Differentiation and Integration

(Lectures on Numerical Analysis for Economists II)

Jesús Fernández-Villaverde\textsuperscript{1} and Pablo Guerrón\textsuperscript{2}

September 20, 2020

\textsuperscript{1}University of Pennsylvania

\textsuperscript{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.

- 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.

- 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:

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

\[
f(x + h) = f(x) + f'(x)h + \frac{f''(\xi)}{2} h^2, \quad \xi \in (x, x + h)
\]

• 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 \( \epsilon \) is the machine precision (I am skipping proof, standard result).
further accuracy, instead of the standard purely imaginary approach of the aforementioned papers. As with the standard complex-step approach, all of the new first-derivative approximations are not subject to roundoff errors. However, they all have a wider range of accuracy for larger step-sizes than the standard imaginary-only approach. The new second-derivative approximations are more accurate than both the imaginary-only as well as traditional higher-order finite-difference approaches. For example, a new 4-point second-derivative approximation is derived whose accuracy is valid up to tenth-order derivative errors. These new expressions allow a designer to choose one step-size in order to provide very accurate approximations, which minimizes the required number of function evaluations.

Reference 10 incorporates up to three function variables as quaternion vector components and evaluates the function using quaternion algebra. Using this method, the Jacobian matrix can be obtained with a single function evaluation, which is far less than the one presented here. On the contrary, the method presented in this paper has no limitation on number of function variables and does not require programming of quaternion algebra, besides standard complex algebra that is often built into existing mathematical libraries of common programming languages. Consequently, the method presented here is more practical for large multi-variable functions. In addition, the Jacobian matrix is simply a byproduct of the determined Hessian matrix. This paper extends the results from Ref. 11 under a generalized framework and thus supersedes it. This work also verifies the optimality of results from that paper.

The organization of this paper proceeds as follows. First, the complex-step approximation for the first derivative of a scalar function is summarized, followed by the derivation of the second-derivative approximation. Then, the Jacobian and Hessian approximations for multi-variable functions are derived. Next, several extensions of the complex-step approximation are shown. A numerical example is then shown that compares the accuracy of the new approximations to standard finite-difference approaches. Then, the second-order Kalman filter is summarized. Finally, simulation results are shown that compare results using the complex-step approximations versus using and standard finite-difference approximations in the filter design.

II. Complex-Step Approximation to the Derivative

In this section the complex-step approximation is shown. First, the derivative approximation of a scalar variable is summarized, followed by an extension to the second derivative. Then, approximations for multi-variable functions are presented for the Jacobian and Hessian matrices.
“Backward differencing” scheme

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

Extension to multivariate functions is:

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

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:

\[ \frac{\partial f(x)}{\partial x_i} \approx \frac{f(x_1, \ldots, x_i + h_i, \ldots, x_n) - f(x_1, \ldots, x_i - h_i, \ldots, x_n)}{h} \]

• Rule for selecting \( h \):

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

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:

\[
\text{Im}(f(x + ih)) = \text{Im}\left(f(x) + f'(x)ih + \frac{f''(\xi)}{2}h^2 + ...ight)
= f'(x)h + ...
\]

- Dividing by \( h \) and reordering:

\[
f'(x) = \frac{\text{Im}(f(x + ih))}{h} + O(h^2)
\]
Following the same logic, we can get second derivative:

\[ f''(x) = \frac{2}{h^2} \ast (f(x) - \text{Real}(f(x + ih))) \]

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.
Many programming languages have symbolic math packages/toolboxes.

1. C++: GiNaC.
2. Python: SymPy.
4. R: Racayas.

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:

\[
xy^2 + \log x = w_1 (w_2)^2 + \log w_1 = w_3 + w_4 = w_5
\]

- We want to compute the derivative of this function with respect to \( x \).
Example

<table>
<thead>
<tr>
<th>Operations to compute value</th>
<th>Operations to compute derivative</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_1 = x$</td>
<td>$w'_1 = 1$ (seed)</td>
</tr>
<tr>
<td>$w_2 = y$</td>
<td>$w'_2 = 0$ (seed)</td>
</tr>
<tr>
<td>$w_3 = w_1 (w_2)^2$</td>
<td>$w'_3 = w'_1 (w_2)^2 + 2w_1 w'_2$</td>
</tr>
<tr>
<td>$w_4 = \log w_1$</td>
<td>$w'_4 = \frac{w'_1}{w_1}$</td>
</tr>
<tr>
<td>$w_5 = w_3 + w_4$</td>
<td>$w'_5 = w'_3 + w'_4$</td>
</tr>
</tbody>
</table>

- We get:

\[
\begin{align*}
    w'_5 &= w'_3 + w'_4 \\
    &= w'_1 (w_2)^2 + 2w_1 w'_2 + \frac{w'_1}{w_1} \\
    &= (w_2)^2 + \frac{1}{w_1} \\
    &= y^2 + \frac{1}{x}
\end{align*}
\]

- 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 $\omega_i$. 
Newton-Cotes Formulas

• Idea: Approximate function with low order polynomials and then integrate approximation

• Step function approximation:
  — compute constant function equalling $f(x)$ at midpoint of $[a, b]$
  — Integral approximation is $\int_a^b f(x) \, dx$

• Linear function approximation:
  — compute linear function agreeing with $f(x)$ at $a$ and $b$
  — Integral approximation is trapezoid $\frac{b-a}{2} f(a) + f(b)$

• Parabolic function approximation:
  — compute parabola agreeing with $f(x)$ at $a, b,$ and $(a + b)/2$
  — Integral approximation is area of $\int_a^b f(x) \, dx$
Midpoint rule

- Simplest (open) rule with one interval:
  \[
  \int_a^b f(x) \, dx = (b - a)f \left( \frac{a + b}{2} \right) + \frac{(b - a)^3}{24} f''(\xi)
  \]

  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.
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.
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

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

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

Other rules: Simpson’s 3/8 rule and Boole’s rule.

Also, we can implement change of variables.
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 \( \omega_i \).

- 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:
\[
\int_{-1}^{1} f(x) (1 - x^2)^{-\frac{1}{2}} \, dx
\]

- 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 \( i = 1, \ldots, n \).

- 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
  \[ \int_{-\infty}^{\infty} f(x) e^{-x^2} \, dx \]
  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 \( \int_a^b f(x) \, dx \).

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

where once again the \( x_i \) are nodes and the \( \omega_i \) 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, usually 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

$$\sigma^2_{\hat{I}_f} = \frac{1}{N} \int_0^1 (f(x) - l_f)^2 \, dx = \frac{1}{N} \sigma^2_f$$
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 0.097540404999410?
- 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:
- Use state-of-art random number generators. It matters!
Stratified 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

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

Calculate $\int_0^1 e^x \, dx$

<table>
<thead>
<tr>
<th>$N$</th>
<th>Midpoint</th>
<th>Trapezoid</th>
<th>Simpson’s</th>
<th>Monte Carlo</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>-0.00071574</td>
<td>0.00143166</td>
<td>0.00000006</td>
<td>0.09523842</td>
</tr>
<tr>
<td>100</td>
<td>-0.00000716</td>
<td>0.00001432</td>
<td>0.00000000</td>
<td>0.01416057</td>
</tr>
<tr>
<td>1000</td>
<td>-0.00000007</td>
<td>0.00000014</td>
<td>-0.00000000</td>
<td>-0.00515829</td>
</tr>
<tr>
<td>5000</td>
<td>-0.00000000</td>
<td>0.00000001</td>
<td>0.00000000</td>
<td>0.00359500</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$N$</th>
<th>Midpoint</th>
<th>Trapezoid</th>
<th>Simpson’s</th>
<th>Monte Carlo</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.00</td>
<td>0.02</td>
<td>0.01</td>
<td>0.00</td>
</tr>
<tr>
<td>100</td>
<td>0.02</td>
<td>0.01</td>
<td>0.02</td>
<td>0.00</td>
</tr>
<tr>
<td>1000</td>
<td>0.01</td>
<td>0.04</td>
<td>0.01</td>
<td>0.02</td>
</tr>
<tr>
<td>5000</td>
<td>0.04</td>
<td>0.07</td>
<td>0.06</td>
<td>0.01</td>
</tr>
</tbody>
</table>
Comparison: 2 dimensions

Calculate $\int_0^1 \int_0^1 e^x e^y \, dx \, dy$

<table>
<thead>
<tr>
<th>N</th>
<th>Midpoint</th>
<th>Monte Carlo</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>-0.00245918</td>
<td>0.33897914</td>
</tr>
<tr>
<td>100</td>
<td>-0.00002460</td>
<td>0.03021147</td>
</tr>
<tr>
<td>1000</td>
<td>-0.00000025</td>
<td>-0.05486922</td>
</tr>
<tr>
<td>5000</td>
<td>-0.00000001</td>
<td>0.01183325</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N</th>
<th>Computation Time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.00</td>
</tr>
<tr>
<td>100</td>
<td>0.11</td>
</tr>
<tr>
<td>1000</td>
<td>9.18</td>
</tr>
<tr>
<td>5000</td>
<td>229.14</td>
</tr>
</tbody>
</table>
## More Monte Carlo draws

<table>
<thead>
<tr>
<th>N</th>
<th>Approximation Error</th>
<th>Computation Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.33897914</td>
<td>0.00</td>
</tr>
<tr>
<td>100</td>
<td>0.03021147</td>
<td>0.00</td>
</tr>
<tr>
<td>1000</td>
<td>-0.05486922</td>
<td>0.00</td>
</tr>
<tr>
<td>10000</td>
<td>0.00365290</td>
<td>0.00</td>
</tr>
<tr>
<td>100000</td>
<td>-0.00177819</td>
<td>0.03</td>
</tr>
<tr>
<td>1000000</td>
<td>-0.00177012</td>
<td>0.25</td>
</tr>
<tr>
<td>10000000</td>
<td>0.00065619</td>
<td>3.39</td>
</tr>
<tr>
<td>100000000</td>
<td>-0.00007068</td>
<td>24.94</td>
</tr>
</tbody>
</table>
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 $x_1, \ldots, x_N$ has a low discrepancy with respect to interval $[a, b]$:

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

- Compare with equidistributed sequence (which we cannot use).
- Intuition.
• Better behavior than Monte Carlo.

• But often difficult to apply.

• Main choices: Halton sequence, Sobol sequence, and Faure sequence.

• Check:
  2. https://www.rdocumentation.org/packages/randtoolbox/versions/1.17/topics/quasiRNG
Sobol vs. pseudorandom