Computing General Integrals using Monte Carlo

Let's consider integrating $f(x, y)$ on domain $\Omega \subset [0, L]^2$

$$G = \int \int_{\Omega} f(x, y) \, dx \, dy = \int_0^L \int_0^L f(x, y) \delta(x, y) \, dx \, dy,$$

where $\delta(x, y) = 1$ if $(x, y) \in \Omega$ and $\delta(x, y) = 0$ if $(x, y) \not\in \Omega$. 

$$G = \int \int_{\Omega} f(x, y) \, dx \, dy = \int_0^L \int_0^L f(x, y) \delta(x, y) \, dx \, dy,$$

where $\delta(x, y) = 1$ if $(x, y) \in \Omega$ and $\delta(x, y) = 0$ if $(x, y) \not\in \Omega$. 

Let $S(x, y)$ be a standard normal distribution with mean 0 and standard deviation 1. 

$$p(x, y) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x^2+y^2)}{2}}$$

and $Z$ distributed by $p$.

$$c = \text{Var}(E[F(Z)]) = \text{Var}(\int f(x, y) p(x, y) \, dx \, dy)$$
\[ \tilde{p}(x, y) = \frac{1}{L^2} \]

\[ G = \left| \mathbb{E} \left[ f(\tilde{x}) \right] \right| = \left| \mathbb{E} \left[ f(x) \frac{p(x)}{\tilde{p}(x)} \right] \right| = \left| \mathbb{E} \left[ f(x) \frac{L^2}{1/L^2} \right] \right| = \left| \mathbb{E} \left[ f(x) L^2 \right] \right| \]
\[ G = \mathbb{E} \left[ f(\mathbf{x}) \rho(\mathbf{x}) \right] \]

\[ \approx \frac{\mathbb{E} \left[ \sum_{i=1}^{N} f(s_i) \rho(s_i) \right]}{N} \]

\[ = \frac{\sum_{i=1}^{N} f(s_i) \rho(s_i)}{N} \]
Monte Carlo Methods: The Good and the Bad

What are some advantages of MC methods?
- Easy to use for integration
- Reliable convergence

What are some disadvantages of MC methods?
- Slow! Especially for simple integration
- Unreliable due to randomness
Computers and Random Numbers

How can a computer make random numbers?

Use an actual source of randomness.
Random Numbers: What do we want?

What properties can ‘random numbers’ have?

- Have a specific distribution
  (e.g. ‘uniform’—each value in given interval is equally likely)
- Real-valued/integer-valued
- Repeatable (i.e. you may ask to exactly reproduce a sequence)
- Unpredictable
  - V1: ‘I have no idea what it’s going to do next.’
  - V2: No amount of engineering effort can get me the next number.
- Uncorrelated with later parts of the sequence
  (Weaker: Doesn’t repeat after a short time)
- Usable on parallel computers
What’s a Pseudorandom Number?

Actual randomness seems like a lot of work. How about ‘pseudo-random numbers’?

**Idea:** Maintain some ‘state’. Every time someone asks for a number:

\[
\text{random_number, new_state} = f(\text{state})
\]

Satisfy:

- **Distribution**
- ‘I have no idea what it’s going to do next.’
- Repeatable (just save the state)
- Typically *not* easy to use on parallel computers
Demo: Playing around with Random Number Generators
Some Pseudorandom Number Generators

Lots of variants of this idea:

- LC: ‘Linear congruential’ generators
- MT: ‘Mersenne twister’
- almost all random number generators you’re likely to find are based on these—Python’s `random` module, `numpy.random`, C’s `rand()`, C’s `rand48()`.
Counter-Based Random Number Generation (CBRNG)

What’s a CBRNG?

**Idea:** Cryptography has *way* stronger requirements than RNGs. *And* the output *must* ‘look random’.

(Advanced Encryption Standard) AES algorithm:
128 encrypted bits = AES (128-bit plaintext, 128 bit key)

We can treat the encrypted bits as random:
128 random bits = AES (128-bit counter, arbitrary 128 bit key)

- Just use 1, 2, 3, 4, 5, . . . . as the counter.
- *No* quality requirements on counter or key to obtain high-quality random numbers
- *Very* easy to use on parallel computers
- Often accelerated by hardware, faster than the competition

**Demo:** Counter-Based Random Number Generation
Outline

Python, Numpy, and Matplotlib

Making Models with Polynomials
Making Models with Monte Carlo

Error, Accuracy and Convergence

Floating Point

Modeling the World with Arrays
  The World in a Vector
  What can Matrices Do?
  Graphs
  Sparsity

Norms and Errors

The ‘Undo’ Button for Linear Operations: LU
LU: Applications
  Linear Algebra Applications
  Interpolation

Repeating Linear Operations:
Eigenvalues and Steady States
Eigenvalues: Applications
Approximate Undo: SVD and Least Squares
SVD: Applications
  Solving Funny-Shaped Linear Systems
  Data Fitting
  Norms and Condition Numbers
  Low-Rank Approximation

Iteration and Convergence
Solving One Equation
Solving Many Equations
Finding the Best: Optimization in 1D
Optimization in \( p \) Dimensions
Every result we compute in Numerical Methods is inaccurate. What is our model of that error?

\[ \text{Computed} = \text{real value} + \text{error} \]

Suppose the true answer to a given problem is \( x_0 \), and the computed answer is \( \tilde{x} \). What is the absolute error?

\[ \text{absolute error} = |x_0 - \tilde{x}| \]

\[ = |\tilde{x} - x_0| \]
Relative Error

What is the relative error?

\[
\text{relative error} = \frac{|x - x_0|}{|x_0|}
\]

Why introduce relative error?

\[
\text{error} = 2, \quad x_0 = 4, \quad x_0 \approx 4.00000000000000000000
\]

What is meant by ‘the result has 5 accurate digits’?

\[
\hat{x} = 0.00005
\]

\[
\bar{x} = x_0 - x_1 x_2 x_3 x_4 \times 10^y
\]
Measuring Error

Why is $|\tilde{x}| - |x_0|$ a bad measure of the error?

If $\tilde{x}$ and $x_0$ are vectors, how do we measure the error?

$$|\tilde{x} - x_0| \neq |\tilde{x}| - |x_0|$$

$$\left\| \tilde{x} - x_0 \right\|$$
Sources of Error

What are the main sources of error in numerical computation?

1. Experimental (input)
2. Truncation error
3. Roundoff error
4. Method error
Digits and Rounding

Establish a relationship between ‘accurate digits’ and rounding error.

\[
\hat{x} = 0.0034
\]

\[
x_0 = 0.00342271
\]

\[
|\hat{x} - x_0| = 0.0000271 \ldots \leq 10^{-4}
\]

\[
\frac{|\hat{x} - x_0|}{|x_0|} \leq 10^{-2}
\]
absolute

\[ |\tilde{x} - x_0| \leq 10^{r-k} \]

if \( \tilde{x} \) has \( k \) accurate digits

and \( x_0 = x \cdot 10^r \)

where \( x = \cdot \ldots \ldots \)

\[ \tilde{x} = 123450.0000 \quad k = 5 \]

\[ x_0 = 123456.789 \quad r = 6 \]

\[ |\tilde{x} - x_0| \leq 10 \]
relative error

\[
\frac{|x - x_0|}{|x_0|} \leq \frac{10^{-k}}{|x_0|}
\]

\[|x_0| \leq 10^k\]
Condition Numbers

Methods $f$ take input $x$ and produce output $y = f(x)$. Input has (relative) error $|\Delta x| / |x|$. Output has (relative) error $|\Delta y| / |y|$. Q: Did the method make the relative error bigger? If so, by how much?

$k = \text{relative error of output} = \frac{|f(x + \Delta x) - f(x)|}{|f(x)|}$
$k = \frac{1}{|f(x)|} \left| \frac{(x + \Delta x) - x}{x} \right| / |x|^{1}$
\[ k = \frac{|f(x + \Delta x) - f(x)|}{|f(x)|} \frac{|f(x)|}{|\Delta x|} \]

\[ r^2 = \frac{|f(x + \Delta x) - f(x)|}{|f(x)| \Delta x} \]
\[ k = \frac{1 \times |f'(x)|}{|f(x)|} \]

\[ f'(a) = \lim_{\Delta x \to 0} \left( \frac{f(a+\Delta x) - f(a)}{\Delta x} \right) \]
nth-Order Accuracy

Often, *truncation error* is controlled by a parameter $h$.

Examples:

- distance from expansion center in Taylor expansions
- length of the interval in interpolation

A numerical method is called ‘*nth-order accurate*’ if its truncation error $E(h)$ obeys

$$E(h) = O(h^n).$$
Outline

Python, Numpy, and Matplotlib
Making Models with Polynomials
Making Models with Monte Carlo
Error, Accuracy and Convergence

Floating Point
Modeling the World with Arrays
  The World in a Vector
  What can Matrices Do?
  Graphs
  Sparsity
Norms and Errors
The ‘Undo’ Button for Linear Operations: LU
LU: Applications
  Linear Algebra Applications
  Interpolation

Repeating Linear Operations: Eigenvalues and Steady States
Eigenvalues: Applications
Approximate Undo: SVD and Least Squares
SVD: Applications
  Solving Funny-Shaped Linear Systems
  Data Fitting
  Norms and Condition Numbers
  Low-Rank Approximation
Iteration and Convergence
Solving One Equation
Solving Many Equations
Finding the Best: Optimization in 1D
Optimization in $n$ Dimensions