PHY407: Computational Physics
Fall, 2017

Lecture 10: Random Processes and Monte-Carlo Integration

Summary & Status

- ☑ Weeks 1-3: Programming basics, numerical errors, numerical integration and differntiation.
- ☑ Weeks 4-5: Solving linear & nonlinear systems and Fourier transforms.
- ☑ Week 6: ODEs Part 1: RK4, Leapfrog, Verlet, adaptive time stepping; customizing python output
- ☑ Week 7: ODEs Part 2: Bulirsch-Stoer, Boundary Value Problems/shooting,
- ☑ Week 8: PDEs Part 1: Elliptic equation solvers, leapfrog time stepping, FTCS
- ☑ Week 9: PDEs Parts 2
  - Crank-Nicholson and Spectral Methods
- ❑ Weeks 10-11: Random numbers & Monte Carlo methods

PHY407: Computational Physics
Fall, 2017

Lecture 10: Random Processes and Monte-Carlo Integration

- Random number generation, non-uniform distributions
- Monte-Carlo integration: three methods

Why do we need random numbers?

Why do we need random numbers?

- For randomly sampling a domain.
- As input to Monte Carlo integration.
- As input to Monte Carlo simulation.
- Stochastic algorithms.
- Cryptography.

How can a computer generate random numbers?

How can a computer generate random numbers?

- It can't! Computers are generally not set up to do anything randomly.
- 2 options:
  - Find a physical process that actually is random, have computer measure that process to provide a random number
  - Use an algorithm to generate a sequence of numbers that approximates the properties of random numbers.
    - *Pseudorandom number generator* (PRNG) or *Deterministic Random Bit Generator* (DRBG)

PRNG

- E.g. 'Linear Congruent' RNG:

$$x_{i+1} = (ax_i + c) \bmod m$$

- You can reproduce a PRNG sequence if you start with the same seed value.

```
from numpy.random import seed, random
s = seed(4219)
print 'seed1: random',random()
s=seed(4220)
print 'seed2: random',random()
s = seed(4219)
print 'seed1: random',random()
```

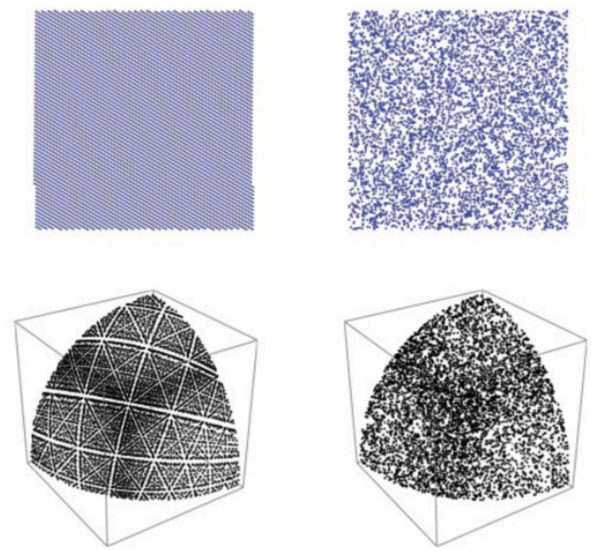
```
seed1: random [ 0.13296973  0.77640973  0.50328907]
seed2: random [ 0.02810287  0.50073785  0.0342652 ]
seed1: random [ 0.13296973  0.77640973  0.50328907]
```

- Or you can let the computer pick it (e.g. pick system time).
- You can pick many seeds to obtain different sequences (e.g. good for cryptography).

Finding the right PRNG

- We want to avoid correlations which the LCG features.
- There are statistical tests on PRNGs to make sure they produce good statistics.
- Python uses the Mersenne Twister (http://dl.acm.org/citation.cfm?doid=272991.272995)

LCG (left), Mersenne Twister (right)



SciNet Computational Science Course

Python's PRNG

- random.py functions:
  - `random()`: Random float uniformly distributed in [0,1)
  - `randrange()`: Random integer in range [m,n-1]
- Uniform distribution: all values have equal probability of being selected.
- You can generate sequences from a uniform distribution in [a,b) by shifting and scaling:

```
z=random(N)#N numbers uniform in [0,1)
x = a+(b-a)*z #N numbers uniform in [a, b)
```

Non-uniform distributions

- The shift-and-scale example can be generalized to non-uniform cases.
- Consider a transformation of the random variable z:

$$x = x(z)$$

- x has distribution $p(x)$, z has distribution $q(z)$and these satisfy

$$p(x)dx = q(z)dz$$

$$\int_{-\infty}^{x} p(x')dx' = \int_{-\infty}^{z(x)} q(z')dz'$$

Non-uniform distributions

- Suppose z is generated by the python RNG, uniformly on [0,1), then
$$\int_{-\infty}^{x} p(x')dx' = \int_{-\infty}^{z(x)} q(z')dz' = z$$
- Suppose we want to draw from a new $p(x)$. To get the appropriate distribution of numbers from the uniform distribution z, integrate the LHS and try to invert to find $x(z)$.
- E.g. for shift-and-scale, we get as before:
$x = a + (b - a)z$

Non-uniform distributions

- Again, for z uniform on [0,1), then

$$\int_{-\infty}^{x} p(x')dx' = \int_{-\infty}^{z(x)} q(z')dz' = z$$

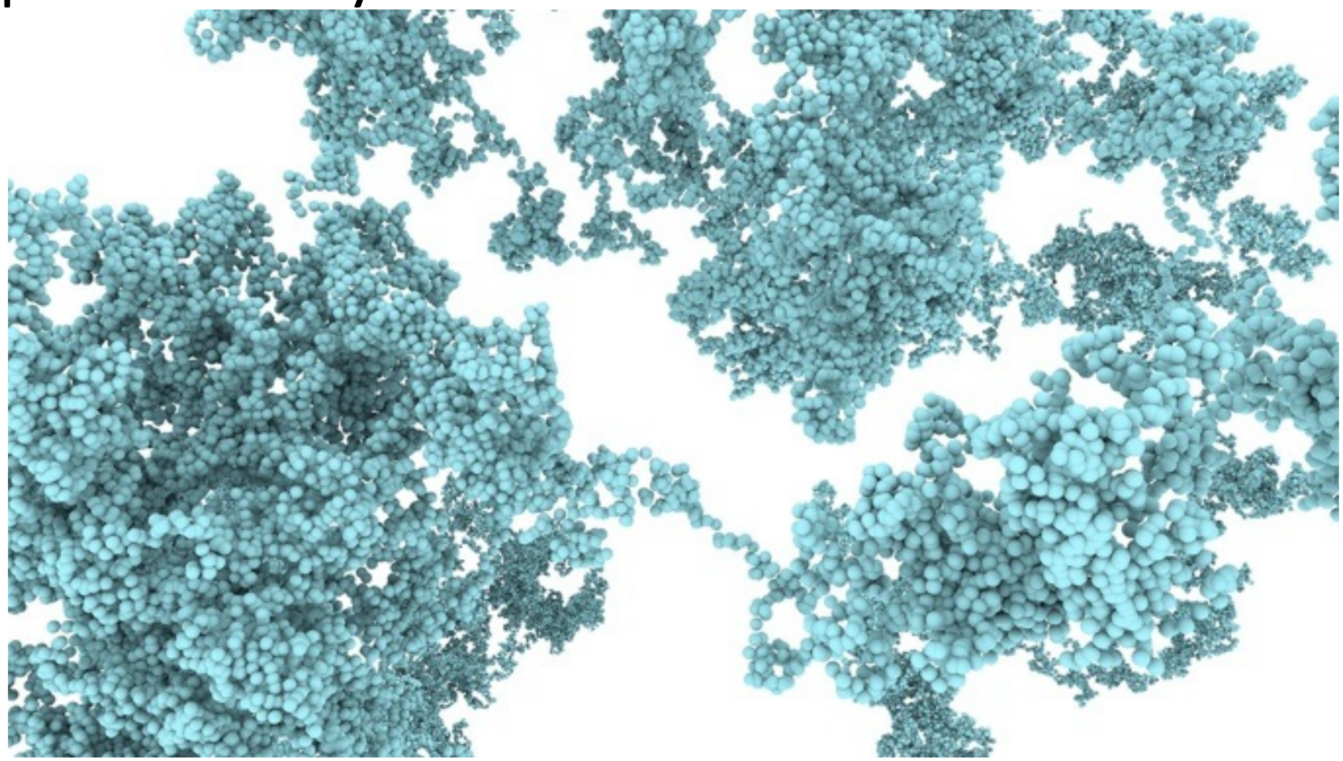
- E.g. for exponential distribution,

$$p(x) = \mu e^{-\mu x}$$
$$\Rightarrow x = -\frac{1}{\mu}\ln(1 - z)$$

- See Lecture10.py

Using Random Numbers

- *Simulate random physical processes:* diffusion, radioactive decay, Brownian motion.
- In Lab 10: simulate Diffusion Limited Aggregation (DLA) (DLA_example.py)
- Used to model clustering of particles to form aggregates or other situations where diffusion is the primary means of transport in the system.

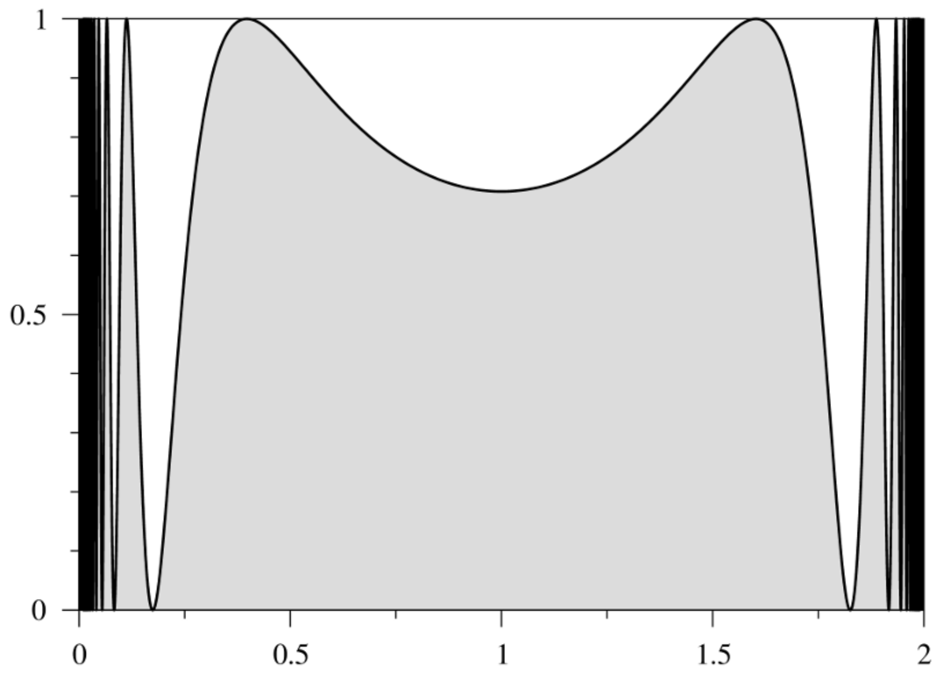


Using Random Numbers

- *Monte Carlo integration*: estimate integrals by randomly sampling many points within the integration domain.
- Techniques we'll learn about:
  - "hit or miss"
  - "mean value"
  - "importance sampling"

Why *another* integration method??

- We've learned a bunch of different methods already. Why another one? (We'll learn that its convergence/error properties are worse than other methods).
- Reason 1: Good for pathological functions or just quickly varying functions. e.g.:



$$I = \int_0^2 sin^2\left[\frac{1}{x(2-x)}\right]dx$$

Why *another* integration method??

- Reason 2: MUCH faster for multi-dimensional integrals:

  - For a dimension ‘d’ integral, you need $O(n^d)$ grid points.

  - e.g. with trapezoid, Simpson or Gaussian integration: for n=1000 points, a 10 dimensional integral need 10^{30} grid points! Yikes!

Why *another* integration method??

- Reason 3: Much easier to implement complicated domains (i.e. boundaries of integration).

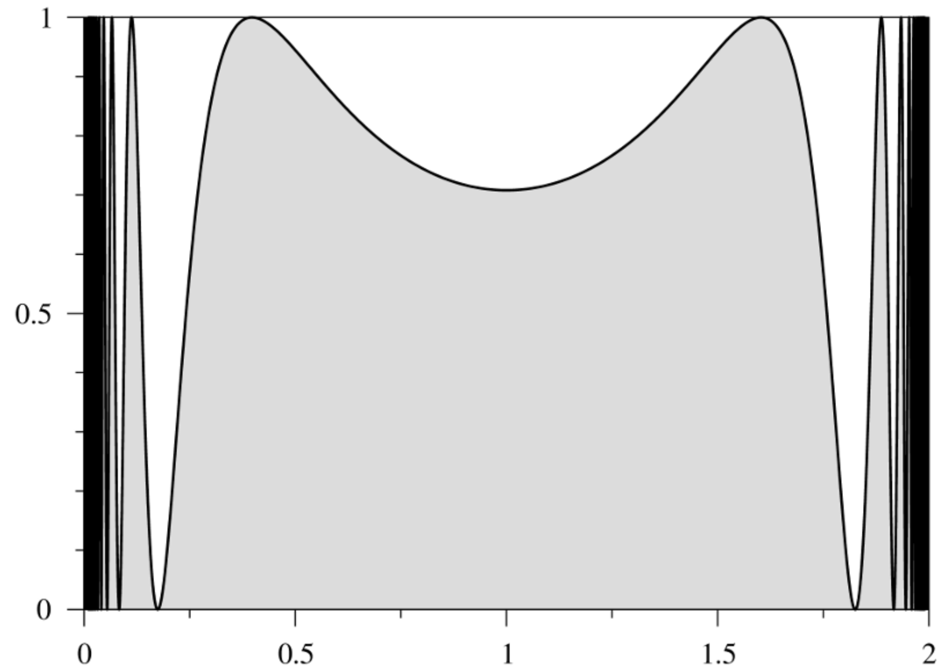
Try integrating over this volume using Gaussian quadrature!

Hit or Miss Monte Carlo

- If your function "fits" in a finite region where we want to integrate from x=0 to x=2:

$$f(x) = \sin^2\left[\frac{1}{x(2-x)}\right]dx$$

  - $f(x)$ fits in box of height 1, width 2.
  - Define area of box: A

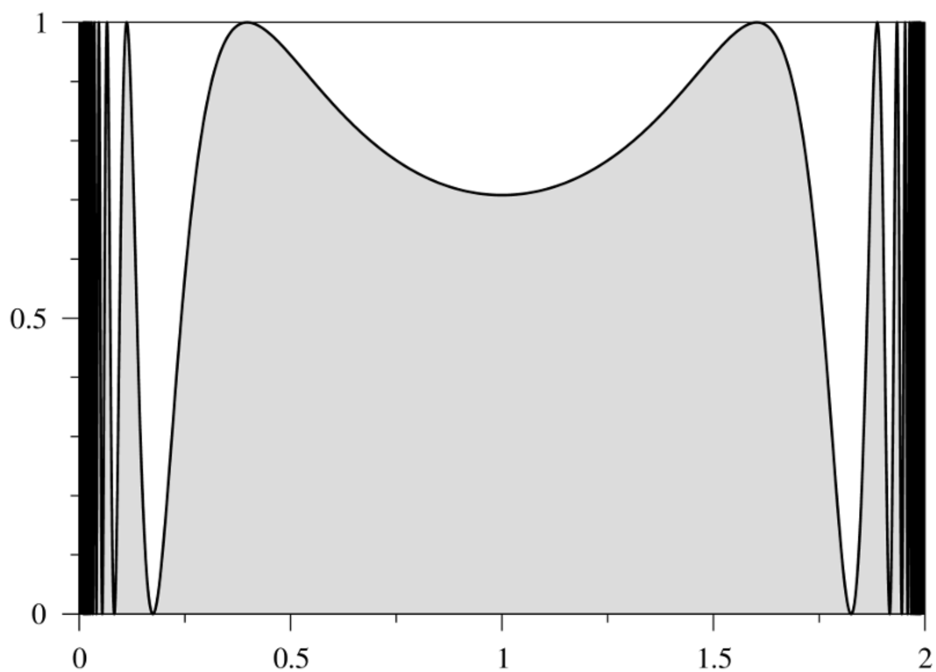


- Integral of function is shaded area in the box (call it I)

Hit or Miss Monte Carlo

- Probability that your random point falls in the shaded region is

$$p = \frac{I}{A}$$



Algorithm:

1. Randomly sample at N points in the box (lots of them)
2. Count the number of samples that are in the shaded region (call the count k)
3. The fraction of samples in the shaded region is k/N. This approximates the probability p.
4. Solve for I:

$$\boxed{p = \frac{I}{A} \approx \frac{k}{N} \Rightarrow I \approx \frac{kA}{N}}$$

Hit or Miss Monte Carlo: Error Estimate

- Can estimate the error on the integral (text gives derivation on page 467, using binomial distribution):

- The 'Expected Error':

$$\boxed{\sigma = \frac{\sqrt{I(A-I)}}{\sqrt{N}}}$$

- Notice it varies as $N^{-1/2}$. This is very SLOW!

- Compare: Trapezoid Rule: error varies as N^{-2}
  Simpson's Rule: error varies as N^{-4}

- This is why you only use Monte Carlo integration if you absolutely have to.

Hit or Miss Monte Carlo: Example

- Exercise 10.5 (a) from the text:
  - Write a program to evaluate the integral:

$$I = \int_0^2 \sin\left[\frac{1}{x(2-x)}\right] dx$$

using the "hit-or-miss" Monte Carlo method with 10000 points. Also evaluate the error on your method.

See exercise_10-5a.py

Mean Value Monte Carlo

- Use the definition of an average (or mean value):

$$I = \int_a^b f(x)\,dx$$

$$< f > = \frac{1}{b-a}\int_a^b f(x)\,dx = \frac{I}{b-a} \Rightarrow$$

$$I = (b-a) < f >$$

- **Algorithm:** Use random numbers to estimate $< f >$. Evaluate f and N random x's, then calculate:

$$< f > \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i) \Rightarrow I \approx \frac{b-a}{N}\sum_{i=1}^{N} f(x_i)$$

Mean Value Monte Carlo: Error Estimate

- Can estimate the error on the integral (text gives derivation on pages 468-469 from probability theory:

- The 'Expected Error':

$$\boxed{\begin{aligned}\sigma &= (b-a)\frac{\sqrt{\operatorname{var} f}}{\sqrt{N}} \\ \operatorname{var} f &= \langle f^2 \rangle - \langle f \rangle^2\end{aligned}}$$

- Notice it also varies as $N^{-\frac{1}{2}}$. However, it turns out the leading constant is smaller than with the hit or miss method (Exercise 10.6).

Mean Value Monte Carlo: Example

- Exercise 10.5 (b) from the text:
  - Write a program to evaluate the integral:

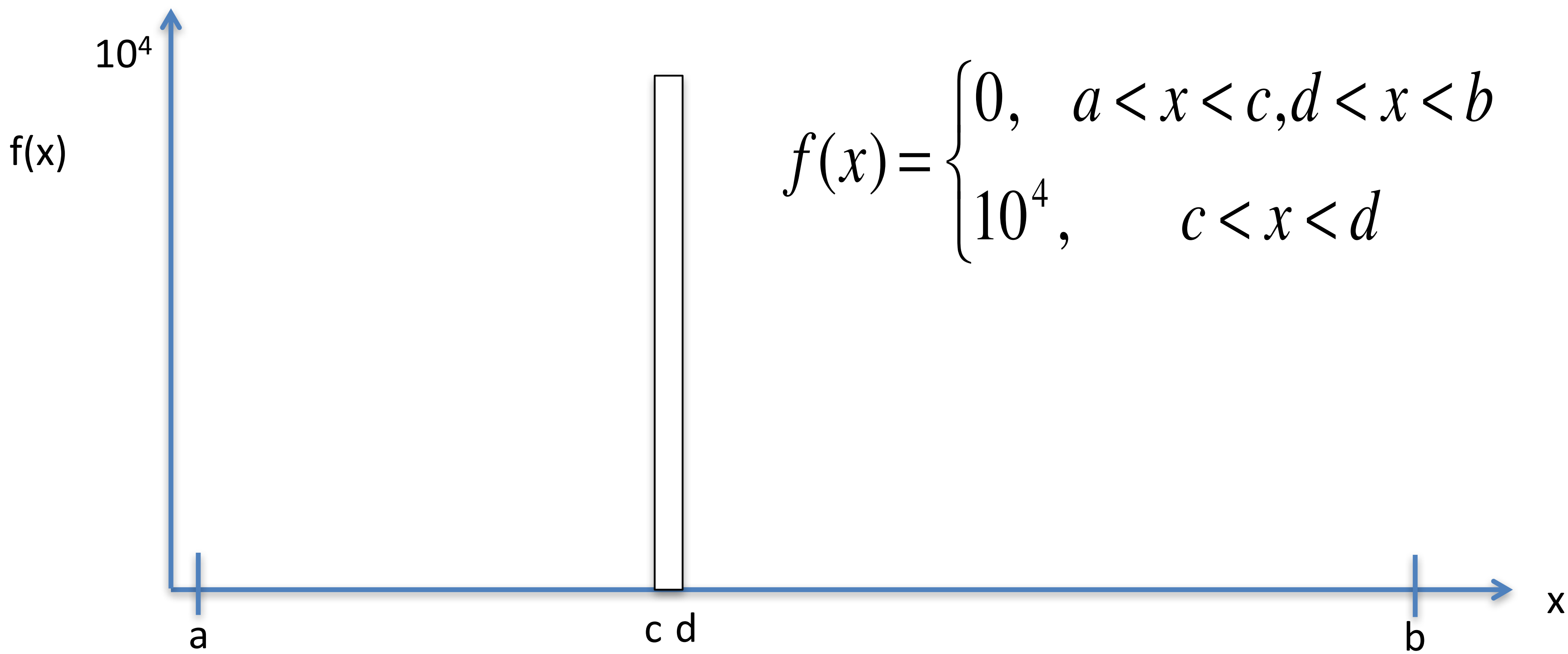
$$I = \int_0^2 \sin\left[\frac{1}{x(2-x)}\right] dx$$

using the “mean value” Monte Carlo method with 10000 points. Also evaluate the error on your method.

See exercise_10-5b.py

Importance Sampling Monte Carlo

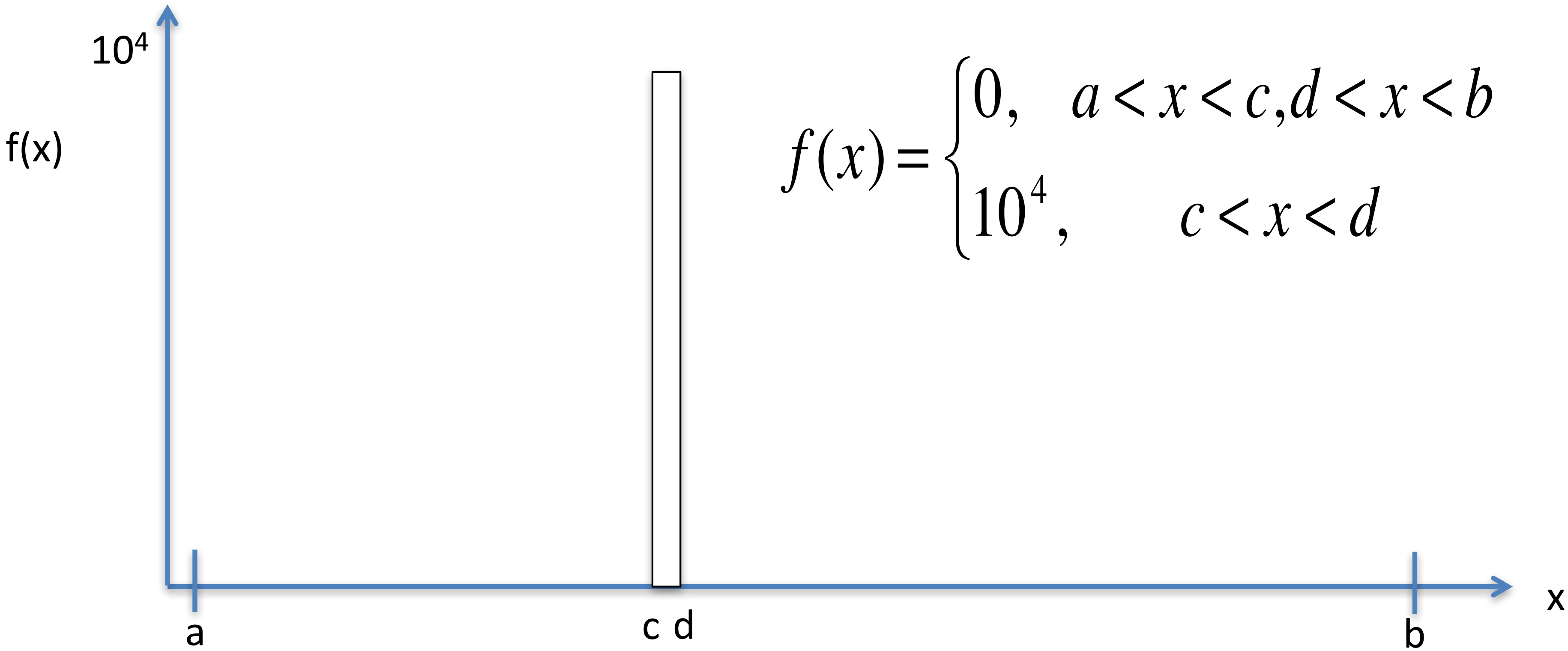
- Good to use when your integrand contains a divergence: want to place more points in region where the integrand is large to better estimate the integral.
- Also when you want to integrate out to infinity or over very small values.
- Illustrative example:



$$f(x) = \begin{cases} 0, & a < x < c, d < x < b \\ 10^4, & c < x < d \end{cases}$$

Importance Sampling Monte Carlo

- Easy to miss the region between c and d with uniformly sampled points
- Evaluating the integral many times using Mean Value or Hit/Miss Monte Carlo (with different randomly sampled points) can give very different answers, much larger than the expected error.



$$f(x)=\begin{cases}0, & a<x<c, d<x<b\\ 10^4, & c<x<d\end{cases}$$

Importance Sampling Monte Carlo

- Solution: sample 'important' regions more frequently. i.e. come up with a non-uniformly distributed set of random numbers.   This is called "Importance Sampling".

- Text shows that using a weight function $w(x)$, you can always write:

$$\boxed{I = \int_a^b f(x)dx = \left\langle \frac{f(x)}{w(x)} \right\rangle_w \int_a^b w(x)dx}$$

- Goal: find a weight function that gets rid of pathologies in integrand $f(x)$. E.g. if $f(x)$ has a divergence, factor the divergence out and hence get a sum (in the $<>$) that is well behaved (i.e. doesn't vary much each time you do the integral).

Importance Sampling Monte Carlo

Example (you will do in Lab 10):

$$I = \int_0^1 \frac{x^{-1/2}}{e^x + 1} dx$$

- Diverges as $x \to 0$ because of numerator. Ok, let $w(x)$ be the numerator. Then

$$\left\langle \frac{f(x)}{w(x)} \right\rangle_w = \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{w(x_i)} = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{e^{x_i} + 1}$$

[where the points x_i are sampled from $w(x)$, next slide].

- This is much better behaved than

$$\langle f(x) \rangle = \frac{1}{N} \sum_{i=1}^{N} \frac{x_i^{-1/2}}{e^{x_i} + 1}$$

Importance Sampling Monte Carlo

- When you've chosen your weight function, you then need to make sure to randomly sample points from the non-uniform distribution:

$$\boxed{p(x) = \frac{w(x)}{\int_a^b w(x)\,dx}}$$

- Use the transformation method described earlier in this lecture to take a uniformly distributed random z and find the corresponding x for this distribution.

Importance Sampling Monte Carlo: Error Estimate

- Can estimate the error on the integral from probability theory:

- The 'Expected Error':

$$\boxed{\sigma = \frac{\sqrt{\mathrm{var}_w(f/w)}}{\sqrt{N}} \int_a^b w(x)dx}$$

- Notice it also varies as $N^{-\frac{1}{2}}$. If you do the integral many times, your values should mostly fall within the expected error.

PHY407: Computational Physics
Fall, 2017

Lecture 10: Random Processes and Monte-Carlo Integration

- Random number generation, non-uniform distributions
- Monte-Carlo integration: three methods