Numerical Analysis

Zac Zerafa

October 13, 2025

Contents

Part I

Numerical methods

For many problems of mathematical analysis, an analytic solution is unavailable and hence one may resort to numerical solutions. The general approach is to use a *numerical method* that converges towards the solution.

Definition 0.1. Given a well posed problem $F(x, y) = 0$ with $F : X \times Y \to \mathbb{R}$ and with some locally Lipshitz function from X to Y at every solution (x, y), a *numerical method* is a sequence of problems $(F_n(x_n, y_n) = 0)_{n\in\mathbb{N}}$. In other words, it is a sequence of (hopefully easier or analytically solved) problems whose solutions are used to approximate the problem $F(x, y) = 0$.

Numerical methods with the same objective may differ in quality, primarily with regard to:

- Numerical stability
- Rate of convergence

Numerical stability refers to whether...

A numerical method's rate of convergence describes, the rate at which absolute error decreases as the numerical method progresses to subsequent terms. Higher rates of convergence are highly desirable

A numerical method's rate of convergence describes

Chapter 1

Errors

One can analyze the efficacy of a numerical method by observing its error after n steps. There are two main ways in which error is quantified.

Definition 1.1. The *absolute error* of an estimation is its distance to the true value.

$$\|\mathbf{x} - \boldsymbol{\xi}\|$$

Definition 1.2. The *relative error* of an estimation is its distance to true value divided by the norm of the estimation.

$$\frac{\|\mathbf{x} - \boldsymbol{\xi}\|}{\|\mathbf{x}\|}$$

- Approximation (the use of an approximate function induces error)
- Truncation (error from terminating numerical method that converges to true value)
- Roundoff (error due to limitations of model of computation to store the true value)
- Statistical (error due to the randomness of a random sample possibly employed in the numerical method)

1.1 Roundoff error

In Python, 100000000000000000 is recognized as the same number as 100000000000000001. This is clearly false mathematically, however this phenomena occurs in real

application. This is due to the limited storage space designated to a number on a PC, meaning that only a finite amount of numers are recognizable. Different implementations have different consequences, however this phenomena is a downside of floating-point arithmetic specifically.

This book aims to discuss numerical analysis in a theoretical, mathematical environment, and Turing machines (basically 'theoretical computers') do not have issues with roundoff error. That said, if one wants to create a mathematical environment that emulates roundoff error, one can introduce a quantity called *machine epsilon*.

Definition 1.3. The *machine epsilon* ε of some model of computation is an upper bound on relative approximation error. A model of computation with machine epsilon ε cannot distinguish x, y if $|x - y| < |x|\varepsilon$

1.2 Approximation error

Many numerical methods draw from approximation theory to use similar functions that are 'nicer' to make calculations.

1.3 Truncation error

Numerical methods are employed in algorithms, and ideally the algorithms we use should terminate (at least in the context of numerical analysis).

- Finite iterations
- Reaching predefined tolerance

When a certain amount of error is acceptable, and one has an infinite numerical method, one may be able to calculate a value with arbitrary *tolerance* to the true value.

Definition 1.4. The *tolerance* ε is a value reperesenting the maximum error.

$$\|\mathbf{x} - \boldsymbol{\xi}\| \leq \varepsilon$$

2. rate of convergence - Rate of convergence - Aitken's delta squared process

Part II

Interpolation and Approximation theory

Chapter 2

Interpolation

Often one has a function of discrete 'data points' to evaluate a function from, however if one seeks to evaluate a function at a point between the given data points one resorts to *interpolation*; extending a function of discrete data points to a continuous domain to evaluate points within the range of the data points.

If one wishes to evaluate points above the largest data point or below the smallest data point, this becomes a problem of *extrapolation*.

2.1 Lagrange polynomial interpolation

Consider a vector $\mathbf{x}$ of domain elements with each entry mapped to the image element in $\mathbf{y}$. Note the following.

$$\mathbf{y}_i = f(\mathbf{x}_i)$$

$$U = \{(x_1, y_1), \ldots, (x_n, y_n)\}$$

$$\ell_n(x) = \frac{\prod_{i \neq n}(x - x_i)}{\prod_{i \neq n}(x_n - x_i)}$$

$$f(x) = \sum_{k=1}^{n} y_k \ell_k(x)$$

2.2 Vandermonde polynomial interpolation

Consider a vector $\mathbf{x}$ of domain elements with each entry mapped to the image element in $\mathbf{y}$. Note the following.

$$\mathbf{y}_i = f(\mathbf{x}_i)$$

Like Lagrange interpolation, we will attempt to use the theory of polynomails to interpolate f.

Proposition 2.1. For a set of n mappings of distinct domain elements , there is a unique polynomial of order $n-1$ that interpolates these mappings.

This gives rise to a strong idea; assume an $n-1$ degree polynomial and solve for its coefficients. This can indeed be reduced to system of linear equations and hence a simple matrix equation. We can express this problem by means of the *Vandermonde matrix*.

Definition 2.1. A *Vandermonde matrix* $\mathbf{V}$ is a matrix of the first n powers of a set of real numbers.

$$\mathbf{V}(\mathbf{x}) = \begin{bmatrix} 1 & \mathbf{x}_1 & \mathbf{x}_1^2 & \dots & \mathbf{x}_1^n \\ 1 & \mathbf{x}_2 & \mathbf{x}_2^2 & \dots & \mathbf{x}_2^n \\ \vdots & \ddots & \vdots & \vdots & \\ 1 & \mathbf{x}_n & \mathbf{x}_n^2 & \dots & \mathbf{x}_n^n \end{bmatrix}$$

$$\mathbf{V}(\mathbf{x})_{ij} = \mathbf{x}_i^j$$

We can now start to see that the vector of coefficients $\mathbf{c}$ is subject to $\mathbf{V}(\mathbf{x})\mathbf{c} = \mathbf{y}$. i have foreshadowed that there must exist a $n-1$ degree polynomial, so it must be that $\mathbf{V}(\mathbf{x})$ is invertible. Indeed, this checks out by the following proposition.

Proposition 2.2.

$$\det[\mathbf{V}(\mathbf{x})] = \prod_{0 \le i < j < n} (\mathbf{x}_j - \mathbf{x}_i)$$

Since we're dealing with distinct domain elements, the determinant is nonzero and hence invertible. We're then left with the following interpolation formula.

$$f(x) = \sum_{k=1}^{n} (\mathbf{V}^{-1}(\mathbf{x})\mathbf{y})_k x^k$$

2.3 Spline interpolation

Chapter 3

Approximation theory

3.1 Taylor series approximation

- Taylor series approximation

$$f(x) = \sum_{k=0}^{n} \frac{f^{(k)}(0)}{k!} x^k$$

3.2 Padé approximant

3.3 Stirling's formula

Part III

Numerical differentiation and integration

Chapter 4

Numerical differentiation

4.1 Forward difference

4.2 Backwards difference

4.3 Balanced difference

4.4 Higher order methods

Chapter 5

Numerical integration

5.1 Riemann sum

The definition of a Riemann integrable function offers a natural

$$\int_a^b f(x)dx \approx \sum_{k=1}^{n} f(x_k)h$$

$$h = \frac{b-a}{n}$$
$$x_k = a + nh$$

5.2 Trapezoidal rule

To reduce the error, one can consider the use of trapezoids along the mesh; hence the integral is estimated as a linear function over each partiton rather than as a constant function.

$$\int_a^b f(x)dx \approx \sum_{k=1}^{n-1} f(x_k)h$$

$$h = \frac{b-a}{n}$$
$$x_k = a + nh$$
$$f_k = \frac{f(x_k) + f(x_{k+1})}{2}$$

5.3 Refined trapezoidal rule

One can remodel the trapezoidal rule as an algorithm where tolerance is specified rather than the mesh refinement. The crux of the idea is to double the refinement of the mesh until the absolute error from the previous iteration is within tolerance, however there are two standard methods for iterating the algorithm:

- Weighted sum of previous value and Riemann sum with 'doubly refined mesh'
- Calculating trapezoidal rule for 'doubly refined mesh'

$$h_n = \frac{b-a}{2^n}$$

$$x_{n,k} = a + kh_n$$

$$I_0 = \frac{f(a)+f(b)}{2}(a-b)$$

$$I_{n+1} = \frac{I_n}{2} + \frac{h_{n+1}}{2}\sum_{i=1}^{2^{n+1}} f(x_{n+1,i})$$

5.4 Romberg integration

One can perform a series acceleration technique known as *Richardson extrapolation* to derive higher order schemes for numerical integration. The use of this technique in numerical integration is formalized as *Romberg integration*.

5.5 Simpson's rule

The theory of Romberg integration allows us to develop higher order schemes than the trapezoidal rule (0-order). *Simpson's rule* is the 1-order algorithm in the Romberg integration scheme, and is otherwise observed by fitting quadratics to each partition rather than lines.

Chapter 6

Advanced methods

6.1 Gauss-Legendre integration

An idea with strong link to functional analysis and the idea of orthogonal polynomials allows the realization of an extremely powerful method of numerical integration. We develop our theory within the Hilbert space $L^2([-1,1])$ (this just means we're going to consider integration problems on $[-1,1]$; one can do the change of variables later).

Consider a function f that can be represented well by a polynomial interpolation h. Let $(P_n)_{n\in\mathbb{N}}$ be the Legendre polynomials. Recall that $\int_{-1}^{1} P_n(x)x^k dx = 0$

By Euclidean polynomial division, we may have $h(x) = q(x)P_n(x) + r(x)$, where

$$\int_{-1}^{1} f(x)dx \approx \int_{-1}^{1} h(x)dx$$

$$= \int_{-1}^{1} q(x)P_n(x) + r(x)dx$$

$$= \int_{-1}^{1} q(x)P_n(x) + r(x)dx$$

$$= \int_{-1}^{1} r(x)dx$$

Since r is a polynomial of degree less than n, it can be interpolated *exactly* by some Lagrange interpolation.

$$r(x) = \sum_{i=1}^{n} \ell_i(x) r(x_i)$$

$$= \int_{-1}^{1} \sum_{i=1}^{n} \ell_i(x) r(x_i) dx$$

$$= \sum_{i=1}^{n} r(x_i) \int_{-1}^{1} \ell_i(x) dx$$

$$w_i = \int_{-1}^{1} \ell_i(x) dx$$

$$= \sum_{i=1}^{n} r(x_i) w_i dx$$

This is close, however we want to deal with the function h instead of r; how can we pick the x_i so that r coincides with h? Returning to the equation $h(x_i) = q(x_i)P_n(x_i) + r(x_i)$, we can see that one way to do this is by letting x_i be the zeroes of P_n!

$$\int_{-1}^{1} f(x) dx \approx \sum_{i=1}^{n} f(x_i) w_i dx$$

With the change of variables, we get the following.

$$\int_{a}^{b} f(x) dx = \approx \frac{b-a}{2} \sum_{i=0}^{n} w_i f(\frac{b-a}{2} u_i + \frac{a+b}{2})$$

6.2 Monte Carlo integration

To integrate for higher dimensions, one approach is to 'nest' singular dimension integrals by setting $n-1$ variables as constants, integrating over this function, changing these $n-1$ variables slightly, integrating again etc. Needless to say, this becomes quite computationally expensive.

Probability theory offers a unique approach to integration that converges relatively slowly, however scales well for higher dimensions. It is an interesting consequence of the law of large numbers that states that the volume ('size' of the domain of integration) multiplied by mapings of random points within that volume equals the desired integral. This is known as *Monte Carlo (MC) integration.*

$$\text{plim}_{n\to\infty} \frac{\sum_{i=1}^{n} f(\mathbf{u}_i)}{n} = \int_{\Omega} f(\mathbf{x}) d\mathbf{x}$$

Ω is the domain of integration (and simultaenously the sample space we will sample on) $U \sim \mathrm{U}(\Omega)$ is a random variable with a uniform distribution on Ω ($\mathbf{u}_i$) is a sequence of random samples of U $\lambda(\Omega)$ represents the 'volume' of Ω (technically the Lebesgue measure of Ω)

In practice, if one cannot generate uniform samples over the domain, a change of variables might be required; similar to Gaussian-Legendre quadrature.

There are methods that seek to reduce the error introduced by the nature of randomness, a prominent example being *MC variance reduction.*

Part IV

Numerical solutions to differential equation

- Euler's method - midpoint method - Runge-Kutta method

Part V

Root finding methods

- corrollary of IVT (bracketing method) - bracketing method - bisection method (IVT) - secant method - forward difference - backward difference - central difference - Newton-Raphson method - false position method - Dekker's method - Brent's method

Part VI

Numerical solutions to systems of equations

- power method

Chapter 7

Power method

It is possible to solve the eigenequation $\mathbf{Av} = \lambda\mathbf{v}$ analytically by the characteristic equation, however this can be computationally expensive since it requires calculating the determinant.

The *power method* is a numerical method that converges to the normalized eigenvector with the largest eigenvalue.

---

while $\|\mathbf{u} - \mathbf{v}\| \leq \varepsilon$ **do**
  $\mathbf{u} \leftarrow \mathbf{v}$
  $\mathbf{v} \leftarrow \frac{\mathbf{Av}}{\|\mathbf{Av}\|}$
end while

Chapter 8

Inverse power method

Perhaps one wants the eigenvector associated with the smallest eigenvalue. Sincethe eigenequation $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$ can be written equivalently as $\mathbf{A}^{-1}\mathbf{v} = \frac{1}{\lambda}\mathbf{v}$, and the power method converges to the eigenvector with the largest eigenvalue, employing the power method on the form $\mathbf{A}^{-1}\mathbf{v} = \frac{1}{\lambda}\mathbf{v}$ returns the eigenvector with the largest $\frac{1}{\lambda}$, or equivalently, with the smallest λ!

This is called the *inverse power method*.

---

while $\|\mathbf{u} - \mathbf{v}\| \leq \varepsilon$ **do**
  $\mathbf{u} \leftarrow \mathbf{v}$
  $\mathbf{v} \leftarrow \frac{\mathbf{A}^{-1}\mathbf{v}}{\|\mathbf{A}^{-1}\mathbf{v}\|}$
end while

Chapter 9

QR iteration

---

while Change this **do**
  A := **QR**
  A ← **RQ**
end while

- QR iteration - sparse matrix A sparse matrix is an $n \times m$ array that has may zero entries. In such cases, there may be more memory efficientr structures like a COO where one stores the row, column and data of a matrix entry. Or a CSC which stores the data of each matrix entry ordered by coulmn, and uses an array of ranges to determine which section COO w

- Row array
- Column array
- Data array
- Row array
- Data array (ordered by column)
- 'Column range' array

- Arnoldi iteration

Chapter 10

Arnoldi iteration

---

for Change this **do**
  $\mathbf{q}_k \leftarrow \mathbf{A}\mathbf{q}_{k-1}$
  for Change this **do**
    $\mathbf{q}_k \leftarrow \mathbf{q}_k - (\mathbf{q}_k^* \mathbf{q}_k)\mathbf{q}_k$
    $\mathbf{q}_{k+1} \leftarrow \frac{\mathbf{q}_k}{\|\mathbf{q}_k\|}$
    $\mathbf{A} \leftarrow \mathbf{R}\mathbf{Q}$
  end for
end for
Augment each $\mathbf{q}_k$ column-wise to create $\mathbf{H}$
Perform QR decomposition on $\mathbf{H}$

Part VII

Numerical optimization

10.1 Bracketing method

Before attempting to optimize a function, a *bracketing method* should be run on the interval of interest. This helps narrow down the bracket considerably before even attempting an specialized method.

The idea is to split an interval into fine partitions, and for each mesh see if the sign of the gradient from beggining to midpoint differs from that of midpoint to end. - bracketing method - numerical derivative test

10.2 Golden section search

Given a bracket containing a minimum, one aims to find a way to shrink this bracket.

- the two regions should have equal length
- the ratio of the nonoverlapping part of the region to the bracket the should be invariant for each iteration

These conditions imply that a region should take up $\frac{1}{\phi}$ of the bracket length.

For the minimum problem, if the function is seen to increase from the edge of region 2 to the edge or region 1, bracket the minima in region 1. Else bracket it in region 2.

10.3 Jarratt's method

Since quadratic functions are easy to minimize, one idea is to interpolate the function on its bracket as a quadratic, take the analytical mininum, and use that result to shrink the bracket. *Jarrat's method* embodies this concept. Quadratic interpolation - golden section search - Brent's method (optimization) - Direction set method - Powell's method - Nedler-meadle method