Numerical Integration

2026-05-06 · cheerful mango Haubentaucher

Smooth Operator.

The Why

In the previous chapter, we explored global optimization strategies to escape local minima. Imagine you are optimizing a function with thousands of tiny, sharp local minima like a high-frequency version of Shekel's Foxholes. A gradient descent algorithm will get stuck instantly in the first microscopic pothole it finds.

Melts all your memories and change into gold

The solution for this is one that will feel familiar, but probably did go unnoticed until now: Numerical integration.

$$I = \int_a^b f(x)\,dx \tag{1}$$

This gives us good reasons to understand numerical integration,

- as it allows us to smoothen the loss landscape and make optimization methods more robust.
- as it tends to find more robust optima due to the averaging effect[1] of a region.

[1]

In deep learning stochastic gradient descent (SGD) is the workhorse optimization algorithm.

It is a Monte Carlo[2,3] method that estimates the gradient of the loss function by averaging over the gradient of a random mini-batch of data points. We are lucky to have numerical integration around, as it would be prohibitive expensive to train models on large-scale datasets.

Monte Carlo is a strategy which basically solves problems by random sampling. Embrace the beauty:

[2]

[3]

Hands On Experience

We already know how to discretize a continuous function, and how to handle finite precision and systems with sparse information at discrete points.

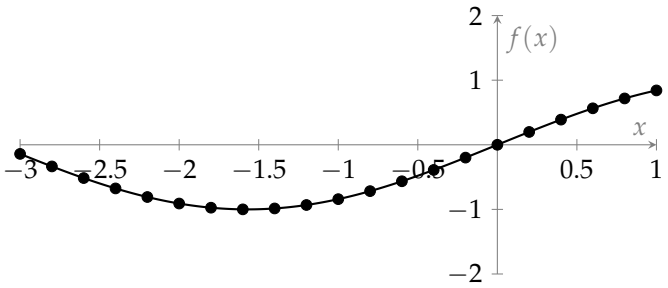

Figure 1: Continuous curve $f(x) = \sin(x)$ and its discretized version (black dots) on $[-3, 1]$ with step size $h = 0.2$.

The Shekel function, as a high-frequency function, did pose a challenge for global optimization, yet did not render our general approaches obsolete. We still discretize and treat these functions as we are used to.

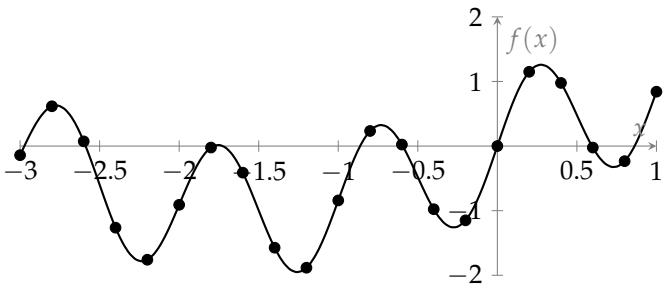

Figure 2: Continuous curve $f(x) = \sin(x) + \sin(2\pi x)$ and its discretized version (black dots) on $[-3, 1]$ with step size $h = 0.2$.

Taking a closer look at any local minima shows how the high-frequency loss landscape will nudge our optimization algorithm into the nearest local minimum. For example at $x = 0.6$ or $x = -0.4$.

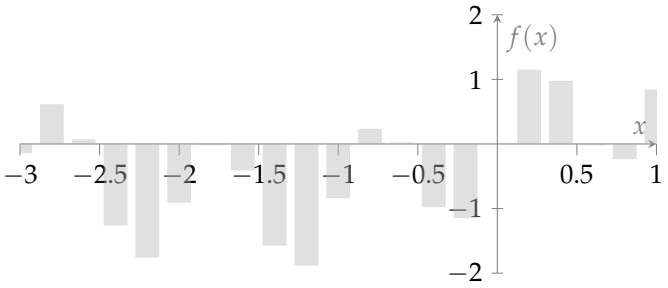

Figure 3: Continuous curve $f(x) = \sin(x) + \sin(2\pi x)$ and its discretized version (black dots) on $[-3, 1]$ with step size $h = 0.2$.

CONSIDER APPROXIMATION BY NUMERICAL INTEGRATION. Let's see what it does at $x = -0.4$, instead of solving directly for:

$$f(-0.4) = -0.97720, \tag{2}$$

let's approximate the value through integration with width $h = 0.2$:

$$\begin{aligned} h \cdot f(-0.4) &\approx \int_{-0.5}^{-0.3} f(x)\,dx \\ &\approx \frac{h}{2}[f(-0.5) + f(-0.3)] \\ &\approx 0.2 \cdot \frac{-1.14973 - 0.97720}{2} \\ &= -0.11285 \end{aligned}$$

$$\begin{aligned} f(-0.4) &\approx \frac{-1.14973 - 0.97720}{2} \\ &= -1.06347 \end{aligned}$$

THIS IS A MUCH BETTER APPROXIMATION having global optima in mind, than the direct evaluation at $x = -0.4$ which gave us -0.97720. The numerical integration gave us a correction of the local loss landscape towards higher losses at this local minima. The next figure shows the effect of this smoothing on the whole curve.

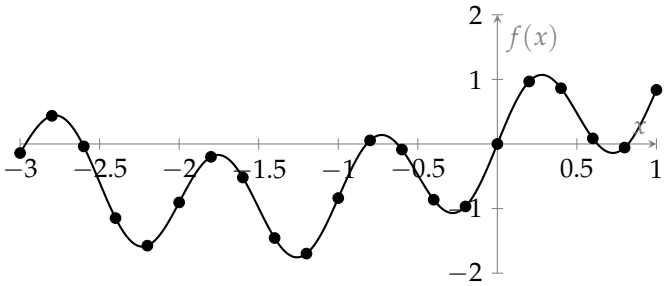

Figure 4: Continuous curve $f(x) = \sin(x) + \sin(2\pi x)$ and its discretized version (black dots) on $[-3, 1]$ with step size $h = 0.2$, and smoothed by numerical integration with interval length h.

THE SMOOTHING becomes even more apparent when integrating over larger intervals in general, or in our engineered example when choosing h such that the frequency of the noisy signal gets averaged out.

WHILE YOU HAVE seen the smoothing effect driven by the interval size, the approximation of the interval so far is a pretty coarse approximation, relying on two points only.

WHAT IF, we would be integrating over an interval which results in destructive inference of the underlying higher frequency sine curve? Think wavelengths.

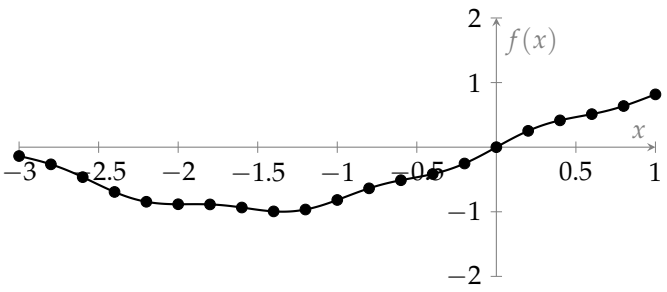

Figure 5: Continuous curve $f(x) = \sin(x) + \sin(2\pi x)$ and its discretized version (black dots) on $[-3, 1]$ with step size $h = 0.2$, smoothed by neighbour averaging with $h = 0.48$ (so $h/2 = 0.24$ gets close to cancel the $\sin(2\pi x)$ component).

The Learning Objectives of this chapter will provide you with the abilities to:

- Derive and apply the midpoint method, trapezoid and Simpson's rules for numerical integration
- Understand composite rules for improved accuracy
- Know about Lagrange interpolation and how it relates to quadrature rules
- Understand how the curse of dimensionality motivates Monte Carlo
- Derive and apply Monte Carlo integration for high-dimensional problems
- Apply Monte Carlo integration and recognize that batch averaging in ML is numerical integration

Methods of Numerical Intgration

THE MOST TRIVIAL APPROXIMATION OF INTEGRALS is using a single discretized value.

$$I = \int_a^b f(x)\,dx.$$
$$I = \frac{b-a}{n} \cdot f\left(\frac{a+b}{2}\right)$$
$$I = h \cdot f(m)$$

It is easy to see that the height of the rectangle is the value at the midpoint, and the width is h.

We can further refine the approximation of the integral by dividing $[a, b]$ into n subintervals of equal width h, and summing the contributions from each interval. This is also known as Riemann sum approximation, and can be expressed as:

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} f(x)\,dx$$
$$\approx \sum_{i=0}^{n-1} h \cdot f(m_i)$$

where $x_i = a + ih$ for $i = 0, 1, \ldots, n$.

THE MIDPOINT RULE of these composite intervals uses the midpoint, which is the average of the endpoints, of each interval, which often gives better results than left or right endpoint methods. Intuitively, this is because the midpoint approximates the curvature of the function better on the interval.

ON THE INTERVAL $[a, b]$, the midpoint value is $f(m)$. With left endpoint, the height is $f(a)$; with right endpoint, the height is $f(b)$.

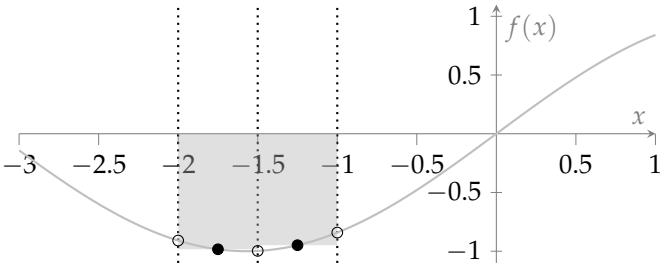

Figure 6: Continuous curve $f(x) = \sin(x)$ and midpoint rule integrals (gray bars) for $a = -2$, $b = -1$, $h = 0.5$. Black dots: midpoints. Circles: endpoints. The bars now touch, illustrating the midpoint rule without a gap.

INCREASING THE NUMBER OF SUBINTERVALS reduces the error.

DOUBLING THE SUB INTERVALS reduced error by $\sim 4\times$. This suggests $O(h^2)$ convergence.

This of course comes at the cost of more function evaluations, and thus more computational cost. So can we do better than approximating midpoints?

TRAPEZOIDS are more expressive geometrically than rectangles, and can capture linear changes in the function between the endpoints.

$$\int_a^b f(x)\,dx \approx \frac{h}{2}\left[f(a) + f(b)\right] \tag{3}$$

where $h = b - a$.

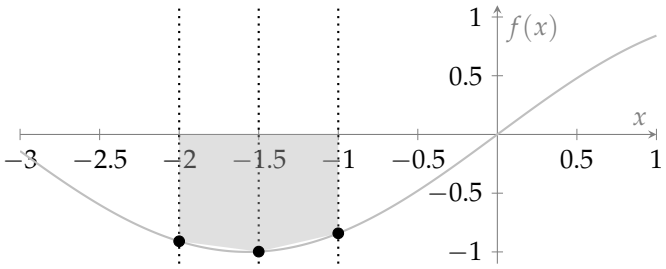

Figure 7: Continuous curve $f(x) = \sin(x)$ and trapezoid rule areas (gray trapezoids) for $a = -2$, $b = -1$, $h = 0.5$. Black dots: endpoints. The area under the straight lines connecting endpoints illustrates the trapezoid rule approximation.

COMPOSITE TRAPEZOID RULE again applies the trapezoid rule to each subinterval and sums the results.

$$\begin{aligned}
\int_a^b f(x)\,dx &\approx \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} f(x)\,dx \\
&\approx \sum_{i=0}^{n-1} \frac{h}{2}\left[f(x_i) + f(x_{i+1})\right] \\
&= \frac{h}{2}\left(f(x_0) + f(x_1)\right) + \frac{h}{2}\left(f(x_1) + f(x_2)\right) + \cdots + \frac{h}{2}\left(f(x_{n-1}) + f(x_n)\right) \\
&= \frac{h}{2}\left[f(x_0) + 2f(x_1) + 2f(x_2) + \cdots + 2f(x_{n-1}) + f(x_n)\right] \\
&= \frac{h}{2}\left[f(a) + 2\sum_{i=1}^{n-1} f(x_i) + f(b)\right]
\end{aligned}$$

where $x_i = a + ih$ for $i = 0, 1, \ldots, n$.

The function values at the endpoints a and b are each weighted by 1 in the formula, while all interior points are weighted by 2. Intuitively, this is because each interior point contributes to two adjacent trapezoids, while the endpoints only contribute to one trapezoid each. In your mind step through the circled enpoints in the figure above, and see how the interior points get counted twice, once as a right endpoint and once as a left endpoint, while the endpoints only get counted once.

DIVIDING THE INTEGRAL BY h gives us a weighted average of the function values, which is a better approximation to the integral than just using the midpoint or endpoints. Implicitly reflecting local curvature and information about the function's behavior across the interval.

Simpson's rule can capture curvature and thus often provides a much better approximation than the trapezoid rule. Following the idea of fitting a linear function, we fit a quadratic polynomial through 3 points instead of a line through 2. You will see Simpson's rule is exact for quadratic polynomials.

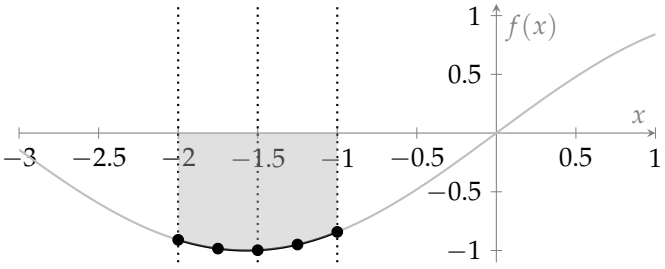

Figure 8: Continuous curve $f(x) = \sin(x)$ and Simpson's rule areas (gray, under the parabolas) for $a = -2$, $b = -1.5$, $h = 0.5$ and $a = -1.5$, $b = -1$, $h = 0.5$. Black dots: endpoints. Black dots: midpoints at $x = -1.75$ and $x = -1.25$. Solid line at $x = -1.5$ separates the two intervals. Each parabola illustrates the quadratic interpolant used by Simpson's rule on its subinterval.

Lagrange interpolation and quadrature. Lagrange interpolation constructs the unique polynomial of degree $\leq n$ that passes through given nodes $\{(x_j, y_j)\}_{j=0}^{n}$ using the basis

$$L_j(x) = \prod_{\substack{m=0 \\ m \neq j}}^{n} \frac{x - x_m}{x_j - x_m}. \tag{4}$$

The general form of a quadratic polynomial is:

$$f(x) = f(a) \cdot \frac{(x-m)(x-b)}{(a-m)(a-b)} + f(m) \cdot \frac{(x-a)(x-b)}{(m-a)(m-b)} + f(b) \cdot \frac{(x-a)(x-m)}{(b-a)(b-m)}$$

Try with some points. Start with $x = 0$ and see how the linear factors play out.

For a symmetric interval, the area under $f(x)$ is best approximated by giving the endpoints weight $h/6$ each and the midpoint weight $2h/3$: $A = C = h/6$, $B = 2h/3$ - we can verify this by integrating the Lagrange basis polynomials over $[a, b]$ and confirming that they yield these weights, but we omit the detailed calculation here. In a symmetric interval, the nodes are equally spaced: $x_0 = a$, $x_1 = m = \frac{a+b}{2}$, $x_2 = b$, so that the midpoint satisfies $m - a = b - m = h$.

Thus,

$$\begin{aligned} \int_a^b f(x)\,dx &\approx \frac{h}{6} f(a) + \frac{2h}{3} f(m) + \frac{h}{6} f(b) \\ &= \frac{h}{6}[f(a) + 4f(m) + f(b)] \end{aligned}$$

Composite Simpson's Rule again applies Simpson's rule on each pair of subintervals and sums the results (assume n is even and $x_i = a + ih$). Starting from the integral:

$$\begin{aligned}\int_a^b f(x)\,dx &\approx \sum_{i=0}^{n-1}\int_{x_i}^{x_{i+1}} f(x)\,dx \\ &\approx \sum_{k=0}^{n/2-1}\frac{h}{6}\left[f(x_{2k})+4f(x_{2k+1})+f(x_{2k+2})\right] \\ &= \frac{h}{6}\Big[f(x_0)+4\sum_{k=0}^{n/2-1}f(x_{2k+1})+2\sum_{k=1}^{n/2-1}f(x_{2k})+f(x_n)\Big] \\ &= \frac{h}{3}\left[f(x_0)+4\sum_{\substack{i=1\\ i\text{ odd}}}^{n-1}f(x_i)+2\sum_{\substack{i=2\\ i\text{ even}}}^{n-2}f(x_i)+f(x_n)\right].\end{aligned}$$

What happens, when you approximate a linear function with a quadratic?

PRACTICAL ADVICE: Stop at Simpson. For higher accuracy, use composite rules with more subintervals. High-degrees ($n \geq 8$) develop negative weights and become unstable.

RUNGE'S PHENOMENON: High-degree polynomial interpolation oscillates due to overfitting, leading to large errors at the edges of the interval.

Method	**Single Interval Error**	**Accumulated Error**
Left/Right	$O(h^2)$	$O(h)$
Midpoint	$O(h^3)$	$O(h^2)$
Trapezoid	$O(h^3)$	$O(h^2)$
Simpson's 1/3	$O(h^5)$	$O(h^4)$
In General (Gaussian Quadrature)	$O(h^{2n})$	$O(h^{2n-1})$

Table 1: Comparison of quadrature method accuracies. Higher-order methods achieve given accuracy with fewer function evaluations. h is the step size, and n is the number of evaluation points for Gaussian quadrature.

Monte Carlo & the Curse of Dimensionality

FOR d-DIMENSIONAL INTEGRALS, deterministic quadrature with N points per dimension needs N^d total points. There is no way to do this, let alone iterate on it during iterative optimization.

A NEURAL NETWORK WITH $d = 10^6$ or 1 million parameters would need $(N)^{10^6}$ grid points. Even $N = 2$ gives 2^{10^6}, a number with 300,000 digits.

RANDOM SAMPLING allows to estimate the integral without evaluating it on a full grid. The integral can be rewritten as an expectation:

$$I = \int_\Omega f(\mathbf{x})\,d\mathbf{x} = |\Omega| \cdot \mathbb{E}_{\mathbf{X}\sim\text{Uniform}(\Omega)}[f(\mathbf{X})]. \tag{5}$$

This identity means the definite integral equals the domain volume times the average value of f under a uniform draw from Ω. Practically this motivates a sampling estimator: draw points uniformly in Ω, compute f at those points, average the results, and multiply by $|\Omega|$ to estimate the integral.

Ω is the domain of integration, and $|\Omega|$ is its length. For a 1D integral over $[a, b]$, we have $|\Omega| = b - a$. For higher dimensions, it's the product of the lengths of each dimension.

MONTE CARLO UNIFORM SAMPLING on $[-2, -1]$.

Let $X \sim \text{Uniform}(-2, -1)$. Then

$$I = \frac{|\Omega|}{N} \sum_{i=1}^{N} f(\mathbf{X}_i), \qquad \mathbf{X}_i \overset{\text{i.i.d.}}{\sim} \text{Uniform}(\Omega). \tag{6}$$

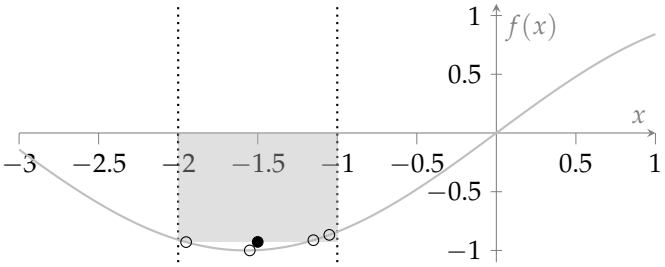

Figure 9: Continuous curve $f(x) = \sin(x)$ and midpoint rule integrals (gray bars) for $a = -2$, $b = -1$, $h = 0.5$. Circles: endpoints. The bars now touch, illustrating the midpoint rule without a gap.

MIDPOINT RULE is a special case of Monte Carlo with $N = 1$ sample at the midpoint.

BIAS. Because expectation is linear, this estimator is unbiased, meaning its expected value equals the true integral, for the number of $N \to \infty$:

$$\mathbb{E}[\hat{I}_N] = I. \tag{7}$$

I.I.D, independent and identically distributed, means that all $\mathbf{X}_i$ are not correlated and that they all come from the same underlying distribution.

VARIANCE. The variance, a measure of the spread of the estimator around its mean, or how how large the error of the estimator can be, follows from independence of the samples:

$$\text{Var}(\hat{I}_N) = \frac{|\Omega|^2}{N} \text{Var}\big(f(\mathbf{X})\big) = \frac{|\Omega|^2 \sigma_f^2}{N}, \tag{8}$$

where $\sigma_f^2 = \text{Var}(f(\mathbf{X}))$. Hence the standard error is

$$\text{SE}(\hat{I}_N) = \frac{|\Omega|\, \sigma_f}{\sqrt{N}}. \tag{9}$$

DIMENSION INDEPENDENT. It is trivial to see, that for large N the sampling error converges with, $O(1/\sqrt{N})$ for Monte Carlo, which makes it attractive in high dimensions. Quadrature rules outperform in terms of their convergence rates, but the convergence effort explodes with the dimension, as we need N^d points to maintain the same accuracy.

CONVERGENCE IN SGD. Mini-batch gradients used in SGD are Monte Carlo estimates of the true gradient, based on a samples drawn into a batch. Increasing the batch size reduces variance roughly by $1/b$, directly analogous to the $1/N$ variance scaling above. This connection explains why batch size controls the noise–variance trade-off in training.

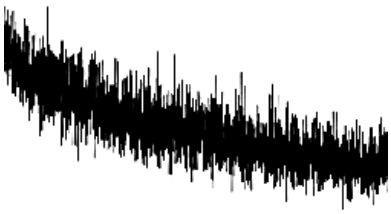

Figure 10: Effect of batch size on gradient noise: smaller batches produce larger variance (showing up in the width of the training loss band) in the gradient estimate (Source: Stanford CS231n Note).

As a beneficial side effect, computational cost is n/b times cheaper per update step. For $n = 10^6$, and $b = 100$: SGD does 10,000 iterations with the compute GD needs for 1.

Examples & Exercises

LET'S STICK to our example from the methods sections, and calculate the integral of $\sin(x)$ from -2 to -1 using different methods. In order to be able to quantify the error of each method, we need the exact value. So let's compute the exact value of the integral:

$$\begin{aligned}\tilde{I} &= \int_{-2}^{-1} \sin(x)\,dx \\ &= [-\cos(x)]_{-2}^{-1} \\ &= -\cos(-1) + \cos(-2) \\ &\approx -0.9564491424.\end{aligned}$$

MIDPOINT RULE BY HAND. Compute $\int_{-2}^{-1} \sin(x)dx$ using the midpoint rule with $n = 1$ interval. The midpoint is $m = (-2 + (-1))/2 = -1.5$, so the midpoint rule gives us:

$$\begin{aligned}\hat{I} &\approx (b-a) \cdot f(m) \\ &= 1 \cdot \sin(-1.5) \\ &\approx -0.99749.\end{aligned}$$

The error is $|\tilde{I} - \hat{I}| \approx 0.04104$, which is the small difference between the midpoint estimate and the true value.

FURTHER THINGS WORTH TRYING are to calculate the approximation on the endpoints of the interval, comparing the error.

MIDPOINT RULE WITH MORE INTERVALS. Now compute the midpoint rule with $n = 2$ intervals, which means we will have two midpoints at -1.75 and -1.25. This gives us:

$$\begin{aligned}h &= 0.5 \\ \hat{I} &\approx h \cdot [f(-1.75) + f(-1.25)] \\ &= 0.5 \cdot [\sin(-1.75) + \sin(-1.25)] \\ &\approx -0.927228.\end{aligned}$$

The error is $|\tilde{I} - \hat{I}| \approx 0.02922$, which is smaller than the error with $n = 1$ interval, illustrating how composites, or increasing the number of intervals improves the approximation.

TRAPEZOID BY HAND. Use the trapezoid rule to compute $\int_{-2}^{-1} \sin(x)dx$ with $n = 2$ intervals. The endpoints are -2, -1.5, and -1. The trape-

zoid rule gives us:

$$\begin{aligned} h &= 0.5 \\ T_2 &= \frac{h}{2}[f(-2) + 2f(-1.5) + f(-1)] \\ &= 0.25 \cdot [-0.90930 + 2(-0.99749) + (-0.84147)] \\ &\approx -0.927228. \end{aligned}$$

The error is the same as the midpoint rule with $n = 2$ intervals, which is not necessarily a coincidence, as both methods are second-order accurate and can yield similar results for certain functions and interval choices.

SIMPSON'S RULE BY HAND. Use Simpson's rule to compute $\int_{-2}^{-1} \sin(x)dx$ with $n = 2$ intervals. The endpoints are -2, -1.5, and -1. Simpson's rule gives us:

$$\begin{aligned} h &= 0.5 \\ S_2 &= \frac{h}{3}[f(-2) + 4f(-1.5) + f(-1)] \\ &= \frac{0.5}{3} \cdot [-0.90930 + 4(-0.99749) + (-0.84147)] \\ &\approx -0.9564491424. \end{aligned}$$

The error is $|\tilde{I} - S_2| \approx 0.0000000000$, which is essentially zero, illustrating that Simpson's rule is exact for this particular integral, as $\sin(x)$ can be well approximated by a quadratic polynomial over the interval $[-2, -1]$.

TRY WITH MORE INTERVALS, see how the error decreases and check whether you know your way around composites in Simpson's and Trapezoid rules.

MONTE CARLO BY HAND. Use Monte Carlo estimation to compute $\int_{-2}^{-1} \sin(x)dx$ with $N = 4$ random samples, which is similar to the compute efforts of the trapezoid method. Let's assume the random samples are:

$$\begin{aligned} X_1 &= -1.95, \quad f(X_1) = \sin(-1.95) \approx -0.92895, \\ X_2 &= -1.55, \quad f(X_2) = \sin(-1.55) \approx -0.99978, \\ X_3 &= -1.15, \quad f(X_3) = \sin(-1.15) \approx -0.91276, \\ X_4 &= -1.05, \quad f(X_4) = \sin(-1.05) \approx -0.86742. \end{aligned}$$

The Monte Carlo estimate is:

$$\begin{aligned} \hat{I} &= (b-a) \cdot \frac{1}{N} \sum_{i=1}^{N} f(X_i) \\ &= 1 \cdot \frac{1}{4}(-0.92895 - 0.99978 - 0.91276 - 0.86742) \\ &\approx -0.927228. \end{aligned}$$

The error is $|\tilde{I} - \hat{I}| \approx 0.02922$, which is the same as the midpoint method.

ALTERNATIVE ERROR ESTIMATION FOR MONTE CARLO. For our concrete example with $N = 4$ and the sample mean of $\bar{f} \approx -0.927228$. Using the unbiased sample variance estimator:

$$Var(f(\mathbf{X})) = \frac{1}{N-1} \sum_{i=1}^{N} (f(X_i) - \bar{f})^2. \tag{10}$$

We obtain the sample variance and standard error of the Monte Carlo estimator as follows:

$$\begin{aligned} \sigma_f^2 &= Var(f(\mathbf{X})) \\ &\approx 0.003036, \\ \sigma_f &\approx 0.05512, \\ \mathrm{SE}(\hat{I}_N) &\approx \frac{|\Omega|\,\sigma_f}{\sqrt{N}} \\ &= \frac{1 \cdot 0.05512}{2} \\ &\approx 0.02756. \end{aligned}$$

What happens if we increase N to 16? The error should decrease by a factor of 2, since the standard error scales as $1/\sqrt{N}$. Briefly elaborate on how the error of the Monte Carlo estimator is dimension independant.

ENTER HIGHER DIMENSIONS ON YOUR MACHINE. Implement Monte Carlo integration for a d-dimensional integral, such as integrating a multivariate Gaussian function over a hypercube. While slowly increasing d, observe how the error behaves for different methods. Even more, observe how grid-based quadrature methods become infeasible in terms of computational cost as d grows. Optionally, see how similar effects arise in optimization by implementing gradient descent and mini-batch stochastic gradient descent on a high-dimensional function. Only watch compute times.

CODE can be found at https://github.com/Quillstacks/lecturecode_numericalmethods.git.

Self-Reflection and Recap

SELF-REFLECTION Questions to guide your understanding:

- How does the error of the midpoint, trapezoid, Simpson's rule and Monte Carlo compare?
- Why has the midpoint rule better accuracy than the left or right endpoint rules?
- How does composite methods improve accuracy, and what is the trade-off?
- Why do deterministic quadrature methods fail in high dimensions?
- In which way, is the midpoint rule a special case of Monte Carlo estimation?
- What is the intuition behind Monte Carlo not exploding in high dimensions?
- When would you use Simpson's rule vs. Monte Carlo?
- How is the mini-batch mean in SGD related to Monte Carlo estimation?
- How does the gradient estimation in SGD help escape local minima?

RECAP of Key Concepts:

- Deterministic quadrature methods (midpoint, trapezoid, Simpson's) approximate integrals using weighted sums of function values at specific points. They are accurate and converge fast for smooth, low-dimensional problems.
- Monte Carlo integration estimates integrals by averaging function values at random samples. It is unbiased and has a convergence rate of $O(1/\sqrt{N})$, making it independent of the dimensionality of the problem.
- In SGD, mini-batch gradients are Monte Carlo estimates of the true gradient.

TEASER. Why do high-order quadrature methods become unstable when approximating low-degree polynomials?

SO FAR STABILITY referred to the sensitivity of the numerical solution to small changes in the input data. But we can also talk about stability in terms of the numerical method itself, and how it behaves

when approximating different types of functions. In the next chapter, we will see other types of stability issues that arise in numerical methods, and how to mitigate them by sophisticated method selection and configuration.