Monte Carlo Techniques

Tim Barklow

Senior Staff Scientist

SLAC National Accelerator Laboratory

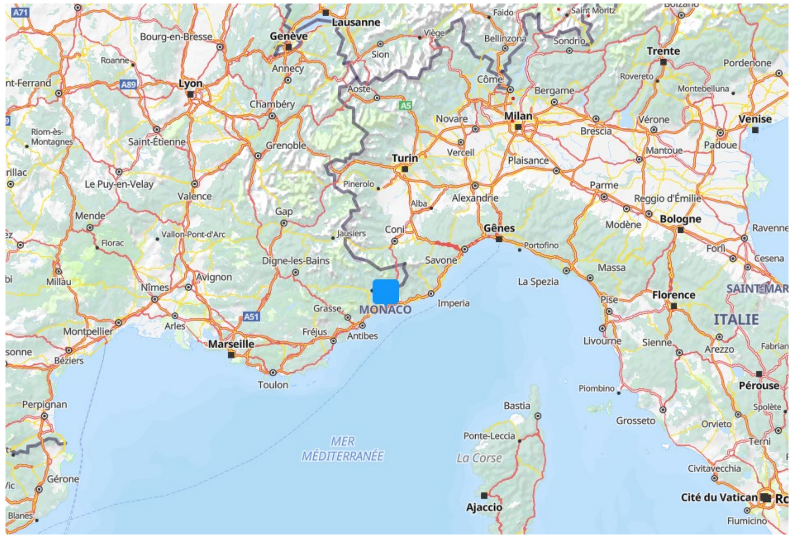
Menlo Park, CA

Overview of Course

- Introduction
- Monte Carlo Integration
    - ❑ General formulae for integration and error
    - ❑ Rejection Method
    - ❑ Example: calculation of π with rejection method
    - ❑ Error analysis of π calculation
- Monte Carlo Simulation of Distributions (aka MC Random Number Generation)
    - ❑ Brute Force Rejection Method
    - ❑ Cumulative Distribution Function (CDF) Inversion Method
    - ❑ Combining Rejection Method with CDF inversion for improved efficiency
    - ❑ Example of energy distribution of Compton scattered photons in $\gamma\gamma$ Higgs factory
- Vegas Monte Carlo Integration and Random Number Generation
    - ❑ Importance sampling as the solution to integration in many dimensions
    - ❑ The Vegas algorithm for both integration and random number generation
    - ❑ Example of 2-d differential $\gamma\gamma$ luminosity distribution – brute force rejection vs. Vegas+rej
- Using these techniques in the CAIN Beam-Beam MC

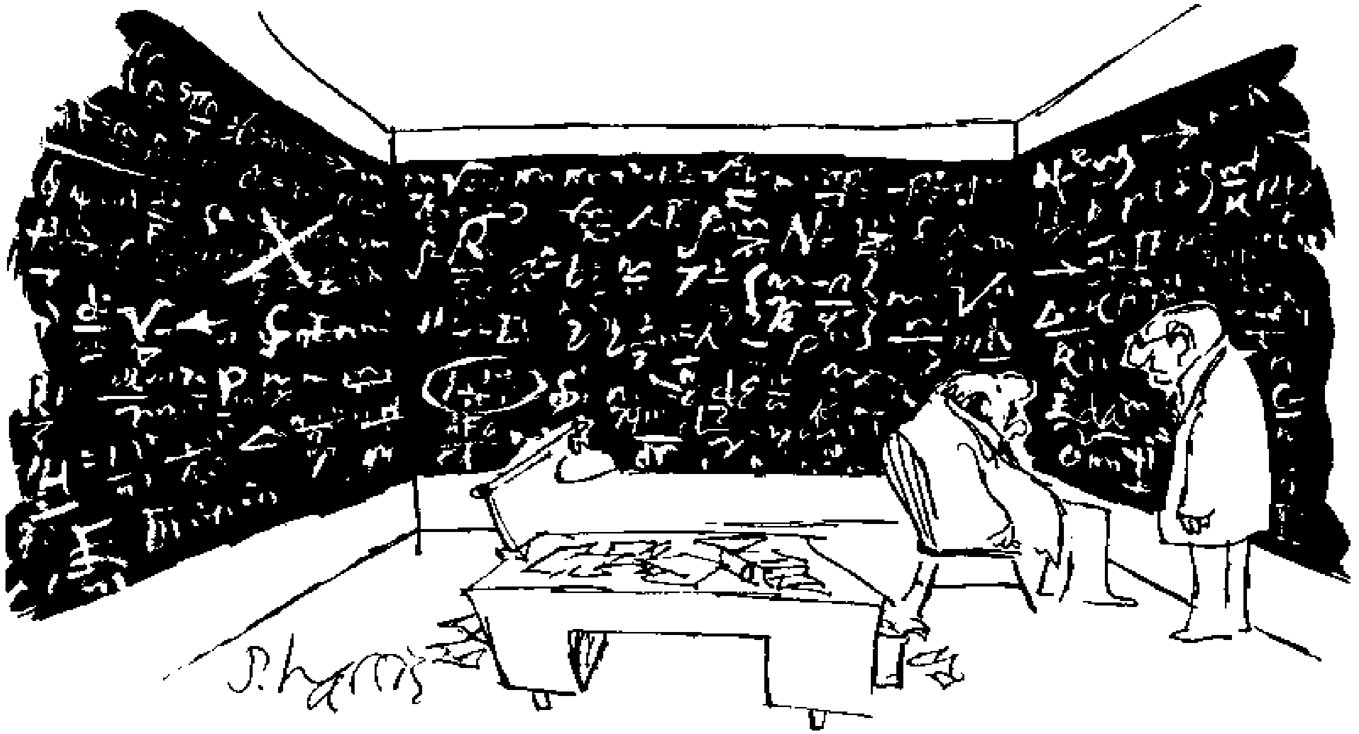
What is Monte Carlo Simulation?

A ***numerical simulation*** method which uses sequences of ***random numbers*** to solve complex problems.



Why use the Monte Carlo Method?

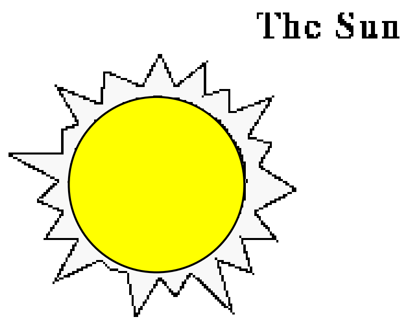
- Other numerical methods typically need a ***mathematical description*** of the system (**ordinary** or **partial differential equations**)
- More and more difficult to solve as complexity increases



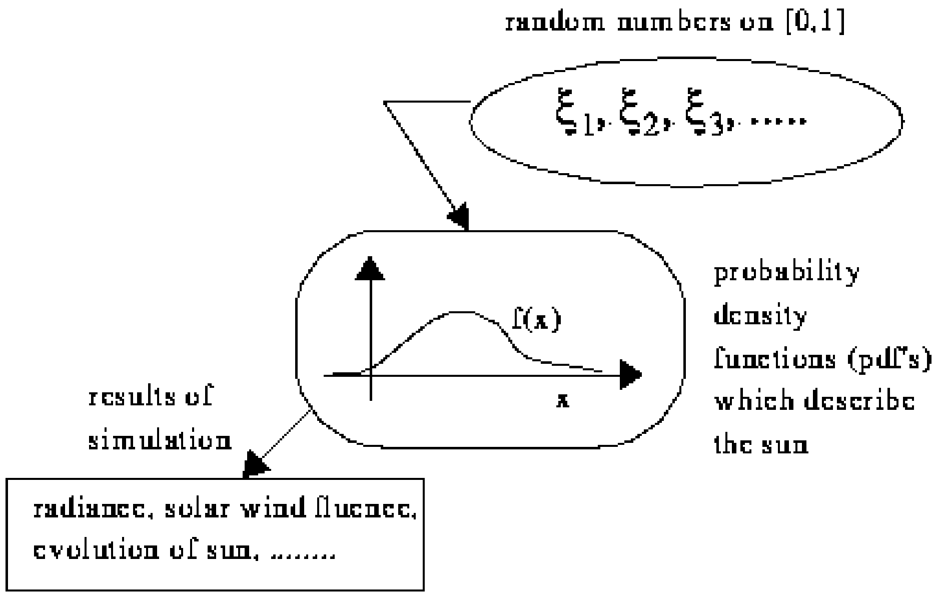
Monte Carlo Simulation Overview

MC assumes that many system components are described by **probability density functions** which can be modeled with ***no need to write down equations***.

These PDF are ***sampled randomly***, ***many simulations*** are performed, particles are propagated through time using the laws of physics, and the result is the **average** over a **number of observations**



Physical System


random numbers on [0,1]
$\xi_1, \xi_2, \xi_3, \ldots$
f(x)
x
probability density functions (pdf's) which describe the sun
results of simulation
radiance, solar wind fluence, evolution of sun, ...

Statistical Simulation

A Brief History - I

- Method formally developed by John Von Neumann during WWII, but already known before

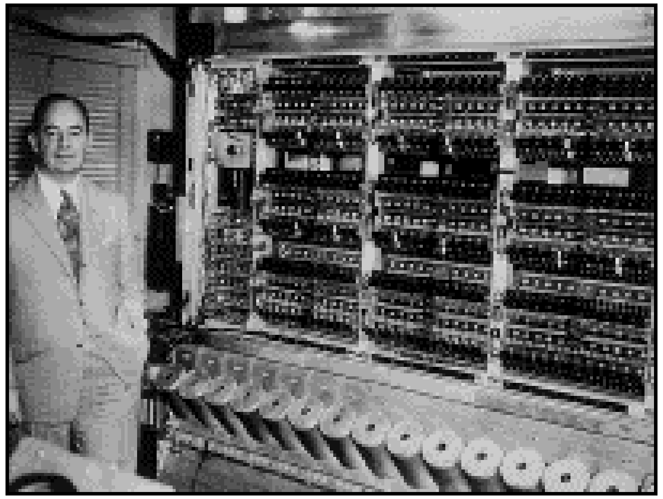
- Fermi used it to simulate **neutron diffusion** in the 1930s. He knew the behavior of one neutron, but he did not have a formula for how a system of neutrons would behave.

He also used it to demonstrate the stability of the **first man-made nuclear reactor** (Chicago Pile, 1942). His model had an analogy with heat diffusion models previously developed.

***Fermi used tables of numbers sorted on a* roulette** to obtain random numbers which he then used in his calculations of neutron absorption.

A Brief History - II

- Manhattan Project of WWII (Von Neumann, Ulam, Metropolis)
  - Scientists used it to construct dampers and shields for the nuclear bomb, experimentation was too time consuming and dangerous.



- Extensively used in many disciplines especially after the advent of high-speed computing:
  - Cancer therapy, traffic flow, Dow-Jones forecasting, oil well exploration, stellar evolution, reactor design, particle physics, ancient languages deciphering,…

Monte Carlo Integration

- Monte Carlo integration and the Monte Carlo simulation of probability density functions (PDF’s) are intimately connected -- they are really one and the same activity.

- General Monte Carlo integration formulae:

$$t = \int d\vec{x}\, g(\vec{x}) = \int d\vec{x}\, \rho(\vec{x})\, f(\vec{x}) \approx \overbrace{\frac{1}{N}\sum_{i=1}^{N} f(\vec{x}_i)}^{\text{Points } \vec{x}_i \text{ in this sum are drawn at random from the distribution } \rho(\vec{x})} = \frac{s}{N} \quad ,$$

$$\int d\vec{x}\, \rho(\vec{x}) = 1 \quad , \qquad s = \sum_i f(\vec{x}_i)$$

For integration we accept all events pulled from the distribution $\rho(\vec{x})$

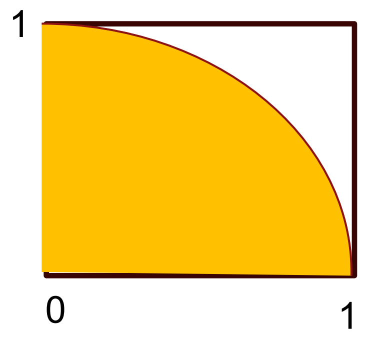
MC integration statistical error: $$(\Delta t)^2 = \frac{1}{N-1}\left[\frac{1}{N}\sum_i f_i^2 - \frac{s^2}{N^2}\right]$$

Exercise 1: Determination of π using MC integration [piCalculation.py]

$$t = \int d\vec{x}\, g(\vec{x}) = \int d\vec{x}\, \rho(\vec{x}) f(\vec{x}) \approx \frac{1}{N}\sum_{i=1}^{N}\frac{g(\vec{x})}{\rho(\vec{x})} = \frac{1}{N}\sum_{i=1}^{N} f(\vec{x}_i) = \frac{s}{N} \quad ,$$

$$\int d\vec{x}\,\rho(\vec{x}) = 1 \quad , \qquad s = \sum_i f_i$$

$$\rho(x,y) = \begin{cases} 1 \text{ if } 0 < x < 1 \;\&\; 0 < y < 1 \\ 0 \text{ otherwise} \end{cases} \qquad f(x,y) = \begin{cases} 1 \text{ if } x^2 + y^2 < 1 \\ 0 \text{ otherwise} \end{cases}$$



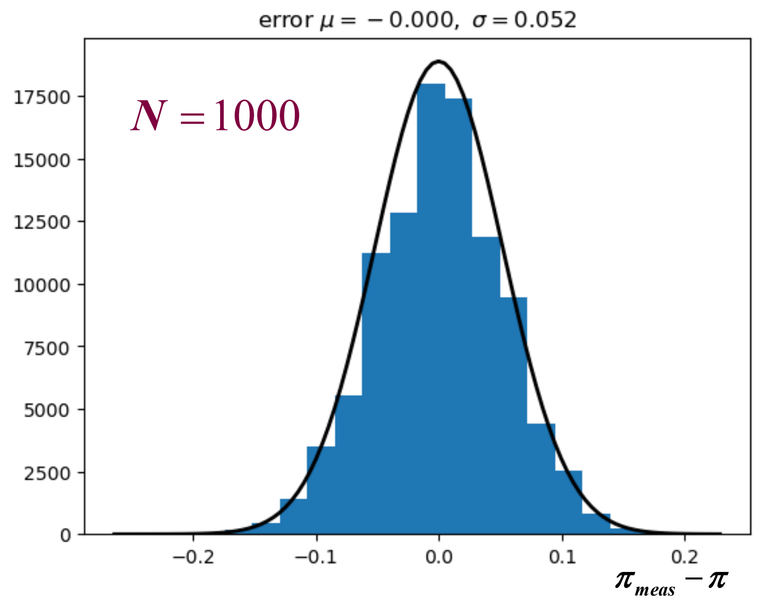
$$t = \frac{1}{N}\sum_{i=1}^{N} f(\vec{x}_i) \rightarrow \frac{\pi}{4} \text{ as } N \rightarrow \infty$$

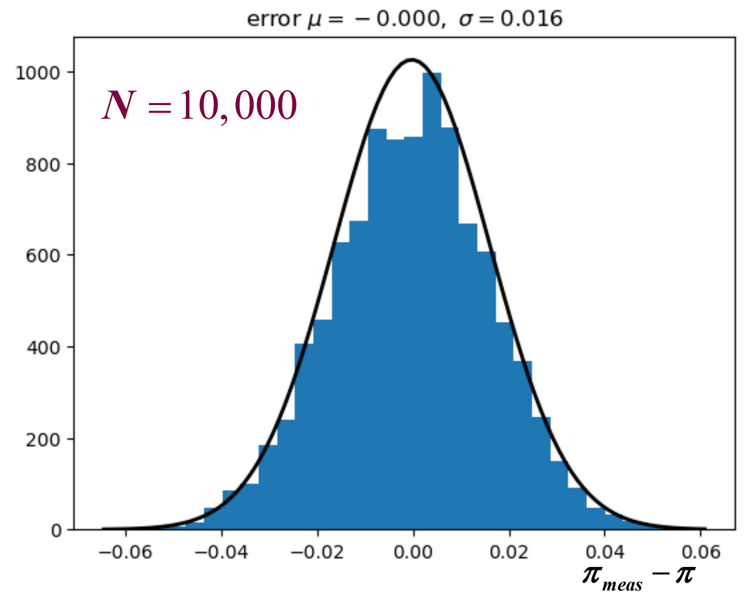
$$(\Delta t)^2 = \frac{1}{N-1}\left[\frac{1}{N}\sum_i f_i^2 - \frac{s^2}{N^2}\right] = \frac{1}{N-1}\left[\frac{s}{N} - \frac{s^2}{N^2}\right] = \frac{1}{N-1}\frac{\pi}{4}\left[1 - \frac{\pi}{4}\right]$$

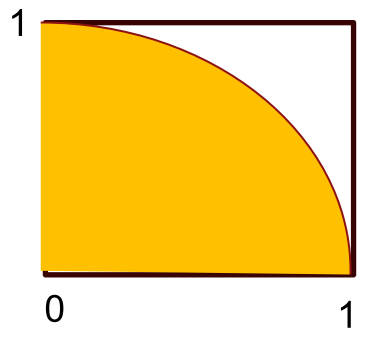
$$\Delta\pi = 4\Delta t = 4\left[\frac{1}{N-1}\frac{\pi}{4}\left[1-\frac{\pi}{4}\right]\right]^{\frac{1}{2}} = \frac{1.642}{\sqrt{N-1}}$$

From the distributions produced by piCalculation.py check that you get this error. Also, the pulls = deviation/error are plotted. These should give a normal distribution with σ=1 if errors are properly calculated.

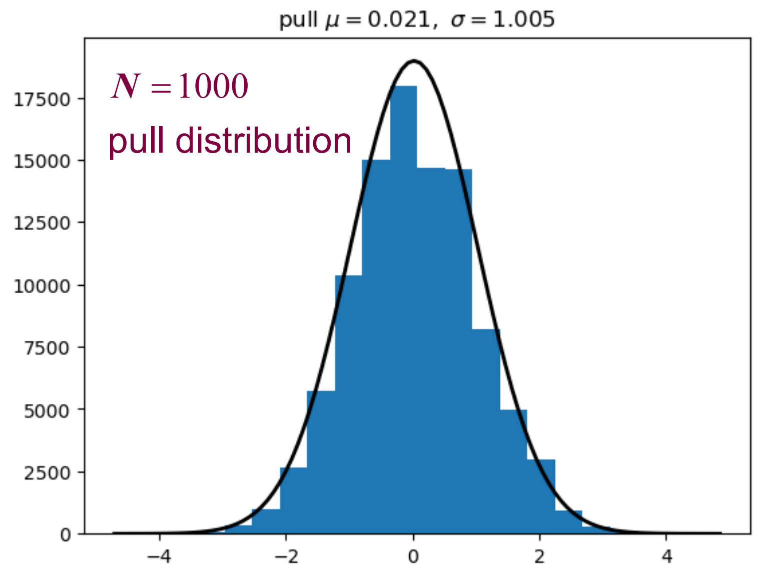
Exercise 1: Determination of π using MC integration [piCalculation.py]

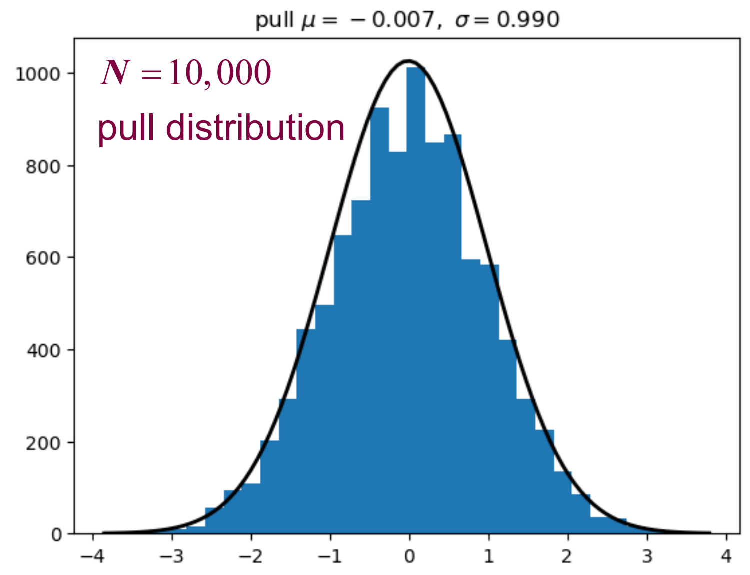






$$\Delta\pi = \frac{1.642}{\sqrt{N-1}}$$





Monte Carlo Probability Density Function (PDF) simulation

- General PDF simulation formulae. Start with the Monte Carlo integration expression:

$$t = \int d\vec{x}\, g(\vec{x}) = \int d\vec{x}\, \rho(\vec{x}) f(\vec{x}) \approx \frac{1}{N}\sum_{i=1}^{N} \frac{g(\vec{x})}{\rho(\vec{x})} = \frac{1}{N}\sum_{i=1}^{N} f(\vec{x}_i) = \frac{s}{N} \quad , \quad \int d\vec{x}\, \rho(\vec{x}) = 1 \quad , \quad s = \sum_i f_i$$

If while performing an MC integration, we wanted in addition to produce an unweighted sample of events reflecting the distribution $g(\vec{x})$ we would only accept events $\vec{x}_i$ pulled from $\rho(\vec{x})$ for which $\frac{g(\vec{x}_i)}{\rho(\vec{x}_i)} > rR_{\max}$ where r is a random number chosen after the event $\vec{x}_i$ has been pulled from $\rho(\vec{x})$ and $R_{\max}$ is the maximum weight for the MC simulation, chosen such that $\frac{g(\vec{x}_i)}{\rho(\vec{x}_i)} < R_{\max}$ for all $\vec{x}_i$. This is called the MC rejection method of producing a unweighted sample. The case of $\rho(\vec{x}) = 1$ will be referred to as the brute force rejection method.

The maximum weight $R_{\max}$ comes up all the time in the production of large MC data sets, since there is tension between low $R_{\max}$ for good efficiency and high $R_{\max}$ to guarantee $\frac{g(\vec{x}_i)}{\rho(\vec{x}_i)} < R_{\max}$ for all $\vec{x}_i$.

In practice a small number of events with $\frac{g(\vec{x}_i)}{\rho(\vec{x}_i)} > R_{\mathrm{ma}}$ is tolerated.

Method of Inverting the Cumulative Density Function (CDF)

The cumulative density function of the pdf $f(x)$ is defined by

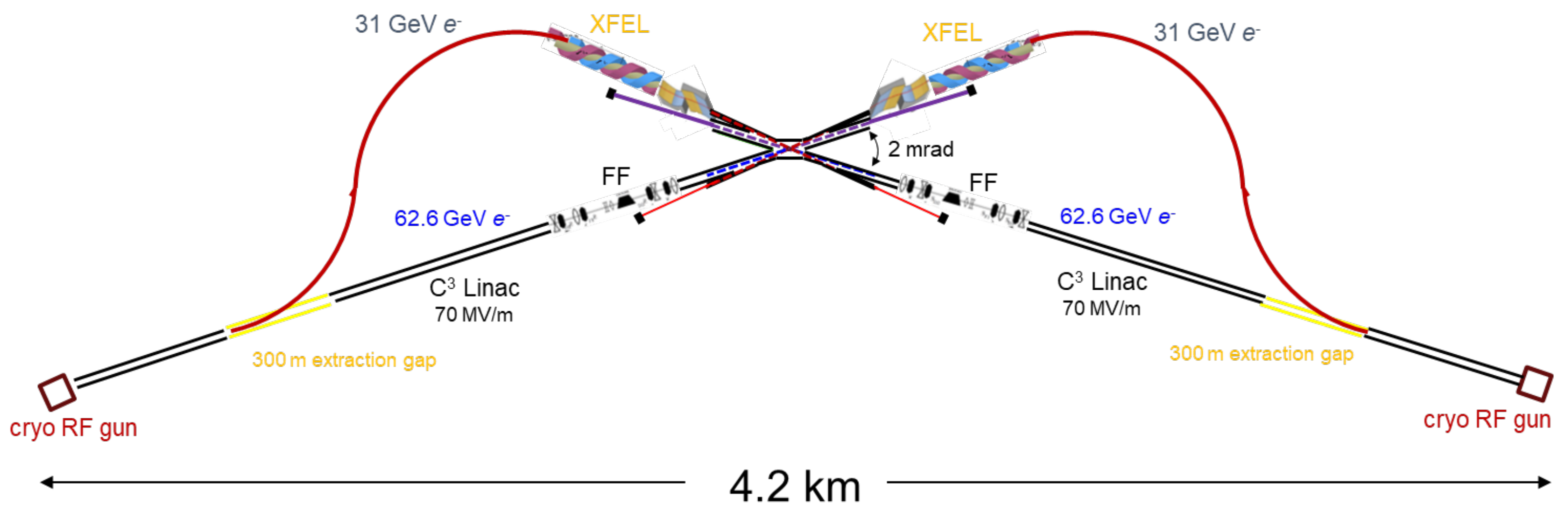
$$r(x) = \left[\int_0^{x\max} dx'\, f(x')\right]^{-1} \int_0^{x} dx'\, f(x') = \left[\int_0^{x\max} dx'\, f(x')\right]^{-1} \left(F(x) - F(0)\right) \qquad \frac{dF}{dx} = f(x)$$

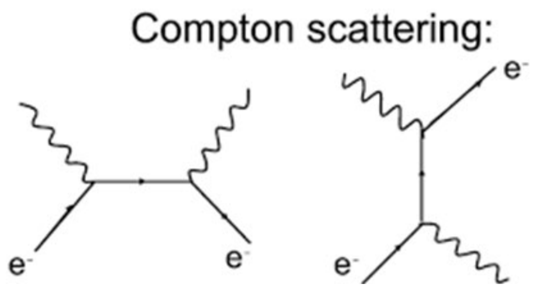
$0 < r(x) < 1$ Changing variables from x to r gives

$\int dx\, f(x) = \int dx \frac{dF}{dx} = \int dF \propto \int dr$ so that the distribution is flat in r. Thus one can immediately map a random number between 0 and 1 directly onto r without having to spend time on a rejection algorithm. This can be extremely efficient. The issue is, given $r(x)$, can we invert the function and solve for x?

XCC: XFEL Compton $\gamma\gamma$ Collider Higgs Factory

XCC s-channel $\gamma\gamma \to H$ @ $\sqrt{s} = 125$ GeV



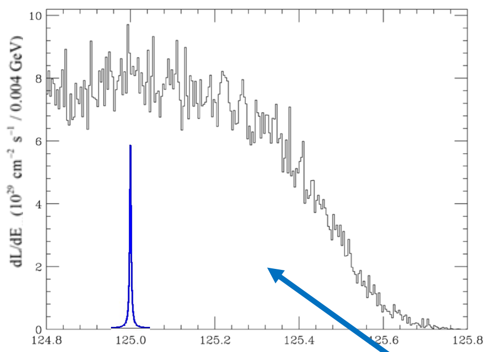


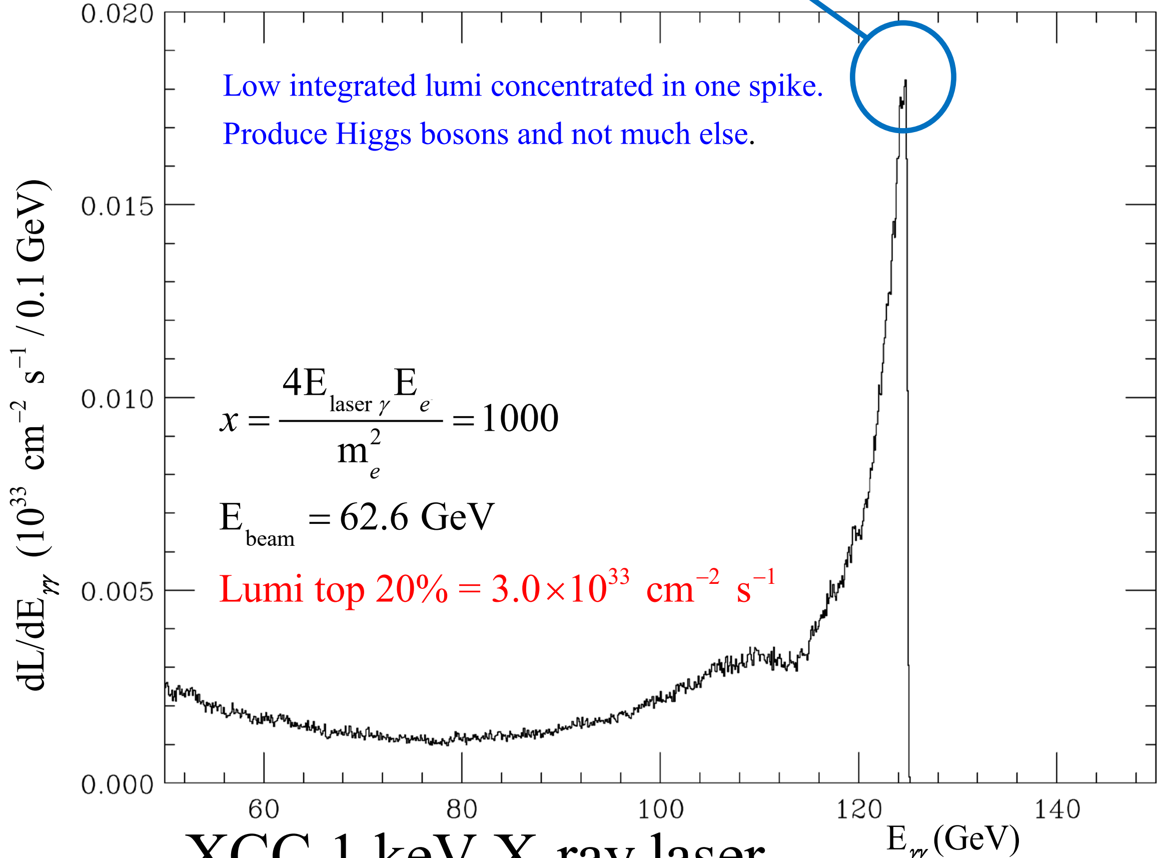
60 μm before the e^-e^- collision point the 62.5 GeV electon beam is hit with a1 keV X-ray beam from an X-ray free electron laser (XFEL). In the Compton scattering process $e^-\gamma \to e^-\gamma$ the electrons and photons basically exchange momenta, so that after the Compton process the photons have 62.5 GeV energy and the electron energy is considerably degraded. The 62.5 GeV photons on each side of the primary collison point annihilate to form a 125 GeV Higgs boson.

One of the issues for γγ colliders is the dependence of the photon energy spectra on the laser energy

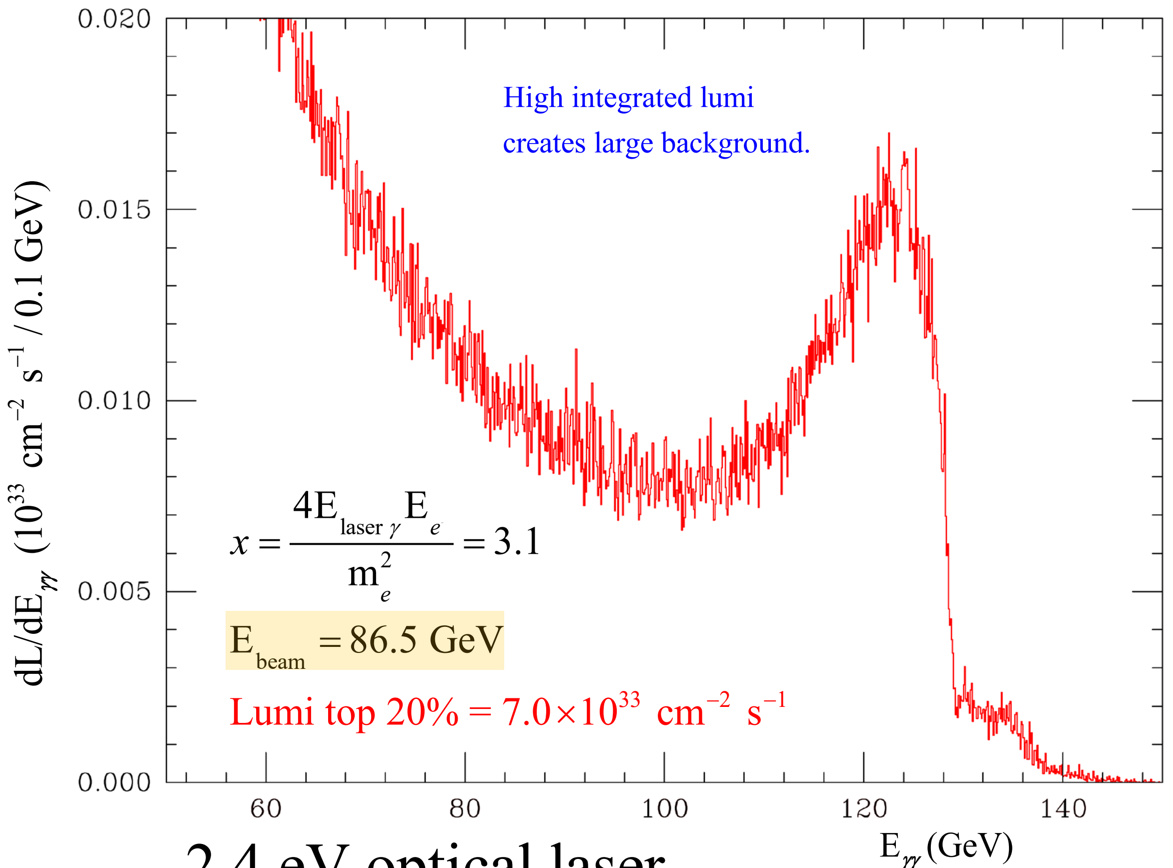
2.4 eV laser

Machine	E_{e^-} (GeV)	Polarization	N_H/yr	N_{Bgnd}/N_H	$N_{pileup/BX}$
XCC	62.8	90% e^-	80,000	170	9.5
OCC	86.5	90% e^-	52,000	1310	50
ILC	125	−80% e^- +30% e^+	98,000	130	1.3
ILC	125	+80% e^- −30% e^+	65,000	50	1.3





XCC 1 keV X-ray laser

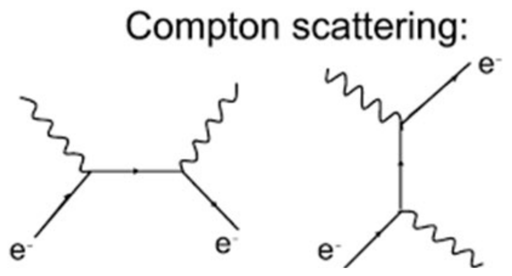


2.4 eV optical laser

Using a CDF to improve MC efficiency

For unpolarized electrons colliding with unpolarized photons through the Compton process $e^-\gamma \to e^-\gamma$

the spectrum of photons with respect to $y = \frac{E_\gamma}{E_{e^-}}$ is given by



$$\frac{d\sigma}{dy} \propto 1 - y + \frac{1}{1-y} - \frac{4y}{x(1-y)} + \frac{4y^2}{x^2(1-y)^2}$$

where $x = \frac{4E_{\text{laser }\gamma}E_{e^-}}{m_e^2}$ is the center of mass energy squared of the e^- - laser γ system in units of the

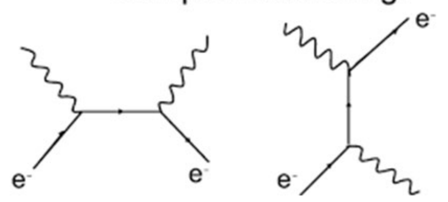
electron mass squared. This expression is dominated by the $\frac{1}{1-y}$ term , so we can use

$\frac{d\sigma}{dy} \approx \frac{1}{1-y}$ as an first approximation to the distribution. It is simple enough that there is

a chance that the integral is invertible. Let's check.

Using a CDF to improve MC efficiency

Compton scattering:



$$\int dy \frac{1}{1-y} = -\ln(1-y).$$

$$r(y) = \left[\int_0^{y_m} dy \frac{1}{1-y}\right]^{-1} [-\ln(1-y)] = \frac{\ln(1-y)}{\ln(1-y_m)}$$ where the maximum γ energy is $y_m = \frac{x}{x+1}$

$\ln(1-y) = r\ln(1-y_m)$ or $1-y = (1-y_m)^r$ or $y = 1-(1-y_m)^r$

$r(y)$ is indeed invertible so we have a very efficient means to generate a $\frac{1}{1-y}$ distribution

Exercise 2: Using a CDF to improve MC efficiency [comptonSpectrum.py]

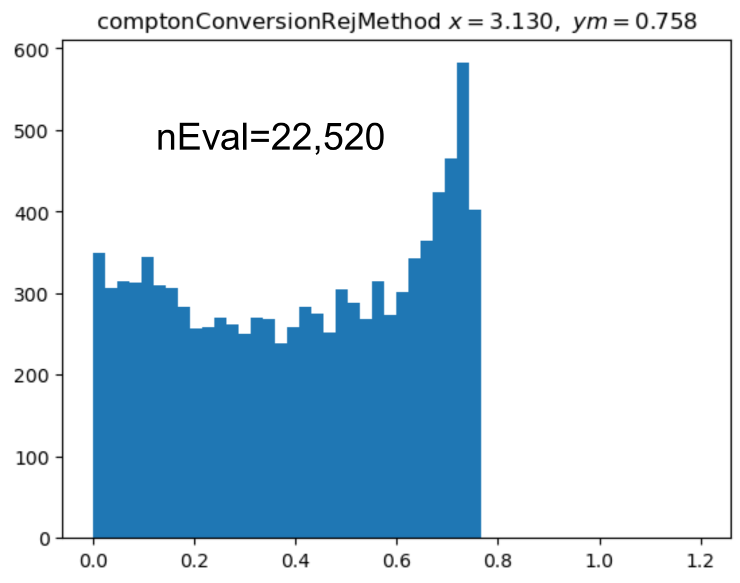
For $t = \int dy \frac{d\sigma}{dy} = \int dy \rho(y) f(y)$ we can try 2 different factorizations of $\frac{d\sigma}{dy} = \rho(y) f(y)$:

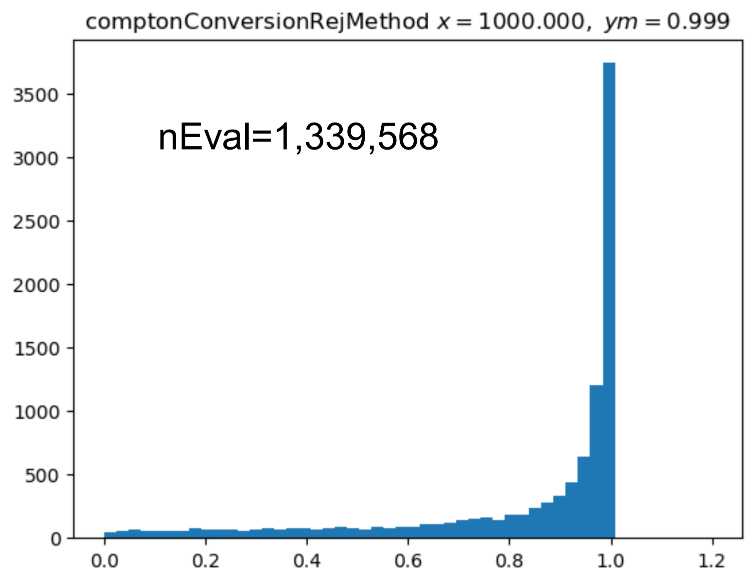
(1) $\rho(y) = 1 \quad f(y) = \frac{d\sigma}{dy}$ (i.e., brute force rejection method)

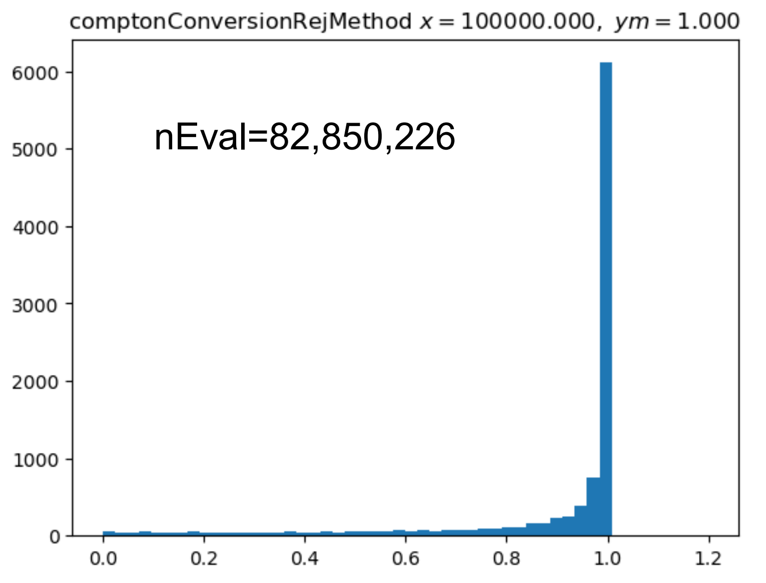
(2) $\rho(y) = \frac{1}{1-y} \qquad f(y) = \frac{d\sigma}{dy}\left[\frac{1}{1-y}\right]^{-1}$ (more refined rejection method starting with an inverted CDF)

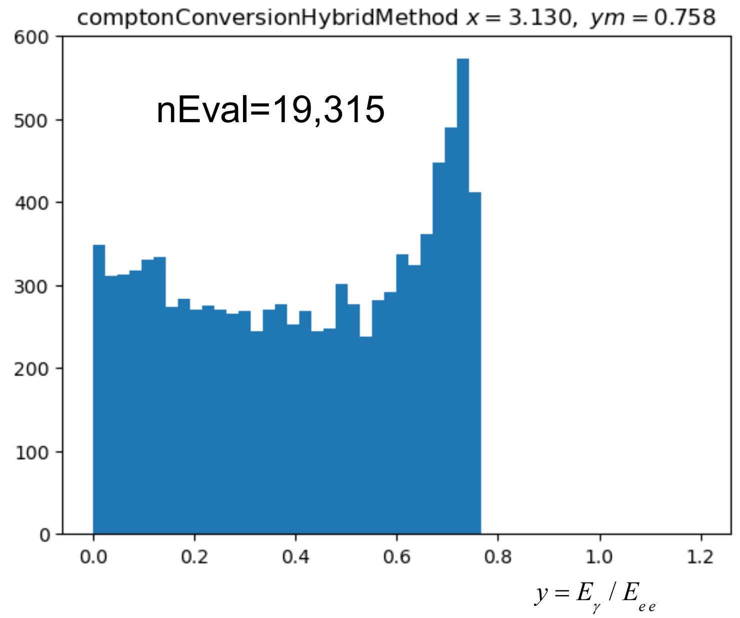
In comptonSpectrum.py compare the number of function evaluation using the brute force method vs. the number of function evaluations using the hybrid CDF/rejection method. Also, note that as the center of mass energy of the electon - laser photon system is increased from x=3.13 to 1000 to 100,000 the photon spectrum becomes more sharply peaked at $y = y_m$. What happens to the difference in function evalulations between brute force rejection and hybrid as the spectrum becomes more sharply peaked?

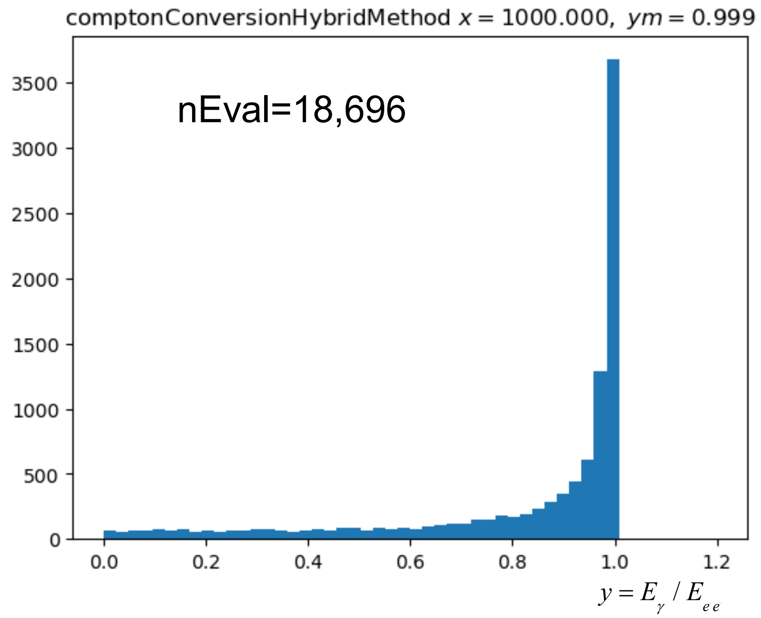
Exercise 2: Using a CDF to improve MC efficiency [comptonSpectrum.py]

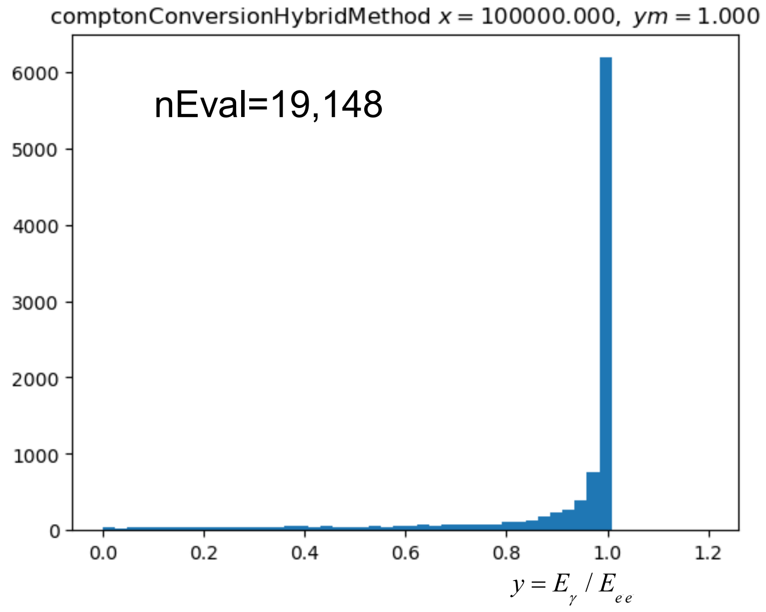












The Vegas Monte Carlo Integration Algorithm

The Vegas algorithm has the following properties:

(a) A reliable error estimate for the integral is readily computed.

(b) The integrand need not be continuous for the algorithm to function and, in particular, step functions pose no problem. Thus integration over hypervolumes of irregular shape is straightforward.

(c) The convergence rate is independent of the dimension of the integral.

(d) The algorithm is adaptive. It automatically concentrates evaluations of the integrand in those regions where the integrand is largest in magnitude.

G. Peter Lepage, 1977: “A New Algorithm for Adaptive Multidimensional Integration “

The Vegas Monte Carlo Integration Algorithm

Before the Vegas MC algorithm was introduced in 1977, all multi-dimensional MC integration programs, even those that were adaptive, required N^n integrand evaluations, where N represents the number of grid points along one axis and n is the dimension of the integral.

The Vegas algorithm avoids the exponential growth in integrand evaluations by using importance sampling.

In importance sampling, the density function $\rho(\vec{x})$ in $t = \int d\vec{x}\, f(\vec{x}) = \int d\vec{x}\, \rho(\vec{x}) \frac{f(\vec{x})}{\rho(\vec{x})}$ is optimized when

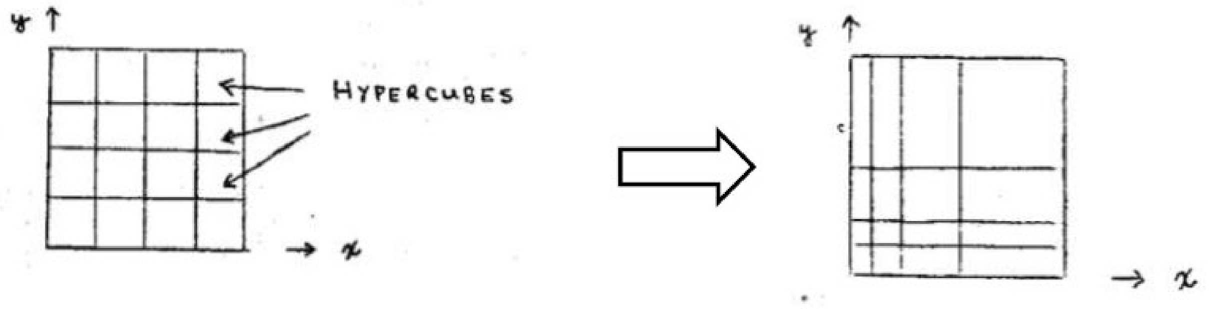
$\rho(\vec{x}) = | f(\vec{x}) | \left[\int d\vec{x} \, | f(\vec{x}) | \right]^{-1}$ i.e. function evaluations are concentrated where the integrand is largest in magnitude, regardless of whether or not the integrand is rapidly changing.

In practice $\rho(\vec{x}) = \rho_1(x_1)\rho_2(x_2)\cdots\rho_n(x_n)$ is a step-function with each axis divided up into its own set of N intervals

Vegas algorithm

Exploratory phase:
• subdivide integration space into rectangular grid
• perform integration in each subspace
• adjust grid according to dominant contributions
• integrate again, approximate optimal

rectangular grid of hypercubes   peak at the origin, adjusted grid



Evaluation phase:
• integrate with high precision and optimized frozen grid
  or efficiently generate events using optimized frozen grid

Vegas algorithm details in 1-d

Consider the integral $t = \int_0^1 dx\, f(x) = \int_0^1 dx\, \rho(x) \frac{f(x)}{\rho(x)}$

Divide the $x-$axis into N equal segments $0 = x_0 < x_1 < \cdots < x_N = 1$, $\Delta x_i = x_i - x_{i-1}$

Define the step-function $\rho(x) = \frac{1}{N\Delta x_i}$ for $x_i - \Delta x_i \le x < x_i$ i=1,2,...,N

First perform Monte Carlo integration with with M integrand evalulation (typically M $\approx$ 2000).

Next, define $m_i = K \frac{\bar{f}_i \Delta x_i}{\sum_j \bar{f}_j \Delta x_j}$ where typically K $\approx$ 2000

$\bar{f}_i = \sum_{x_i - \Delta x \le x < x_i} |f(x)| \propto \frac{1}{\Delta x_i} \int_{x_i - \Delta x}^{x_i} dx\, |f(x)|$ and divide each increment Δx_i into $m_i + 1$ subincrements.

Since the total number of subincrements is now $\gg N$ the subincrements are combined into N groups which serves to define our new set of $0 = x_0 < x_1 < \cdots < x_N = 1$, $\Delta x_i = x_i - x_{i-1}$

Iterate in this manner until integral error is optimized.

To avoid rapid, destabilizing changes in the grid, it is better to damp the subdivisions using

$$m_i = K \left\{ \left[\frac{\bar{f}_i \Delta x_i}{\sum_j \bar{f}_j \Delta x_j} - 1 \right] \log^{-1} \left(\frac{\bar{f}_i \Delta x_i}{\sum_j \bar{f}_j \Delta x_j} \right) \right\}^{\alpha} , \quad 1 < \alpha < 2$$

Implementation Details of Vegas algorithm 2-d

In 2-d we write $t = \int_0^1 dx \int_0^1 dy\, f(x,y) = \int_0^1 dx\, \rho_x(x) \int_0^1 dy\, \rho_y(y) \frac{f(x,y)}{\rho_x(x)\rho_y(y)}$

Divide $x-$axis into N equal segments $0 = x_0 < x_1 < \cdots < x_N = 1$, $\Delta x_i = x_i - x_{i-1}$

Divide $y-$axis into N equal segments $0 = y_0 < y_1 < \cdots < y_N = 1$, $\Delta y_i = y_i - y_{i-1}$

$\rho_x(x) = \frac{1}{N\Delta x_i}$ for $x_i - \Delta x_i \le x < x_i$, i=1,2,...,$N$

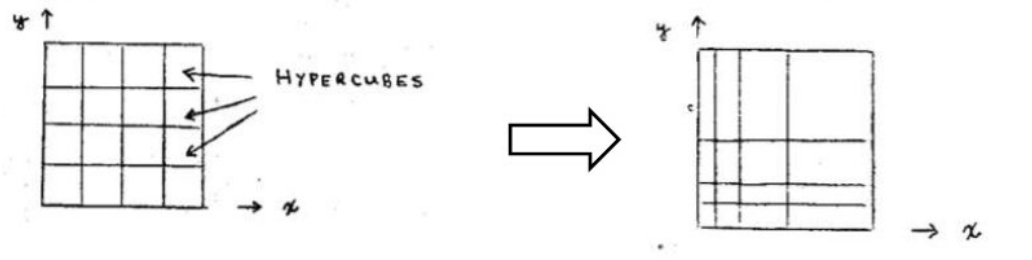
$\rho_y(y) = \frac{1}{N\Delta y_i}$ for $y_i - \Delta y_i \le y < y_i$, i=1,2,...,N

1-d algorithm applied along $x-$axis with

$$(\bar{f}_i)^2 = \sum_{x_i-\Delta x\le x<x_i} \sum_{0<y<1} \frac{f^2(x,y)}{\rho_y^2(y)} \propto \frac{1}{\Delta x_i} \int_{x_i-\Delta x_i}^{x_i} dx \int_0^1 dy \frac{f^2(x,y)}{\rho_y(y)}$$

1-d algorithm applied along y$-$axis with

$$(\bar{f}_i)^2 = \sum_{y_i-\Delta y\le y<y_i} \sum_{0<x<1} \frac{f^2(x,y)}{\rho_x^2(x)} \propto \frac{1}{\Delta y_i} \int_{y_i-\Delta y_i}^{y_i} dy \int_0^1 dx \frac{f^2(x,y)}{\rho_x(x)}$$



Use of Vegas integration in MC event sample generation

Upon completion of the Vegas Monte Carlo integration, the step-function probability density function $\rho(\vec{x}) = \rho_1(x_1)\rho_2(x_2)\cdots\rho_n(x_n)$ should be a good representation of the function $f(\vec{x})$, and can therefore be used to efficiently generate events according to the function $f(\vec{x})$. i.e, one pulls events according to $\rho(\vec{x}) = \rho_1(x_1)\rho_2(x_2)\cdots\rho_n(x_n)$ and then applies a rejection algorithm using $\dfrac{f(\vec{x})}{\rho(\vec{x})}$, which shouldn't deviate too much from a constant value if $\rho(\vec{x})$ is a good representation of $f(\vec{x})$. Most if not all particle physics event generators utilize this technique.

Exercise 3: Vegas MC simulation of 2-d γγ lumi distribution [vegasCompton.py]

For unpolarized electrons colliding with unpolarized photons through the Compton process $e^- \gamma \to e^- \gamma$

the 2-d differential luminosity with respect to $y_1 = \frac{E_{\gamma 1}}{E_{e^-}}$ and $y_2 = \frac{E_{\gamma 2}}{E_{e^-}}$ is given by

$$\frac{d^2 L}{dy_1 dy_2} \propto \left[1 - y_1 + \frac{1}{1-y_1} - \frac{4y_1}{x(1-y_1)} + \frac{4y_1^2}{x^2(1-y_1)^2}\right]\left[1 - y_2 + \frac{1}{1-y_2} - \frac{4y_2}{x(1-y_2)} + \frac{4y_2^2}{x^2(1-y_2)^2}\right]$$

where $x = \frac{4E_{\text{laser}\,\gamma} E_{e^-}}{m_e^2}$ is the center of mass energy squared of the e^- - laser γ system in units of the electron mass squared.

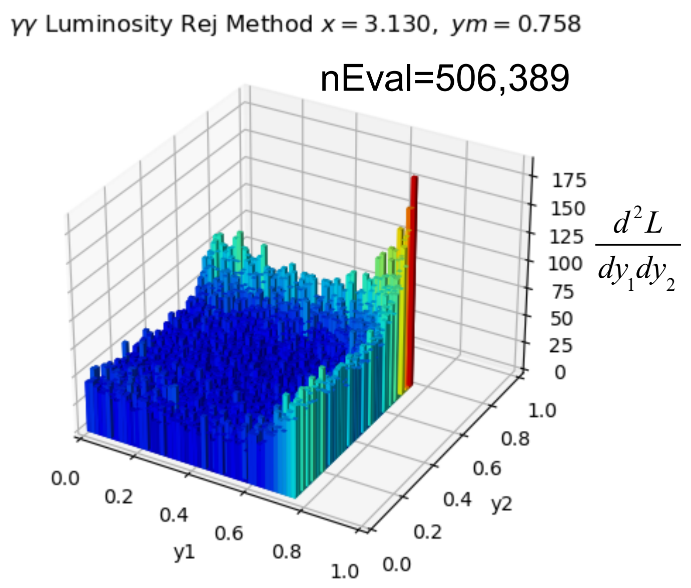
In vegasCompton.py the 2-d differential luminosity distribution is created in two ways:

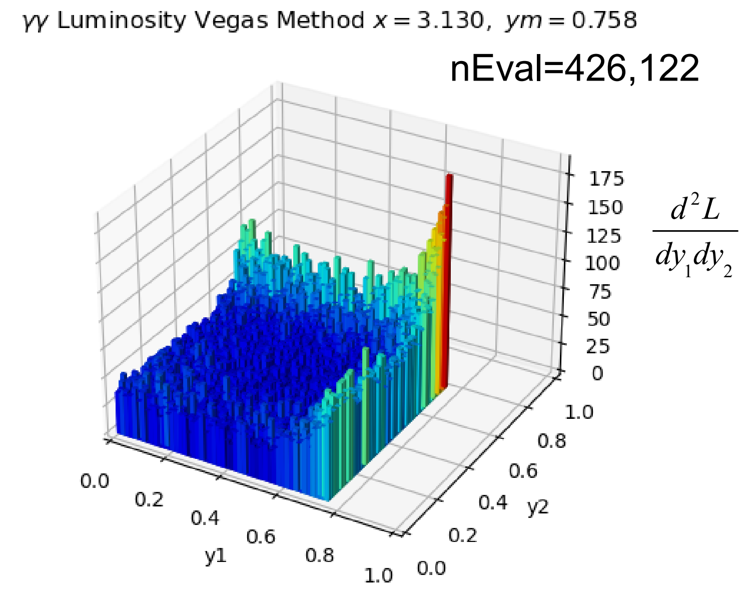
(1) $\rho(y_1, y_2) = 1 \quad f(y_1, y_2) = \frac{d^2 L}{dy_1 dy_2}$ (i.e., brute force rejection method)

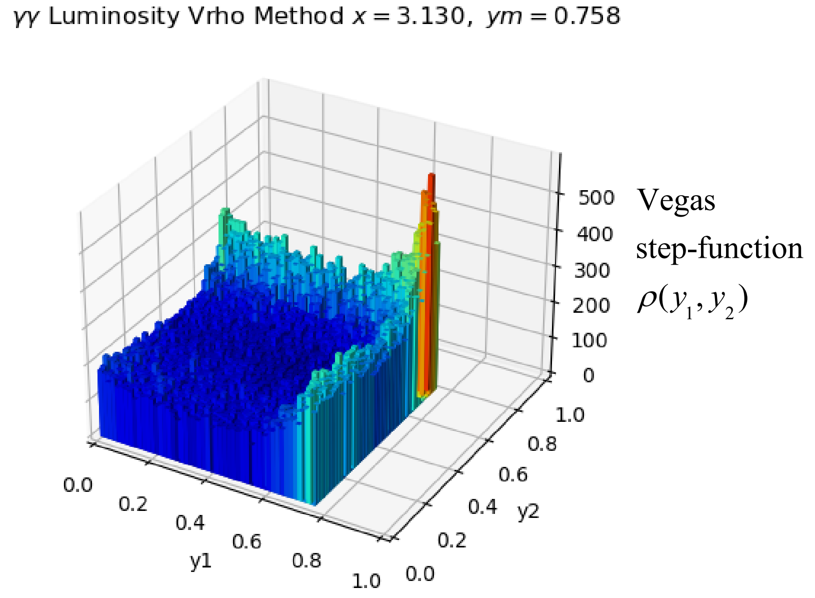
(2) $\rho(y_1, y_2) =$ final Vegas step-function grid after integration $f(y_1, y_2) = \frac{d^2 L}{dy_1 dy_2}[\rho(y_1, y_2)]^{-1}$

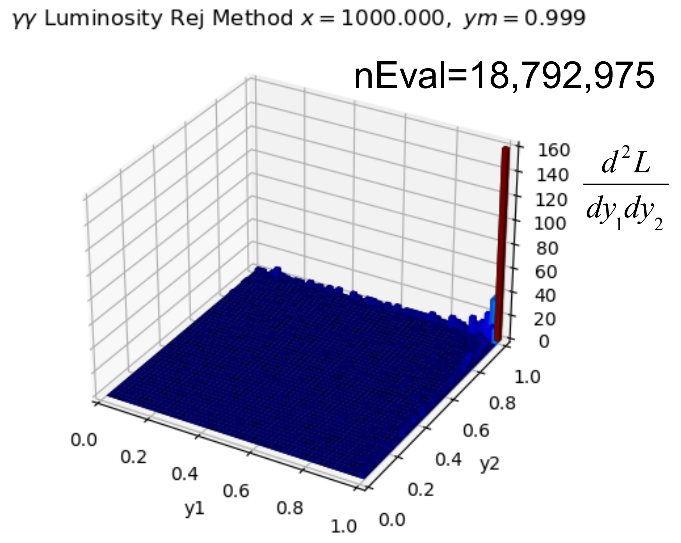
Distributions are created for two different values of $x = 3.13$ and $x = 1000.$ Also, the Vegas step-function grid $\rho(y_1, y_2)$ is plotted (files named vrhoCompton_vrho...)

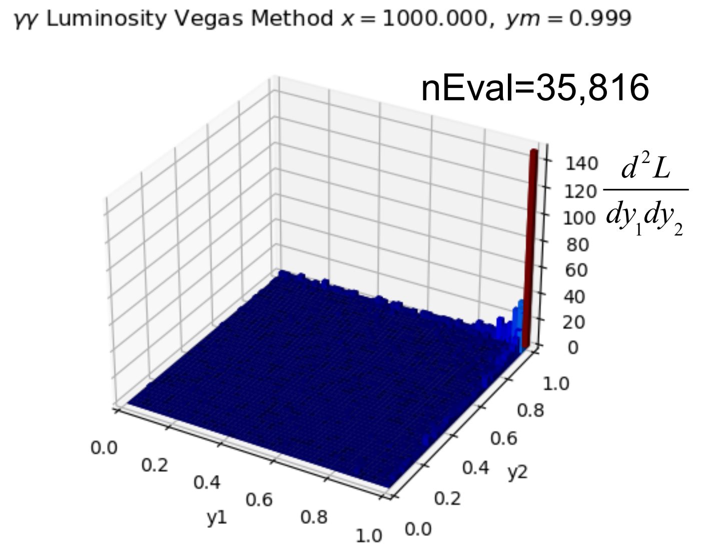
Exercise 3: Vegas MC simulation of 2-d $\gamma\gamma$ lumi distribution [vegasCompton.py]

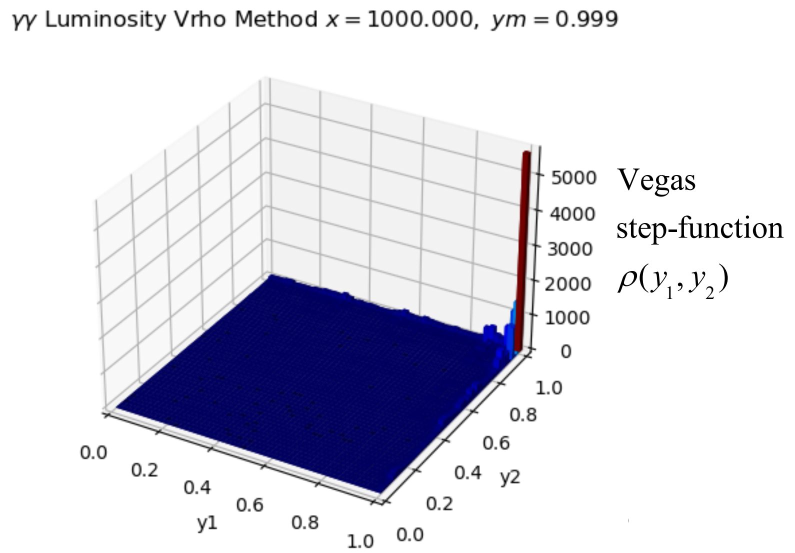












Homework/Exam: Modify vegasCompton.py to produce a double Gaussian distribution

Produce a 10,000 event distribution of $f(x,y) = A_1 e^{-\frac{1}{2}\left(\frac{(x-\mu_{1x})^2}{\sigma_{1x}^2}+\frac{(y-\mu_{1y})^2}{\sigma_{1y}^2}\right)} + A_2 e^{-\frac{1}{2}\left(\frac{(x-\mu_{2x})^2}{\sigma_{2x}^2}+\frac{(y-\mu_{2y})^2}{\sigma_{2y}^2}\right)}$ over the range 0< $x, y < 5$ by modifying vegasCompton.py . Produce the distribution using the same two methods we used in vegasCompton.py:

(1) $\rho(x,y) = 1 \quad f(x,y) = A_1 e^{-\frac{1}{2}(...)} + A_2 e^{-\frac{1}{2}(...)}$ (i.e., brute force rejection method)

(2) $\rho(x,y) =$ final Vegas step-function grid after integration $f(x,y) = \left[A_1 e^{-\frac{1}{2}(...)} + A_2 e^{-\frac{1}{2}(...)} \right] [\rho(x,y)]^{-1}$.

Compare the number of function evaluations required for these two methods.

The following parameters should be used:

$A_1 = 1.0 \qquad \mu_{1x} = 2.5 \qquad \sigma_{1x} = 1.0 \qquad \mu_{1y} = 2.5 \qquad \sigma_{1y} = 1.0$

$A_2 = 4.0 \qquad \mu_{2x} = 2.5 \qquad \sigma_{2x} = 0.1 \qquad \mu_{2y} = 2.5 \qquad \sigma_{2y} = 0.1$

Homework/Exam: Modify vegasCompton.py to produce a double Gaussian distribution

Notes to minimize the number of modifications:

Delete the method $dsig\_dy(xnumber, yy)$ and directly code the double Gaussian function in the method $dlum\_dy1dy2(xnumber, yy1, yy2)$

The parameter $xnumber$ has no meaning in this exercise. Instead of eliminating it as a function argument, just leave it in the argument lists and loop over xnumber=3.13 only.

Set $y_m = 5$ so that you produce a distribution for $0 < x, y < 5$

The maximum values of the function are near $x, y = 2.5$ instead of $x, y = y_m$, so you will have to modify the calculation of the maximum weight.

The CAIN Beam-Beam Simulation Monte Carlo

- Stand-alone Monte Carlo program for simulations of beam–beam interactions involving high-energy electrons, positrons and photons

.
- Written by K. Yokoya et al., KEK, Japan, 1984–2011.

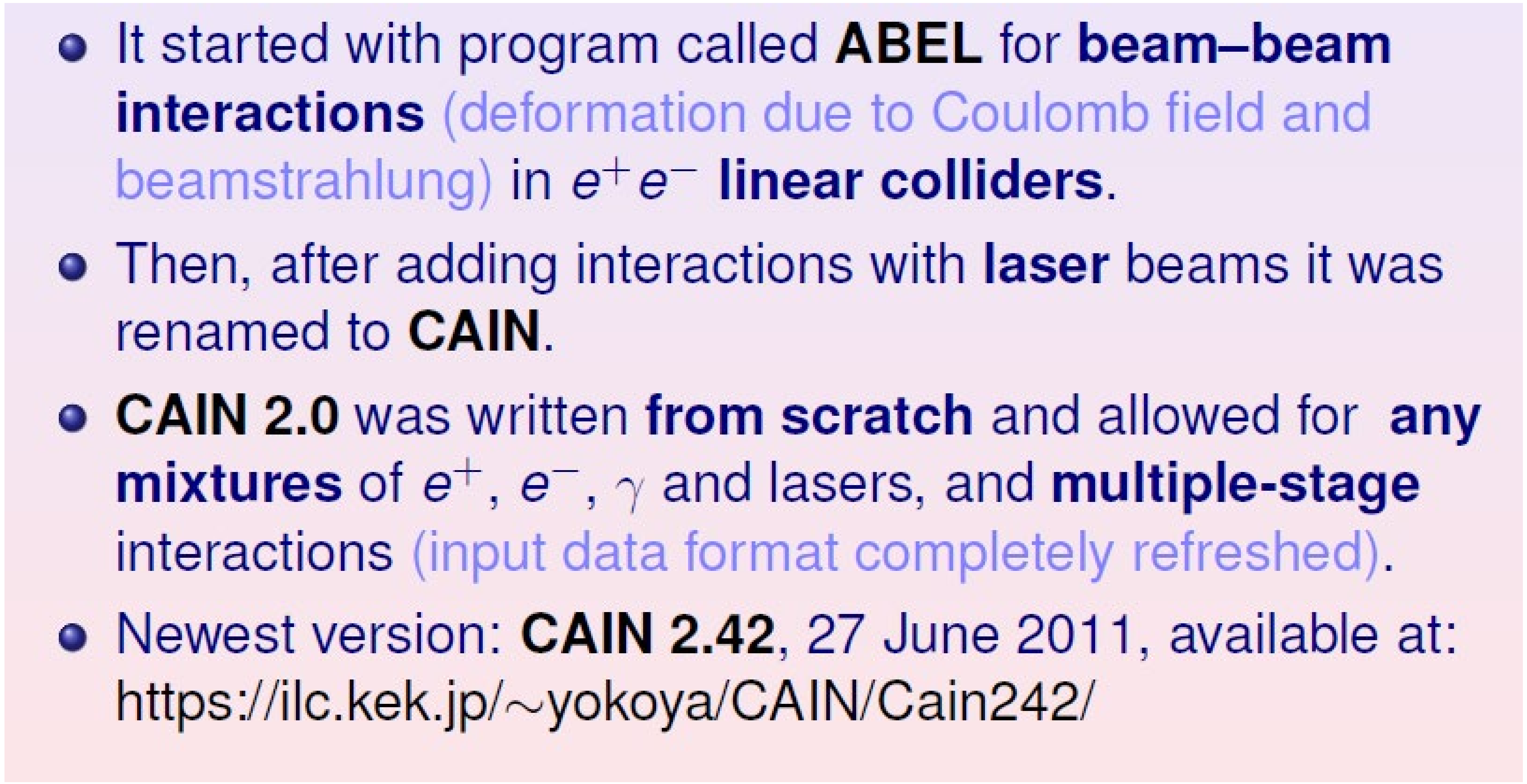
- Code is a mixture of FORTRAN 77 and FORTRAN 90/95, 45 000 lines in 400 files

- Code not documented, comments in code scarce. But very good user manual

- Dedicated, elaborate meta-language for defining Input (65 pages of description in User Manual).

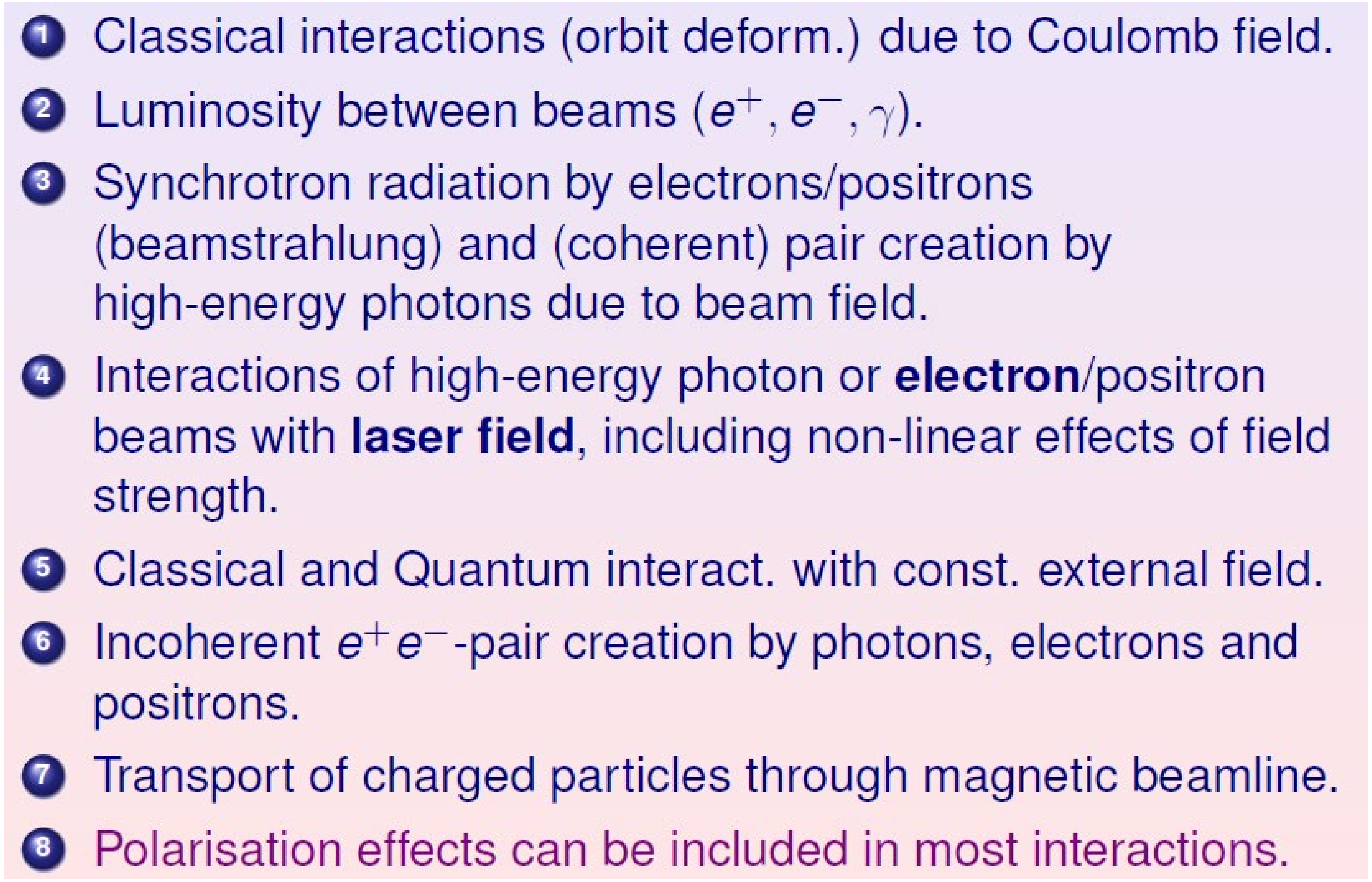
- Output in form of text files with all particle information and TopDrower histograms

CAIN History



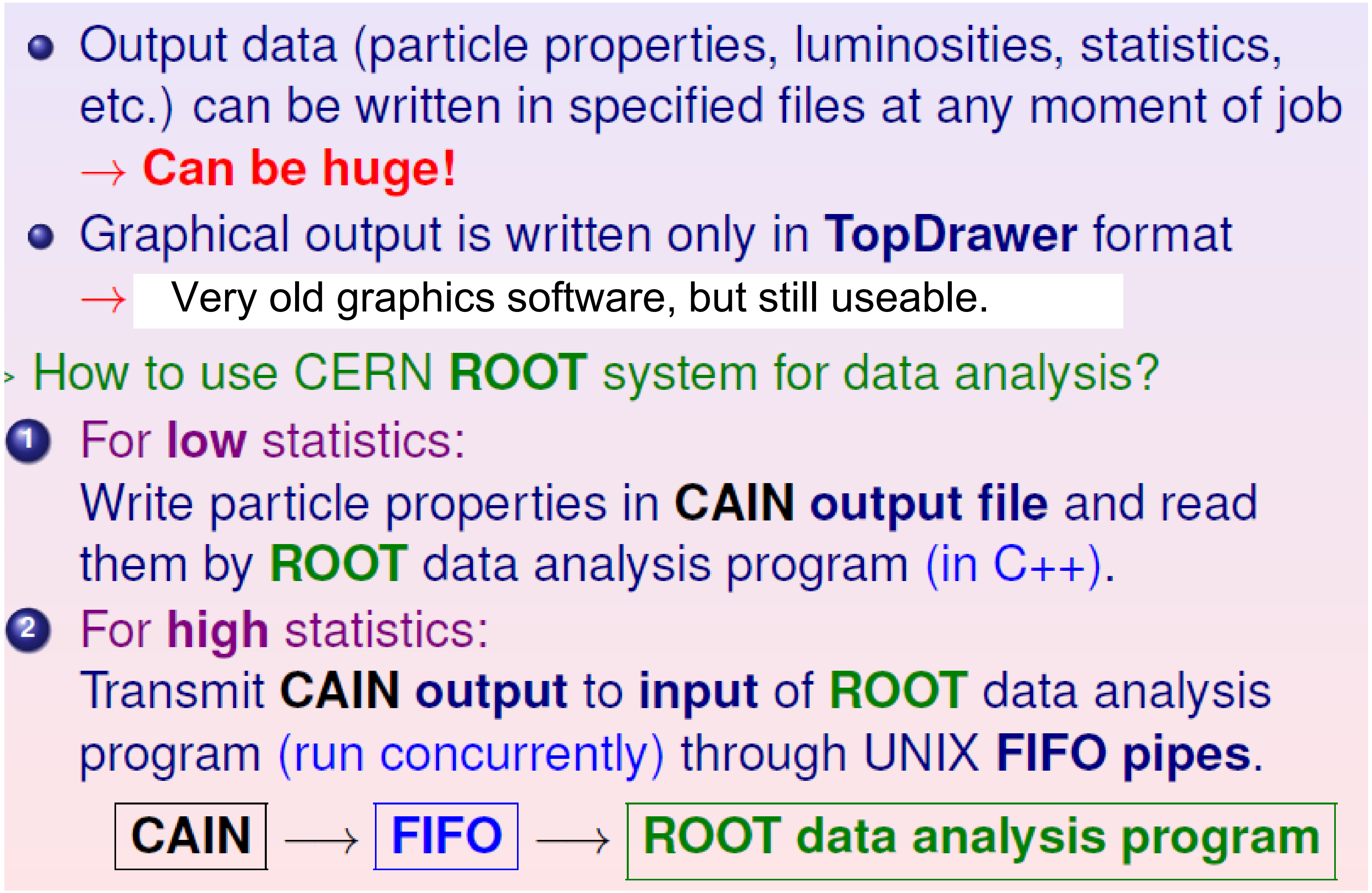
- It started with program called **ABEL** for **beam–beam interactions** (deformation due to Coulomb field and beamstrahlung) in e^+e^- **linear colliders**.
- Then, after adding interactions with **laser** beams it was renamed to **CAIN**.
- **CAIN 2.0** was written **from scratch** and allowed for **any mixtures** of e^+, e^-, γ and lasers, and **multiple-stage** interactions (input data format completely refreshed).
- Newest version: **CAIN 2.42**, 27 June 2011, available at: https://ilc.kek.jp/~yokoya/CAIN/Cain242/

CAIN Physical Processes



1. Classical interactions (orbit deform.) due to Coulomb field.
2. Luminosity between beams (e^+, e^-, γ).
3. Synchrotron radiation by electrons/positrons (beamstrahlung) and (coherent) pair creation by high-energy photons due to beam field.
4. Interactions of high-energy photon or **electron**/positron beams with **laser field**, including non-linear effects of field strength.
5. Classical and Quantum interact. with const. external field.
6. Incoherent e^+e^--pair creation by photons, electrons and positrons.
7. Transport of charged particles through magnetic beamline.
8. Polarisation effects can be included in most interactions.

CAIN Output



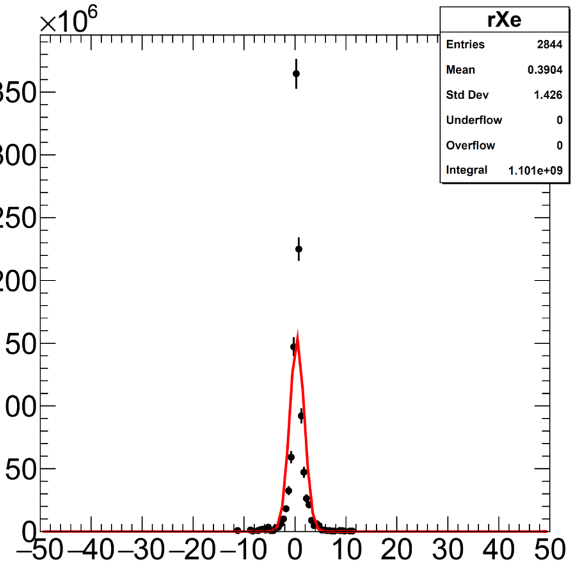
- Output data (particle properties, luminosities, statistics, etc.) can be written in specified files at any moment of job
  $\rightarrow$ **Can be huge!**
- Graphical output is written only in **TopDrawer** format
  $\rightarrow$ Very old graphics software, but still useable.

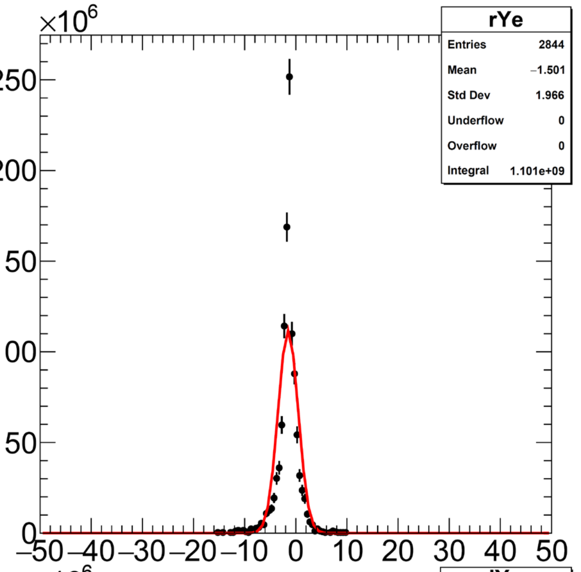
How to use CERN **ROOT** system for data analysis?

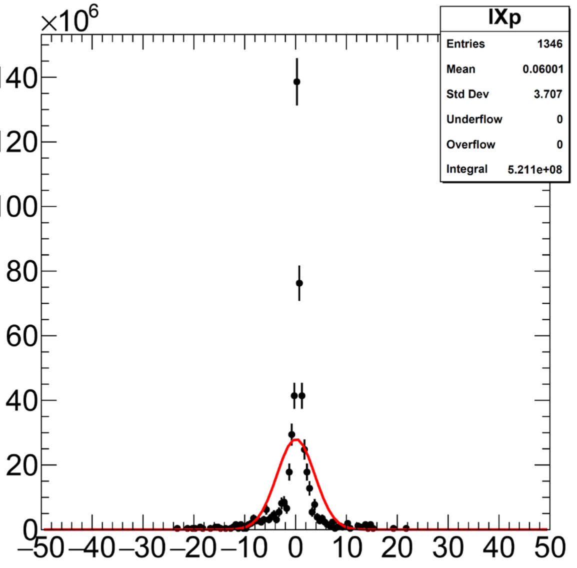
1. For **low** statistics:
   Write particle properties in **CAIN output file** and read them by **ROOT** data analysis program (in C++).
2. For **high** statistics:
   Transmit **CAIN output** to **input** of **ROOT** data analysis program (run concurrently) through UNIX **FIFO pipes**.

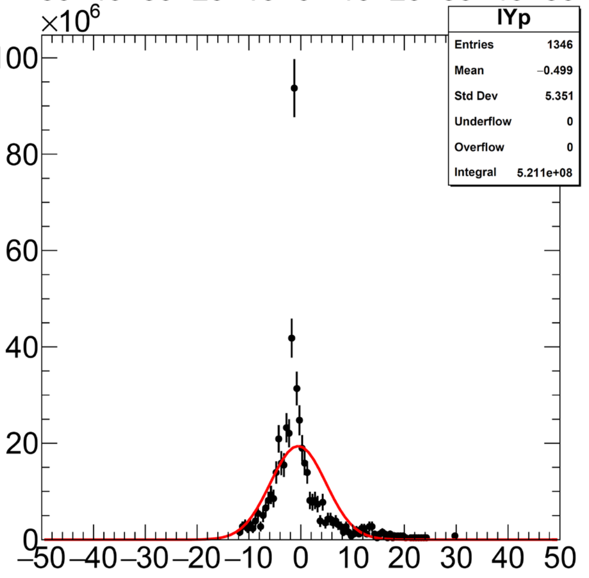
$$\begin{tikzcd} \boxed{\textbf{CAIN}} \arrow[r] & \boxed{\textbf{FIFO}} \arrow[r] & \boxed{\textbf{ROOT data analysis program}} \end{tikzcd}$$

Surprise in collision of unscattered 70 GeV electron beam with 70 GeV photon + electron beam: pinching instead of anti-pinching









No one predicted this. Would not have been noticed without the beam beam Monte Carlo program.

This pinching creates very high fields $\Rightarrow$ prob. to radiate γ in time slice >1 and CAIN program terminates