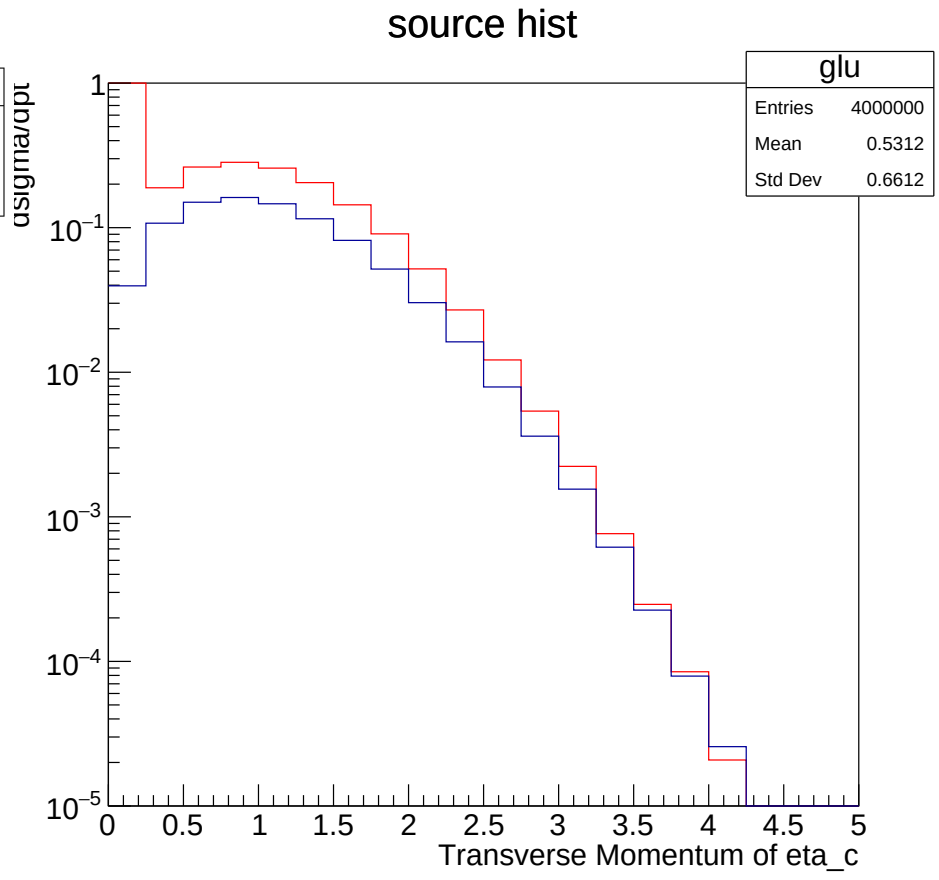


Figure: Comparison of calculations of $\frac{d\sigma}{d\hat{T}}$ in theory (blue) and Pythia (red)

Total cross section comparison

Table: Total cross section comparison in theoretical calculations(include both CPM and GPM) and Pythia

process $g + g \to ...$	Theory	Pythia8
$\eta_c + g$	398 nb **(CPM)**	419 nb
η_c	1283,895 nb **(GPM)**	2230 nb
$\eta_c \to \gamma + \gamma$	0.22 nb **(GPM)**	0.46 nb

In fact, Pythia uses a CPM approach that takes into account initial transverse momenta, but not in the way that GPM does.
Therefore, I switch off the initial k_T to compare the CPM theoretical calculation with Pythia.
GPM differs by a factor of 2 from Pythia results, in my opinion, because of the difference in the consideration of initial p_T in GPM and Pythia.

Conclusions

SigmaProcess class is a good opportunity to create new processes that fits well with the theory. This class provide us to make:

- $2 \to 2$ processes
- $2 \to 3$ processes (not tested by me)
- $2 \to 1$ processes with using ResonanceWidth class in addition

Processes of η_c production that do not exist in the database was implemented and tested. A good agreement with results from theoretical calculations was obtained.

Thank you for your attention!

Comparing of $\frac{d\sigma}{dt}$ in SigmaOnia with formula in Gastmans

```
In[25]:= a1 = (10 * Pi^2 * m * als^3 * 6 * 3 * R^2 / (4 * Pi * 81 s^2)) *
                число пи                                  число пи
     ((s^2 * (t + u)^2 + t^2 * (u + s)^2 + u^2 * (s + t)^2) / ((s + t)^2 * (t + u)^2 * (u + s)^2))
```

$$\text{Out[25]=}\ \frac{5\,\text{als}^3\,\text{m}\,\pi\,\text{R}^2\left((\text{s}+\text{t})^2\,\text{u}^2+\text{t}^2\,(\text{s}+\text{u})^2+\text{s}^2\,(\text{t}+\text{u})^2\right)}{9\,\text{s}^2\,(\text{s}+\text{t})^2\,(\text{s}+\text{u})^2\,(\text{t}+\text{u})^2}$$

```
In[26]:= a2 =
          5 * Pi * als^3 * R^2 * m / (9 * s^2) *
            число пи
           (s^2 / ((t - m^2)^2 * (u - m^2)^2) + t^2 / ((u - m^2)^2 * (s - m^2)^2) +
             u^2 / ((s - m^2)^2 * (t - m^2)^2)) /. {m^4 → (s + t + u)^2, m^8 → (s + t + u)^4, m^2 → s + t + u}
```

$$\text{Out[26]=}\ \frac{5\,\text{als}^3\,\text{m}\,\pi\,\text{R}^2\left(\frac{\text{s}^2}{(-\text{s}-\text{t})^2\,(-\text{s}-\text{u})^2}+\frac{\text{t}^2}{(-\text{s}-\text{t})^2\,(-\text{t}-\text{u})^2}+\frac{\text{u}^2}{(-\text{s}-\text{u})^2\,(-\text{t}-\text{u})^2}\right)}{9\,\text{s}^2}$$

```
In[27]:= a2 - a1 // FullSimplify
                  упростить в полном объёме
```

```
Out[27]= 0
```

It is used NRQCD formula:

$$< \mathcal{O}^{\mathcal{H}}[^3S_1^{(1)}] >= 2N_c(2J+1)\frac{|R(0)|^2}{4\pi}$$