Differentiation and Integration

(Lectures on Numerical Analysis for Economists II)

Jesús Fernández-Villaverde[1] and Pablo Guerrón[2]

February 15, 2022

[1]University of Pennsylvania

[2]Boston College

Motivation

- Two basic operations in scientific computation are differentiation and integration.
- Key, for example, for:
  1. Equation solving.
  2. Optimization (gradients, Hessians, Jacobians, ...).
  3. ODEs and PDEs.
  4. Statistics and econometrics.
  5. Machine learning.
- Potentially costly operations because of their repetition.
- Not general algorithm for them. Instead, menu of options.

Algorithms

- Differentiation:

  1. Forward/backward/central differentiation.
  2. Complex Step Differentiation.
  3. Symbolic differentiation.
  4. Automatic differentiation.

- Integration:

  1. Quadrature methods.
  2. Monte Carlo.
  3. Quasi Monte Carlo.

Numerical Differentiation

Forward differencing

- The derivative of a function $f(\cdot)$ is

$$f'(x) = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon) - f(x)}{\varepsilon}$$

- Thus, a natural approximation is "forward differencing" scheme:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

  for a small $h > 0$.
- Extension to multivariate functions is straightforward:

$$\frac{\partial f(x)}{\partial x_i} \approx \frac{f(x_1, ..., x_i + h_i, ..., x_n) - f(x_1, ..., x_i, ..., x_n)}{h}$$

Error

- For linear functions, a "forward differencing" scheme approximation is exact.
- For non-linear functions, recall that a Taylor expansion of $f(x+h)$ around $h=0$ is:

$$f(x+h) = f(x) + f'(x)h + \frac{f''(\xi)}{2}h^2, \qquad \xi \in (x, x+h)$$

- Then, we can derive an expression for the truncation error:

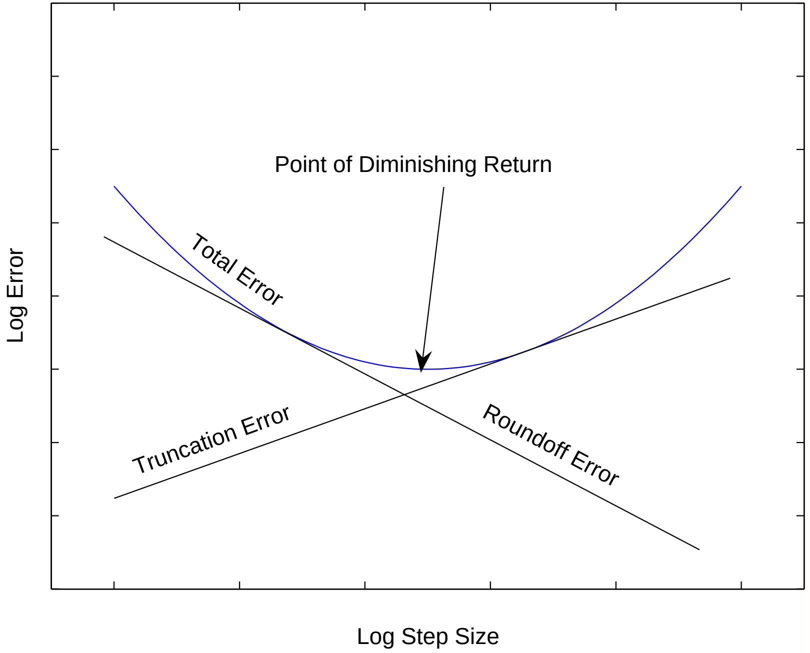
$$f'(x) = \frac{f(x+h) - f(x)}{h} + O(h)$$

which is valid if $f(x)$ has two continuous derivatives.

- Which h to pick?

$$h = \max(|x|, 1)\sqrt{\epsilon}$$

where ϵ is the machine precision (I am skipping proof, standard result).


Point of Diminishing Return
Total Error
Truncation Error
Roundoff Error
Log Error
Log Step Size

Backward difference

- "Backward differencing" scheme

$$f'(x) \approx \frac{f(x) - f(x-h)}{h}$$

- Extension to multivariate functions is:

$$\frac{\partial f(x)}{\partial x_i} \approx \frac{f(x_1, ..., x_i, ..., x_n) - f(x_1, ..., x_i - h_i, ..., x_n)}{h}$$

- Also, an approximation error of order h. Same proof as before.
- Thus, in practice, one uses forward or backward differences depending on whether we care more about left or right derivative (kinks, finite difference solvers for ODEs, ...).

Centered difference

- "Centered differencing" scheme

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

- Extension to multivariate functions is:

$$\frac{\partial f\left(x\right)}{\partial x_i} \approx \frac{f\left(x_1, ..., x_i + h_i, ..., x_n\right) - f\left(x_1, ..., x_i - h_i, ..., x_n\right)}{h}$$

- Rule for selecting h:

$$h = \max(|x|, 1)\sqrt[3]{\epsilon}$$

This choice minimizes the sum of round-off and truncation error.

Error with centered difference

- Truncation error can be found by subtracting Taylor expansions of $f(x+h)$ and $f(x-h)$:

$$f(x+h) - f(x-h) = 2f'(x)h + \frac{(f'''(\xi_1) + f'''(\xi_2))}{6}h^3 \Rightarrow$$
$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2)$$

- Centered differencing is more precise: truncation error of order h^2.
- Trade-off between accuracy and efficiency for functions of higher dimensions.
- Suppose $f : \mathbb{R}^n \to \mathbb{R}^m$, then computing the Jacobian matrix of f requires $m(n+1)$ function evaluations using the forward differencing scheme and $2mn$ evaluations using the centered differencing scheme, roughly twice as many when n is large.

Higher-order and cross-derivatives

- Numerical differentiation accumulates error.
- For second- and higher-order derivatives, and often, for cross-derivatives, the error can become substantial.
- We can improve upon the previous schemes: Richardson's extrapolation.
- Probably a good idea to search for alternatives:
  1. Complex step differentiation.
  2. Symbolic derivatives.
  3. Automatic differentiation.

Richardson's extrapolation

- Recall that centered differencing generates errors of order $O(h^2)$.
- We can increase the efficiency by using Richardson's extrapolation.
- However, we need to know also $f(x-2h)$ and $f(x+2h)$.
- We will skip the derivation (check any standard numerical analysis textbook).
- Result:
$$f'(x) = \frac{-f(x+2h) + 8f(x+h) - 8f(x-h) + f(x-2h)}{12h} + O(h^4),$$
a fourth-order approximation.
- Adjust h accordingly.

Complex step differentiation I

- Let $f(x)$ be an analytic function.
- Then:

$$f(x+ih) = f(x) + f'(x)ih + \frac{f''(\xi)}{2}h^2 + ...$$

- Take imaginary part on both sides of the expression:

$$\begin{aligned} \mathrm{Im}(f(x+ih)) &= \mathrm{Im}\left(f(x) + f'(x)ih + \frac{f''(\xi)}{2}h^2 + ...\right) \\ &= f'(x)h + ... \end{aligned}$$

- Dividing by h and reordering:

$$f'(x) = \frac{\mathrm{Im}\left(f(x+ih)\right)}{h} + O(h^2)$$

Complex step differentiation II

- Following the same logic, we can get second derivative:

$$f''(x) = \frac{2}{h^2} * (f(x) - \text{Real}(f(x + ih)))$$

- Similar formulae apply for Jacobians and Hessians.
- This algorithm depends on the ability of your programming language to deal efficient and accurately with complex arithmetic.

Symbolic differentiation

- Many programming languages have symbolic math packages/toolboxes.
  1. `C++`: GiNaC.
  2. `Python`: SymPy.
  3. `Julia`: SymPy.jl.
  4. `R`: Racayas.
  5. `Matlab`: Symbolic Math Toolbox.
- Disadvantages:
  1. Performance penalty.
  2. Limitations in abstractions.

Automatic differentiation

Definition

Set of techniques designed to numerically evaluate the derivative of a function while minimizing the amount of arithmetic operations.

- Automatic differentiation divides the function to derivate into small parts and then applies the chain rule to solve for the derivative.
- Example:

$$\begin{aligned} & xy^2 + \log x \\ = \quad & w_1 \left(w_2\right)^2 + \log w_1 \\ = \quad & w_3 + w_4 \\ = \quad & w_5 \end{aligned}$$

- We want to compute the derivative of this function with respect to x.

Example

Operations to compute value	Operations to compute derivative
$w_1 = x$	$w_1' = 1$ (seed)
$w_2 = y$	$w_2' = 0$ (seed)
$w_3 = w_1 (w_2)^2$	$w_3' = w_1' (w_2)^2 + 2 w_1 w_2'$
$w_4 = \log w_1$	$w_4' = \frac{w_1'}{w_1}$
$w_5 = w_3 + w_4$	$w_5' = w_3' + w_4'$

- We get:

$$\begin{aligned} w_5' &= w_3' + w_4' \\ &= w_1' (w_2)^2 + 2 w_1 w_2' + \frac{w_1'}{w_1} \\ &= (w_2)^2 + \frac{1}{w_1} \\ &= y^2 + \frac{1}{x} \end{aligned}$$

- How do you implement it in the computer?

Quadrature Integration

Overview

Quadrature

Method of solving an integral numerically by exploiting the definition of the integral.

- Trade-offs between accuracy, coding time, and running time.
- There are several quadrature methods, each evaluating the integral at different points and using the evaluations differently.
  1. Newton-Coates.
  2. Gaussian.
  3. Clenshaw-Curtis.
- Some methods are more general, but slower, whereas others can be more restrictive and complicated, but have faster running time.

Newton-Cotes: overview

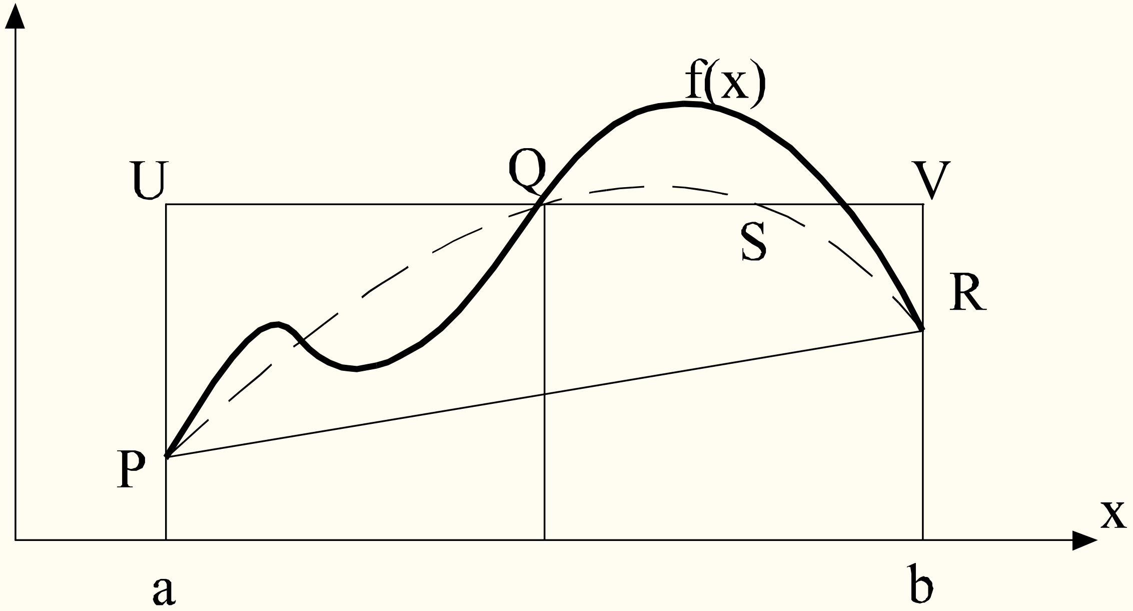
- Evaluate f at a finite number of points to create a piecewise-polynomial function.
- Then, integrate this approximation of f to approximate:

$$\int_D f(x)dx$$

- All Newton-Cotes rules are of the form:

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} \omega_i f(x_i)$$

for some quadrature nodes $x_i \in [a, b]$ and quadrature weights ω_i.


f(x)
U
Q
V
S
R
P
x
a
b

Midpoint rule

- Simplest (open) rule with one interval:

$$\int_a^b f(x)dx = \underbrace{(b-a)f\left(\frac{a+b}{2}\right)}_{\text{Integration rule}} + \underbrace{\frac{(b-a)^3}{24}f''(\xi)}_{\text{Error term}}$$

  where $\xi \in [a, b]$.

- Proof is application of Taylor' theorem and intermediate value theorem.

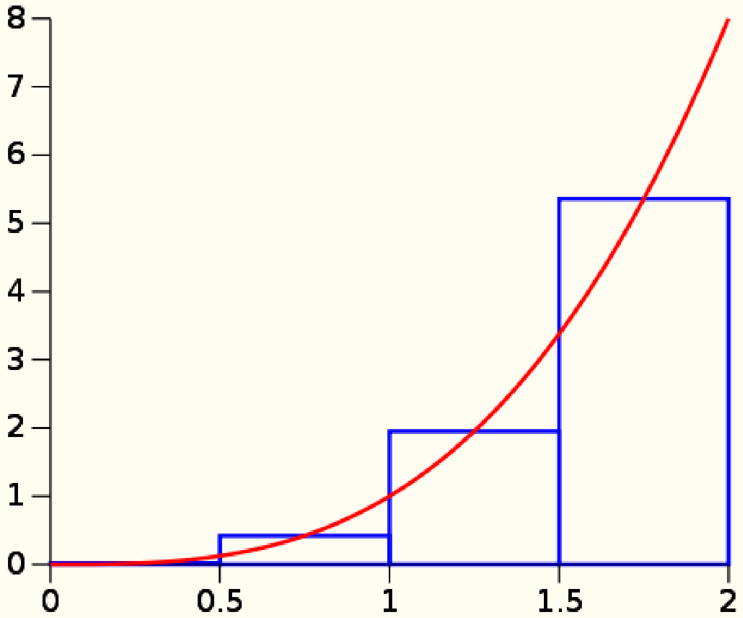
- With $n > 1$ intervals and $h = \frac{b-a}{n}$ step size, the *composite midpoint rule* is:

$$\int_a^b f(x)dx \approx h\sum_{j=1}^{n} f\left(a + \left(j - \frac{1}{2}\right)h\right) + \frac{h^2 \cdot (b-a)}{24}f''(\xi)$$

  for some $\xi \in [a, b]$.

- Note quadratic convergence: doubling intervals, reduces error $\approx 75\%$.

- We can define irregularly-spaced step sizes if additional information about $f(\cdot)$ is available.



8
7
6
5
4
3
2
1
0
0
0.5
1
1.5
2

Trapezoid rule

- The trapezoid (close) rule uses a linear approximation of f along with the values of f at the endpoints:

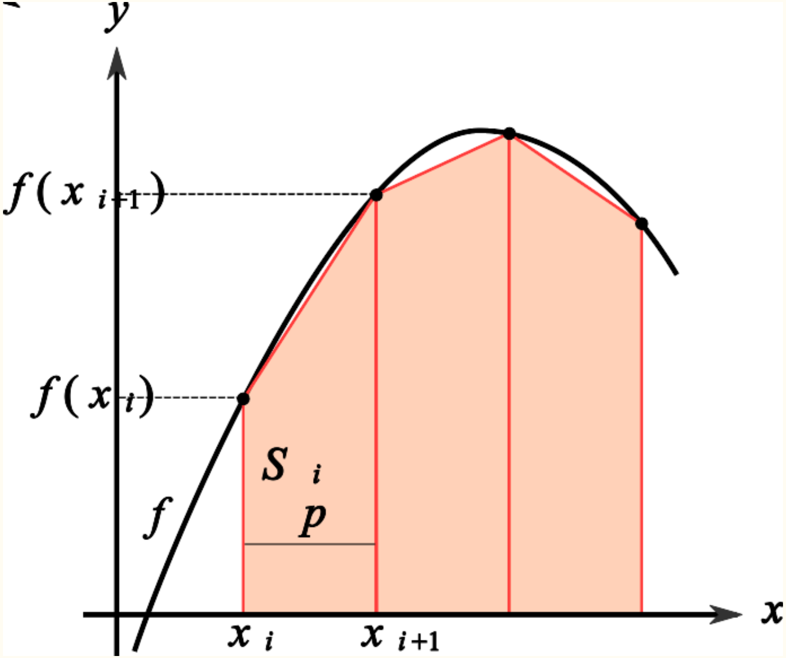
$$\int_a^b f(x)dx = (b-a)\frac{f(a)+f(b)}{2} - \frac{(b-a)^3}{12}f''(\xi)$$

  where $\xi \in [a,b]$.

- We can define the *composite trapezoid rule* as we did with the *composite midpoint rule*:

$$\int_a^b f(x)dx \approx \frac{b-a}{n}\left(\frac{f(a)}{2} + \sum_{j=1}^{n-1} f\left(a + \left(j\frac{b-a}{2}\right)\right) + \frac{f(a)}{2}\right)$$

- Again, we can have irregularly-spaced steps.


y
$f(x_{i+1})$
$f(x_i)$
S_i
p
f
x_i
x_{i+1}
x

Simpson rule

- The Simpson rule uses a piecewise-quadratic approximation of f along with the values of f at the endpoints and the midpoint.

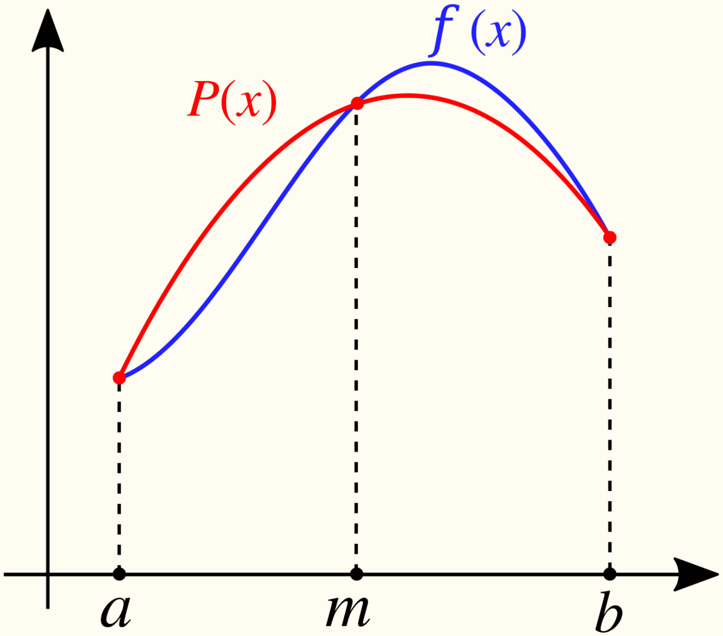
$$\int_a^b f(x)dx = \frac{b-a}{6}\left[f(a) + 4f\left(\frac{b+a}{2}\right) + f(b)\right] - \frac{(b-a)^5}{2880} f^{(4)}(\xi)$$

  where $\xi \in [a, b]$.

- Analogous *composite rule* with $n \geq 2$ intervals, $h = \frac{(b-a)}{n}$ and $x_j = a + jh$, is defined as

$$S_n(f) = \frac{h}{3}[f_0 + 4f_1 + 2f_2 + 4f_3 + ... + 4f_{n-1} + f_n] - \frac{h^4(b-a)}{180} f^{(4)}(\xi)$$

  where $\xi \in [a, b]$
- Other rules: Simpson's 3/8 rule and Boole's rule.
- Also, we can implement change of variables.


$f(x)$
$P(x)$
a
m
b

Gaussian: overview

- For any fixed non-negative weighting function $w(x)$ Gaussian quadrature creates approximation of the form:

$$\int_a^b f(x)w(x)dx \approx \sum_{i=1}^{n} \omega_i f(x_i)$$

  for some nodes $x_i \in [a, b]$, and positive weights w_i.

- Gaussian quadrature builds on orthogonal Legendre polynomials approximations.
- In general, more efficient than Newton-Cotes. Why?
- Often, computing the nodes and weights is not required, as the more useful Gaussian quadrature nodes and weights can be found in tables.

Chebyshev quadrature

- Used for integrals of the form:

$$\int_{-1}^{1} f(x)\underbrace{\left(1-x^2\right)^{-\frac{1}{2}}}_{\text{Weighting fun}} dx$$

- Gauss-Chebyshev quadrature formula:

$$\int_{-1}^{1} f(x)(1-x^2)^{-\frac{1}{2}} dx = \frac{\pi}{n}\sum_{i=1}^{n} f(x_i) + \frac{\pi}{2^{2n-1}}\frac{f^{(2n)}(\xi)}{(2n)!}$$

  for some $\xi \in [-1, 1]$ where the quadrature nodes are $x_i = cos\left(\frac{2i-1}{2n}\pi\right)$ with $i = 1, ..., n$.

- Change of variables to accommodate different intervals.

- Constant weight $\frac{\pi}{n}$ for each node and quadrature nodes that are easy to compute.

Hermite quadrature

- Used for integrals of the form
$$\int_{-\infty}^{\infty} f(x) \underbrace{e^{-x^2}}_{\text{Weighting fun}} dx$$
where the function is evaluated at the Hermite polynomial roots.
- For a random variable Y with distribution $\mathcal{N}(\mu, \sigma^2)$, a linear change of variables gives
$$\int_{-\infty}^{\infty} f(y) e^{-\frac{(y-\mu)^2}{2\sigma^2}} dy = \int_{-\infty}^{\infty} f(\sqrt{2}\sigma x + \mu) e^{-x^2} \sqrt{2}\sigma dx$$
- Useful for economics due to the common use of normally distributed random variables, especially in macro and finance.

Interpolatory rules

- Interpolatory quadrature rules involves using derivatives of $f(x)$ to approximate the integral $\int_a^b f(x)dx$.

  $$\int_a^b f(x)w(x)dx \approx \sum_{i=i}^{n} \sum_{j=1}^{m} \omega_{ij} f^{(j)}(x_i)$$

  where once again the x_i are nodes and the ω_i are weights.

- It often involves substantial extra calculation due to evaluating the derivatives.

Newton-Cotes vs. Gaussian

- In Newton-Cotes formulas, the x_i points are chosen arbitrarily, ususally uniformly spaced on $[a, b]$ whereas in the Gaussian formulas, both the nodes and weights are chosen efficiently.
- Efficiency is measured using the *exact integration* for finite-dimensional collection of functions.
- While Gaussian may have a time advantage over Newton-Cotes, this comes at the cost of having to perform complex calculations to find the weights and nodes.
- Clenshaw-Curtis quadrature –based on an expansion of the integrand in terms of Chebyshev polynomials– is often a good intermediate compromise.

Multidimensional quadrature

- One approach to deal with multidimensional integrals is to directly extend the one-dimensional methods via product rules.
- However:
  - The algebra becomes very challenging.
  - There is no guarantee of a solution and if there is a solution, then there will be multiple and it is possible that they will have negative weights.
  - The curse of dimensionality is acute.
- We will revisit Smolyak grids when we talk about projection methods.
- The main practice used to extend to higher dimensions is Monte Carlo integration.

Monte Carlo Integration

A bit of historical background and intuition

- Metropolis and Ulam (1949) and Von Neuman (1951)
- Why the name "Monte Carlo"?
- Two silly examples:
  1. Probability of getting a total of six points when rolling two (fair) dices.
  2. Throwing darts at a graph.

Overview

- This method can handle problems of far greater complexity and size than most other methods.
- As well, Monte Carlo methods can deliver accurate results using moderate number of points (which are randomly selected).
- Some strengths of this method are its robustness and simplicity.
- It is based on the law of large numbers and the central limit theorem.
- Monte Carlo produces a random approximation, which puts structure on the error term.
- For an approximation $\hat{I}_f$ the variance will be

$$\sigma^2_{\hat{I}_f} = \frac{1}{N}\int_0^1 \left(f(x) - I_f\right)^2 dx = \frac{1}{N}\sigma^2_f$$

Crude method

- A crude Monte Carlo estimate of $E[f(X)] = \int_0^1 f(x)dx$ is calculated by generating N draws from $U[0,1]$,$\{x_i\}_{i=1}^N$ and takes the form

$$\hat{I}_f = \frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

  where $\hat{I}_f$ is also a random variable.

- Although this estimator is unbiased, it is not commonly used, because of its large variance.

- There are variety of simple techniques that can reduce the variance, but retain its unbiasedness.

Randomness in computation

Von Neumann (1951)

Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.

- Let's us do a simple experiment.
- Let's us start `Matlab`, type `format long`, type `rand`.
- Did we get 0.8147?
- This does not look terribly random.
- Why is this number appearing?
- `Matlab` uses highly non-linear iterative algorithms that "look like" random.
- That is why sometimes we talk of pseudo-random number generators.

How do we generate random numbers?

- Large literature on random number generation.
- Most basic algorithms draw from a uniform distribution.
- Other (standard and nonstandard) distributions come from manipulations of the uniform.
- Two good surveys:
  1. Luc Devroye: *Non-Uniform Random Variate Generation*, Springer-Verlag, 1986.

     Available for free at: `http://www.nrbook.com/devroye/`.
  2. Christian Robert and George Casella, *Monte Carlo Statistical Methods*, 2nd ed, Springer-Verlag, 2004.
- Use state-of-art random number generators. It matters!

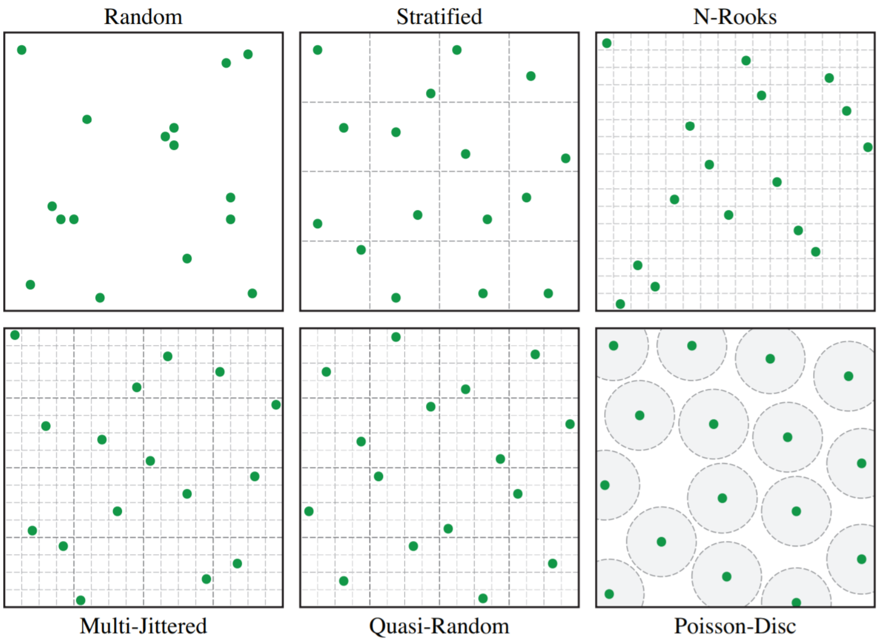
Stratefied sampling

- This sampling method exploits the fact that there will be subintervals with lower variance.
- Suppose that we divide $[0, 1]$ into $[0, \alpha]$ and $[\alpha, 1]$, then if we have N points in each interval we can form the estimate

$$\hat{I}_f = \frac{\alpha}{N} \sum_i f(x_{1i}) + \frac{1-\alpha}{N} \sum_i f(x_{2i})$$

  where $x_{1i} \in [0, \alpha]$ and $x_{2i} \in [\alpha, 1]$.
- Its variance is

$$\sigma^2_{\hat{I}_f} = \frac{\alpha}{N} \int_0^\alpha f^2 + \frac{1-\alpha}{N} \int_\alpha^1 f^2 - \frac{\alpha}{N} \left(\int_0^\alpha f \right)^2 - \frac{1-\alpha}{N} \left(\int_\alpha^1 f \right)^2$$


Random
Stratified
N-Rooks
Multi-Jittered
Quasi-Random
Poisson-Disc

Importance sampling

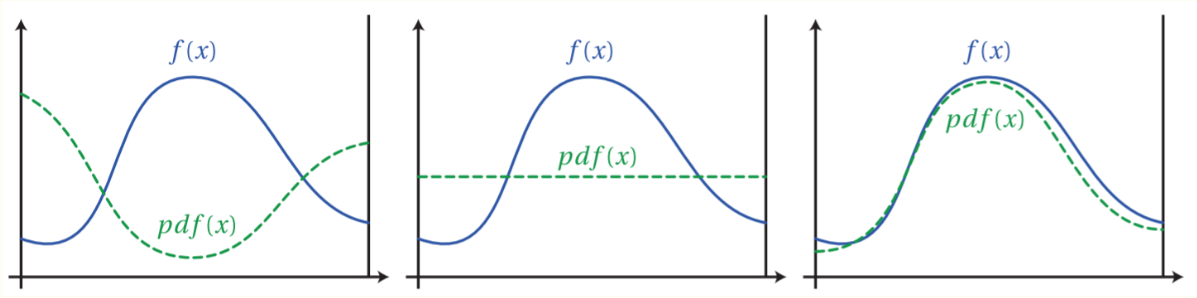
- The idea of the method is that we sample more intensively where f is large, which is where f is making the greatest contribution to $\int f(x)dx$.
- If $p(x) > 0$, and $\int_0^1 p(x)dx = 1$, then $p(x)$ is a density and

$$I_f = \int_0^1 f(x)dx = \int_0^1 \frac{f(x)}{p(x)} p(x)dx$$

- Therefore, if x_i is drawn with density $p(x)$, then the following is an unbiased estimator of I

$$\hat{I}_f = \frac{1}{N} \sum_{i=1}^{n} \frac{f(x_i)}{p(x_i)}$$

and its variance has decreased.



$f(x)$
$pdf(x)$
$f(x)$
$pdf(x)$
$f(x)$
$pdf(x)$

Comparison: 1 dimension

$$\text{Calculate } \int_0^1 e^x \, dx$$

Approximation Error				
N	Midpoint	Trapezoid	Simpson's	Monte Carlo
10	-0.00071574	0.00143166	0.00000006	0.09523842
100	-0.00000716	0.00001432	0.00000000	0.01416057
1000	-0.00000007	0.00000014	-0.00000000	-0.00515829
5000	-0.00000000	0.00000001	0.00000000	0.00359500

Computation Time (sec.)				
N	Midpoint	Trapezoid	Simpson's	Monte Carlo
10	0.00	0.02	0.01	0.00
100	0.02	0.01	0.02	0.00
1000	0.01	0.04	0.01	0.02
5000	0.04	0.07	0.06	0.01

Comparison: 2 dimensions

$$\text{Calculate } \int_0^1 \int_0^1 e^x e^y \, dxdy$$

Approximation Error		
N	Midpoint	Monte Carlo
10	-0.00245918	0.33897914
100	-0.00002460	0.03021147
1000	-0.00000025	-0.05486922
5000	-0.00000001	0.01183325

Computation Time (sec.)		
N	Midpoint	Monte Carlo
10	0.00	0.02
100	0.11	0.01
1000	9.18	0.01
5000	229.14	0.02

More Monte Carlo draws

N	Approximation Error	Computation Time
10	0.33897914	0.00
100	0.03021147	0.00
1000	-0.05486922	0.00
10000	0.00365290	0.00
100000	-0.00177819	0.03
1000000	-0.00177012	0.25
10000000	0.00065619	3.39
100000000	-0.00007068	24.94

Quasi Monte Carlo Integration

General idea

- Similar to Monte Carlo.
- Rely on ideas from number theory and Fourier analysis.
- Main difference: use low-discrepancy sequences instead of pseudo-random sequences.
- Low-discrepancy sequence: a sequence with the property that for all values of N, its subsequence $x_1, ..., x_N$ has a low discrepancy with respect to interval $[a, b]$:

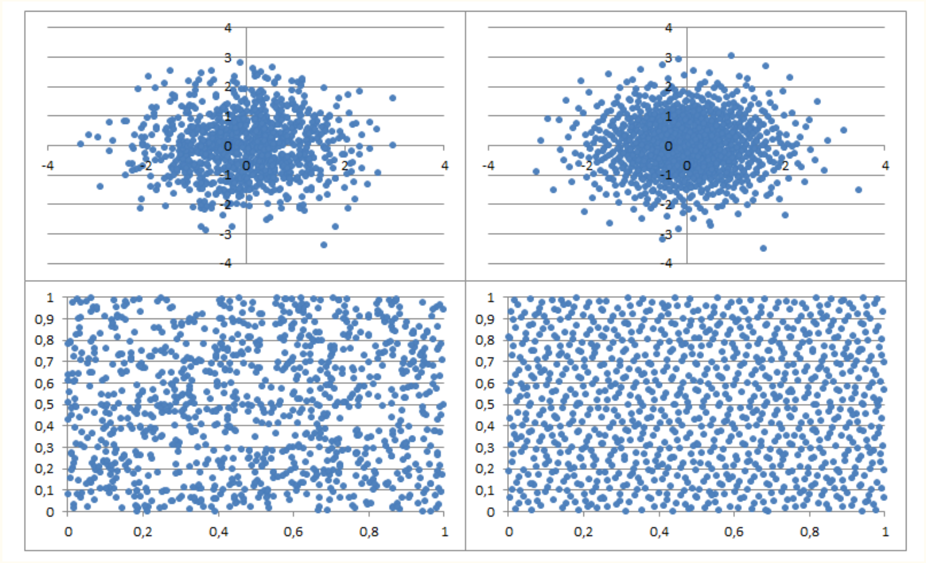
$$D_N = \sup_{a \leq c \leq d \leq b} \left| \frac{\# \left| \{x_1, ..., x_N\} \cap [c, d] \right|}{N} - \frac{d-c}{b-a} \right|$$

- Compare with equidistributed sequence (which we cannot use).
- Intuition.

More resources

- Better behavior than Monte Carlo.
- But often difficult to apply.
- Main choices: Halton sequence, Sobol sequence, and Faure sequence.
- Check:
    1. http://mikejuniperhill.blogspot.com/2014/03/using-c-nag-random-numbers-in-excel.html
    2. https://www.rdocumentation.org/packages/randtoolbox/versions/1.17/topics/quasiRNG

Sobol vs. pseudorandom


4
3
2
1
0
-1
-2
-3
-4
-4
-2
0
2
4
1
0,9
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
0
0,2
0,4
0,6
0,8
1