Methods to Improving Monte Carlo

Brendon Madison

University of Kansas

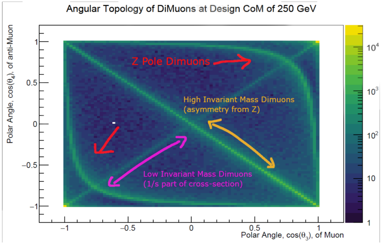
March 25, 2023

Contents

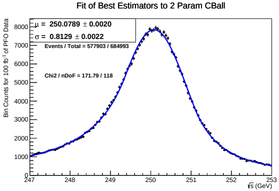
Background: What do I do?

- HEP Research
  - Future colliders: ILC, FCC-ee, ReLiC
  - Monte Carlo Generation of Events, Detector Response
  - Perfecting energy, momentum precision
  - Energy Recovery Linacs (ERLs)

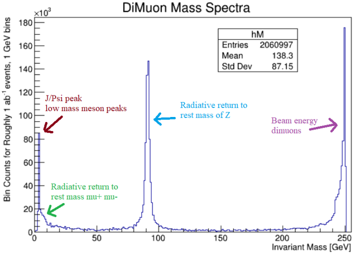
- Dimuon Angular Spectra :



- Energy Calibration:

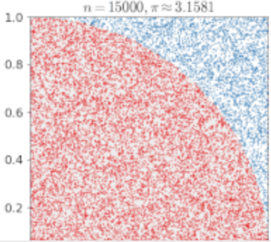


- Using the DiMuon Mass Spectra!:



Current Talk – Why should you care about this?

- 1.) Understanding Monte Carlo
- 2.) Understanding how to Monte Carlo FAST
- 3.) Understanding how to Math
- Observations:
  - Time for breakthroughs in physics is longer
  - Physics is expensive, impact on environment
  - Understanding from first principles is hard
  - To quote a KU CS Professor recalling a talk with a member of the White House:
  - "Machine learning is not a solution. It is successive approximations designed to fool (less intelligent) people that it is a solution."
  - What to do?
- Solution: Fast, precise, computations with quick design turn around
- Monte Carlo can be this!



Current Talk – Funny view of the “future”

- Just get ChatGPT to write your Monte Carlo...

b write a program in c++ that can generate a gaussian random number

Here's a C++ code that generates a Gaussian random number using the Box-Muller transform:

cpp Copy code

```cpp
#include <iostream>
#include <cmath>
#include <random>

double generateGaussian(double mean, double stddev) {
    static std::random_device rd;
    static std::mt19937 gen(rd());
    std::normal_distribution<double> dist(mean, stddev);
    return dist(gen);
}

int main() {
    double mean = 0.0;
    double stddev = 1.0;
    double gaussian = generateGaussian(mean, stddev);
    std::cout << "Gaussian random number: " << gaussian << std::endl;
    return 0;
}
```

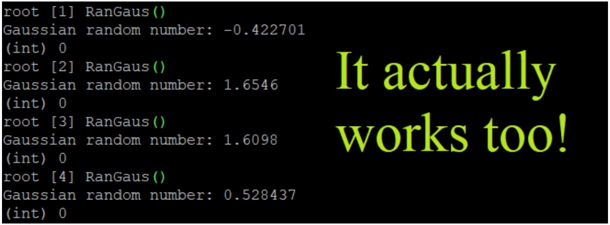
Explanation:

The `generateGaussian` function takes in the mean and standard deviation of the Gaussian distribution and returns a single Gaussian random number. It uses the `std::normal_distribution` class from the `<random>` header to generate random numbers from a Gaussian distribution with the specified mean and standard deviation. The function also uses a static `std::mt19937` random number generator initialized with a `std::random_device` to ensure the generated numbers are truly random.

The `main` function simply calls `generateGaussian` with a mean of 0 and a standard deviation of 1, and prints the generated Gaussian random number to the console.

Current Talk – Funny view of the "future"

- Then compile and run the code using ROOT on KU HPC...



- But this method isn't the best for reasons that will be shown later!

- Using randomness, random variables, to solve mathematical or statistical or numerical problems.
- No “one size fits all” approach.

NOTABLE USES:

- Integration of otherwise unintegrable (difficult) functions
- Optimization (fitting) of large parameter space (difficult) problems – MCMC fitting
- Convolution of functions
- Having data and model driven simulations –Can be semi-analytical or entirely empirical

Gotta go fast p1 – Precision

- Speed and precision are coupled in computation (MC)
- Ex: Time to loop using different variables:
- Integer (16 bit) = 1 N
- Float (32 bit) = 1.2 N
- Double (64 bit) = 2.3 N
- Long (128 bit) = 5.6 N
- Conclusion: Understand your precision needs.
- Find ways to do your large loops or sampling using integers.

Type	Sign	Exponent	Significand field	Total bits	Exponent bias	Bits precision	Number of decimal digits
Half (IEEE 754-2008)	1	5	10	16	15	11	~3.3
Single	1	8	23	32	127	24	~7.2
Double	1	11	52	64	1023	53	~15.9
x86 extended precision	1	15	64	80	16383	64	~19.2
Quad	1	15	112	128	16383	113	~34.0

- Ex.
  Your best measured constant needed is Z Boson mass (known to 6 digits) ... Can get away with Float(Single) precision as it is good to 7.2 digits.

Gotta go fast p2 – Sorting

- Suppose you have the following data (made concise for presentation):

Example Data					
Energy	Px	Py	Pz	Mass	Charge
100.4	0.00	0.01	100.4	m_e	1.0
100.1	0.05	0.02	100.1	m_e	1.0
100.3	0.01	0.02	100.3	m_e	1.0
100.2	0.02	0.00	100.2	m_e	1.0

- You want to randomly sample these as rows...
- So you choose the "Energy" column and randomly choose one
- WHY IS THIS SLOW (BAD)?

Gotta go fast p2 – Sorting

- Remove repeated columns, create an index (SORT) following an ORDERED column

Example Data				
Energy	Px	Py	Pz	Index
100.4	0.00	0.01	100.4	4
100.1	0.05	0.02	100.1	1
100.3	0.01	0.02	100.3	3
100.2	0.02	0.00	100.2	2

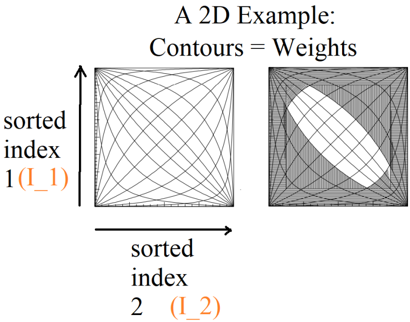
- Generate a random permutation of [1...4] instead of asking for one of [100.4,100.1,100.3,100.2]
- This change may seem minor to yourself BUT
  Integer permutation faster than random choice Float/Double

Gotta go fast p3 – Fitted and Sorted Sampling

- What if you have some weighting for sampling? Then how to proceed?
- **Two questions** to ask:
  –Do you know the function for the weighting that depends on one of the variables e.g. $F_w(E)$?
  –If no can you fit a weighting function across the entire range?
- If yes to the first question then use the weighting function (F_w) that you already know.
- If no and then yes then fit F_w across your data range.
- If no to both then try to fit a weighting function in various sub-ranges of your data. Then you will have $F_{w,1}$,$F_{w,2}$... for all the sub-ranges.
- This method doesn't help if you go to $F_{w,N}$ for N data points.

Gotta go fast p3 – Fitted and Sorted Sampling

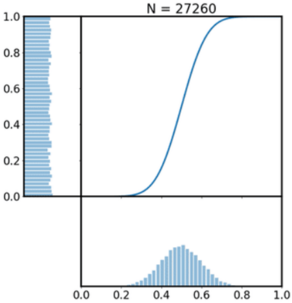
- Dynamically cuts sampling indices, weights, using the fit and sorting



- Have $F_w(I_1, I_2)$; randomly choose a F_w range thus reduce range on I_1,I_2 to only randomly sample the desired sub-range.
- Normalize weights in range too so computer does less operations before successful pass in sub-range.
- Ex. MC with 100k database generating 1000 events using Fitted Sorting reduced execution from 16:30 min to 4:05 min.

Gotta go fast p4 – Analytic Derivation of Random Variates

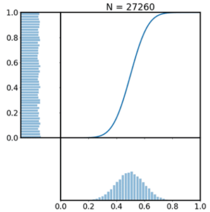
- Earlier ChatGPT used Box-Muller to get Gaussian Random Variate
- This is disadvantageous for two reasons:
  1.) It is slow
  2.) It is a numerical approximation
- HOW TO GO FASTER?
  –TIME TO USE MATH
- Use INVERSE TRANSFORM SAMPLING



Gotta go fast p4 – Analytic Derivation of Random Variates

- Inverse Transform Sampling , How to Math
- Cumulative Distribution Function (CDF) of distribution $= F(x)$
- Set $F(x)$ equal to uniform random variate R_U
- Solve for x
- Ex. Gaussian $F_g(x) = R_U = \frac{1}{2}\left[1 + \text{erf}\left(\frac{x-\mu}{\sqrt{2}\sigma}\right)\right]$

$$2R_U - 1 = \text{erf}\left(\frac{x-\mu}{\sqrt{2}\sigma}\right)$$
$$\text{erf}^{-1}(2R_U - 1) = \frac{x-\mu}{\sqrt{2}\sigma}$$
$$x = \sqrt{2}\sigma[\text{erf}^{-1}(2R_U - 1)] + \mu$$



- Test this!

```
root [1] GausTime()
Time to generate 1M Gaussian Variates: 0.278119
(int) 0
root [2] .x RanGaus.C
1.67974e+09
Time for 1M Gaussian Variates: 0.466704
(int) 0
```

ROOT , Inverse Sampling Method

ChatGPT method is ~2x time

Finishing this off – Do other distributions

- You can do this with tons of distributions!
- Eight examples in https://github.com/BrendonMadison/InverseSamplingExamples
- Can even make your own distributions (random variates)
- For example – Skewed Arc Sine (useful in parity violating particle events) like dimuon production!
- PDF : $P(x) = \frac{1}{\pi}[x(1-x)]^{f(A_{LR})}$
- Where $f(A_{LR})$ is a function dependent on the Left-Right asymmetry of the events
- Example output: