Practical Numerical Analysis

Alberto Paganini

November 22, 2017

Contents

1 Root finding

Let $f : \mathbb{R} \to \mathbb{R}$ be a given (possibly nonlinear) function. The goal is to find a point c such that $f(c) = 0$. For instance, f could be the function

$$f(x) = \cos(x) - x\,, \quad \text{or} \quad f(x) = \exp(x) - x\,, \quad \text{or} \quad f(x) = x^7 - 2\,, \quad \text{or similar.}$$

Besides solving nonlinear equations, root finding can be used in optimization to seek the root of the first derivative of a function to be maximized/minimized. For mathematical convenience, let assume that f is at least continuous.

Bisection algorithm: Assume that we have to points $a, b \in \mathbb{R}$ with $f(a)f(b) < 0$. Then, by the intermediate value theorem, we know that there is a $c \in (a, b)$ such that $f(c) = 0$. To find it, we could proceed as follow:

1. test if the middle point $(a + b)/2$ is a solution,
2. if not, tighten the interval $[a, b]$.

This can be realized with the following algorithm.

```
fa = f(a); fb = f(b);
c = (a+b)/2; fc = f(c);
while  fc != 0
    if fc*fa > 0
        a = c; fa = fc;
    else
        b = c;  fb = fc;
    end
    c = (a+b)/2; fc = f(c);
end
```

The previous algorithm uses `fc != 0` to test if c is a roof of f. The problem with this is that, due to round-off error, this condition will be very rarely satisfied, and the algorithm will never stop. A sounder test is to check that `abs(fc) < TOL`, where `TOL` is a prescribed (residual) tolerance (like 10^{-6} or `sqrt(eps)`). An alternative stopping criteria is `abs(a-b) < TOL`. In this case, `TOL` is a prescribed (solution) tolerance. Which criteria should be used is often problem (and user) specific.
Be aware that often users are too ambitious with these tolerances (especially if f cannot be evaluated accurately due to some approximation error). Therefore, it is better to stop the algorithm after a maximal number of iterations (and inform the user about it).
Finally, note that at each step we half the interval and hence the error after n iterations is

$$|c_n - c| \leq (b - a)/2^n\,,$$

which implies linear convergence with rate $1/2$.

Regula falsi: instead of choosing the midpoint of $[a, b]$, we can approximate f with the linear function $\tilde{f}(x) = f(a) + (x - a)(f(b) - f(a))/(b - a)$ and pick c as the root of $\tilde{f}$. Sometimes this works better, and sometimes not. An improved version that combines the bisection algorithm and the regula falsi is called **Illinois algorithm**.

Newton's method: in many instances, we can do much better if we have access to the first derivative of f (tacitly assuming that f is differentiable). Newton's method proceeds as follows: given an initial guess c_0, one replaces f with the linear approximation $\tilde{f}(x) = f(c_0) + f'(c_0)(x - c_0)$ and chooses c_1 as the zero of $\tilde{f}$. Under a certain number of assumption (among which $c^{(0)}$ should be sufficiently close to the true solution), the resulting sequence of root approximations converges quadratically, that is,

$$|c - c_{n+1}| \leq C|c - c_n|^2 \tag{1.1}$$

(in most cases quadratic convergence leads to a doubling of the number of correct digits at each iteration). However, keep in mind that Newton's method require a certain number of assumptions to converge, and these are not alway satisfied. For instance, it cannot work if suddenly $f'(c_n) = 0$ for a certain iterate c_n. In general, there is a whole zoo of so called quasi-Newton methods that converge superlinearly under less stringent requirement and usually only require an approximation of f'. The most famous of such methods is the Broyden-Fletcher-Goldfarb-Shanno algorithm (BFGS).

Newton's method (and quasi-Newton's methods) can be extended straightforwardly to higher-dimensional problems. For instance, let $\mathbf{F} : \mathbb{R}^d \to \mathbb{R}^d$ and let $\mathbf{c}_0 \in \mathbb{R}^d$ be given. Under suitable conditions, the sequence of vectors $\{\mathbf{c}_n\}_{n\in\mathbb{N}}$ defined iteratively by

$$\mathbf{F}(\mathbf{c}_n) + \mathbf{DF}(\mathbf{c}_n)(\mathbf{c}_{n+1} - \mathbf{c}_n) = \mathbf{0},$$

(where $\mathbf{DF}(\mathbf{c}_n)$ denotes the Jacobian matrix of $\mathbf{F}$ evaluated at $\mathbf{c}_n$) converges quadratically to the vector $\mathbf{c}$ that satisfies $\mathbf{F}(\mathbf{c}) = 0$.

Essentially, the convergence of Newton's algorithm relies on the function

$$\mathbf{G}(\mathbf{x}) \coloneqq \mathbf{x} - \mathbf{DF}(\mathbf{x})^{-1}\mathbf{F}(\mathbf{x})$$

being a contraction in a neighborhood of $\mathcal{U}$ of $\mathbf{c}$, that is, $\mathbf{G}(\mathcal{U}) \subset \mathcal{U}$ and there is a constant $L \in [0, 1)$ such that $\|\mathbf{G}(\mathbf{x}) - \mathbf{G}(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\|$ for every $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$. If these conditions are met, the Banach Fixed Point Theorem states the sequence given by $\mathbf{c}_{n+1} = \mathbf{G}(\mathbf{c}_n)$ is guaranteed to converge to the unique fixed point $\mathbf{c}$ of $\mathbf{G}$ (provided that $\mathbf{c}_0 \in \mathcal{U}$). At this point note that if you want to find the zero of a function $\mathbf{F}$ that is itself a contraction, you could simply use a fixed point iteration (algthough the convergence may not be quadratic).

2 Interpolating data

Little data (in 1D): We have a (relatively) small finite number of pairs $(x_k, y_k) \in \mathbb{R}^2$ (function samples) sorted with respect to the component x_k and we want to find a function f that satisfies $f(x_k) = y_k$ (we tacitly assume that the sequence $\{x_k\}_{k=1}^N$ is strictly monotone increasing) so that we can evaluate f at some other point x. Mathematically speaking, this question is not really well posed because, for a finite sequence $\{(x_k, y_k)\}_{k=1}^N$, there are infinitely many such functions f (for instance, infinitely many polynomials of degree N). Nevertheless, we want a meaningful f (whatever that means).
A very popular approach to compute such an f is to use splines. A spline of degree p is a function f

(i) that satisfies $f(x_k) = y_k$,

(ii) whose restriction $f|_{[x_k, x_{k+1}]}$ is a polynomial of degree p,

(iii) that is $(p-1)$-times continuously differentiable on $[x_1, x_N]$.

A piecewise polynomials f can be represented with an $(N-1) \times (p+1)$ matrix, where the ith row contains the coefficients $a_{i,1}, \ldots, a_{i,p+1}$ of the monomial expansion

$$f(x) = a_{i,1}x^p + a_{i,2}x^{p-1} + \cdots + a_{i,p}x + a_{i,p+1} \quad \text{for } x \in (x_i, x_{i+1}) .$$

This implies that a piecewise polynomials has $(N-1)(p+1)$ degrees of freedom ($N-1$ intervals). The regularity condition (iii) provides $p(N-2)$ constraints ($(p-1)$ derivatives and the function itself are continuous at the $(N-2)$ internal nodes), whereas the interpolatory condition gives additional N constraints. Therefore, there remain additional $(N-1)(p+1) - p(N-2) - N = p-1$ degrees of freedom to choose (unless $p = 1$).
If $y_1 = y_k$, a common choice is to go for periodic splines and further requires that

$$f^m(x_1) = f^m(x_N) \quad \text{for } m = 1, \ldots, p-1.$$

Otherwise, when $p = 2\ell - 1$, $\ell \geq 2$, a common choice are natural splines, which further satisfy

$$f^{\ell+m}(x_1) = f^{\ell+m}(x_N) = 0 \quad \text{for } m = 0, 1, \ldots, \ell-2.$$

Big data (in higher-dimension): We have a very large finite number of pairs $(\mathbf{x}_k, y_k) \in \mathbb{R}^{d+1}$ which correspond to (noisy) function samples and we want to find a function f that emulates the behavior of the data. Emulate the data is usually interpreted in the least-squares sense, that is, find f such that

$$\sum_{i=1}^{N} (f(\mathbf{x}_i) - y_i)^2 \tag{2.1}$$

is minimal. Of course, we cannot search among all possible functions, but have to content ourselves with (vector) space of functions X_M of finite dimension M

(M can be very large, but we assume $M \leq N$; and most often we have $M \ll N$). Let $\{b_i\}_{i=1}^M$ be a basis of X_M. Than, any function $f \in X_M$ can be uniquely represented in terms of $\{b_i\}_{i_1}^M$, that is, for every function $f \in X_M$ there is a unique real vector $\mathbf{c} = (c_1, \ldots, c_m)^\top \in \mathbb{R}^M$ such that

$$f(\mathbf{x}) = \sum_{i=1}^{M} c_i b_i(\mathbf{x}) \quad \text{for every } \mathbf{x} \in \mathbb{R}^d \,.$$

Henceforth, we use the notation $f(\mathbf{x}, \mathbf{c})$ for functions f that are in X_M. The key observation here is that f depends linearly on $\mathbf{c}$, as we can write

$$f(\mathbf{x}, \mathbf{c}) = \mathbf{b}(\mathbf{x}) \cdot \mathbf{c} \,, \quad \text{where} \quad \mathbf{b}(\mathbf{x}) = (b_1(\mathbf{x}), \ldots, b_M(\mathbf{x}))^\top.$$

Let $\mathbf{B} \in \mathbb{R}^{N,M}$ be the matrix whose ith row contains $\mathbf{b}(\mathbf{x}_i)^\top$. Then,

$$\sum_{i=1}^{N} (f(\mathbf{x}_i, \mathbf{c}) - y_i)^2 = \|\mathbf{B}\mathbf{c} - \mathbf{y}\|^2 \,, \quad \text{where } \mathbf{y} = (y_1, \ldots, y_N)^\top \,.$$

Therefore, we can rewrite the restriction of (2.1) to X_M as follows.

$$\text{Find } \mathbf{c} \in \mathbb{R}^M \text{ such that } F(\mathbf{c}) \coloneqq \|\mathbf{B}\mathbf{c} - \mathbf{y}\|^2 \text{ is minimal.} \tag{2.2}$$

This is an unconstrained quadratic convex optimization problem and can be solved exactly (provided that $\mathbf{B}$ has maximal rank and neglecting round-off errors) by finding $\mathbf{c}$ such that $\mathbf{D}_{\mathbf{c}} F(\mathbf{c})(\boldsymbol{\delta}\mathbf{c}) = 0$ for every $\boldsymbol{\delta}\mathbf{c} \in \mathbb{R}^d$. Note that

$$\mathbf{D}_{\mathbf{c}} F(\mathbf{c}) = \mathbf{B}^\top \mathbf{B}\mathbf{c} - \mathbf{B}^\top \mathbf{y} \tag{2.3}$$

(use $\|\mathbf{B}\mathbf{c} - \mathbf{y}\|^2 = (\mathbf{B}\mathbf{c} - \mathbf{y}) \cdot (\mathbf{B}\mathbf{c} - \mathbf{y})$ to compute the expansion of $F(\mathbf{c} + \boldsymbol{\delta}\mathbf{c})$). Note that, depending on the choice of the basis functions, the matrix $\mathbf{B}^\top \mathbf{B}$ may be very ill-conditioned. In such an instance, small errors in $\mathbf{y}$ have a high impact on $\mathbf{c}$. When this happens, the problem is said to be ill-posed, and one should consider a regularization term. A common solution is Tychonoff regularization, where (2.2) is replaced with

$$\text{Find } \mathbf{c} \in \mathbb{R}^M \text{ such that } \|\mathbf{B}\mathbf{c} - \mathbf{y}\|^2 + \alpha \|\mathbf{c}\|^2 \text{ is minimal,} \tag{2.4}$$

for an $\alpha > 0$ (and of course, the solution $\mathbf{c}$ will depends on α).

3 Interpolating functions

Let $f : [-1, 1] \to \mathbb{R}$ be a scalar continuous function.

Using polynomials: The Weierstrass Theorem states that f can be approximated arbitrary well by polynomials, that is,

$$\text{for every } \varepsilon > 0 \text{ there is a polynomial } p \text{ such that } \|f - p\| \leq \varepsilon. \tag{3.1}$$

Polynomials are much more handy to deal with compared to generic functions. For instance, they are easy to evaluate, add, multiply, differentiate, integrate, etc... So the question here is the following: how can we construct a good polynomial approximation p of a given function f?
The space $\mathcal{P}_n$ of polynomials (on the interval $[-1, 1]$) of degree $\leq n$ is a vector space of dimension $n + 1$. Let $\{b_i\}_{i=0}^n$ be a basis of $\mathcal{P}_n$. Then, any element $p \in \mathcal{P}_n$ can be written as $p = \sum_{i=0}^n \mu_i b_i$ for certain coefficients $\{\mu_i\}_{i=0}^n$. An idea to find a polynomial $I_n(f)$ that resembles f is to pick $n+1$ interpolation points $\{x_i\}_{i=0}^n$ and solve the linear system

$$\sum_{i=0}^{n} \mu_i b_i(x_j) = f(x_j) \quad \text{for } j = 0, \ldots, n\,. \tag{3.2}$$

How well the resulting interpolatory polynomial $I_n(f)$ approximates the function f (in particular, how $\|I_n(f) - f\|$ behaves as n increases) depends on (i) the regularity of the function f and (ii) the choice of the interpolation points (note that (3.1) does not guarantee that a good approximation can be achieved via interpolation). Although the choice of the basis $\{b_i\}_{i=0}^n$ is irrelevant from a theoretical point of view, it has a huge practical impact on the condition of linear system (3.2).
However, if we only want to evaluate $I_n(f)$, there is no need to go through (3.2), because the barycentric interpolation formula shows that

$$I_n(f)(x) = \frac{\sum_{j=0}^n \frac{\lambda_j f(x_j)}{x - x_j}}{\sum_{j=0}^n \frac{\lambda_j}{x - x_j}}, \quad \text{where} \quad \lambda_j \coloneqq \left(\prod_{k \neq j} (x_j - x_k) \right)^{-1}. \tag{3.3}$$

Believe it or not, the right-hand side of (3.3) is indeed a polynomial and its evaluation is stable. In particular, there is no problem in trying to evaluate $I_n(f)(x_i)$ (despite the seeming division by 0). Note that this does not imply that $I_n(f)$ is a good approximation of f, though.

Using piecewise polynomials: interpolation with piecewise polynomials is what we did in the previous lecture; see **Little data (in 1D)**. Nevertheless, it is worth investing some time to recast it as in (3.2), that is, to figure out what are the basis functions that span the vector space of piecewise polynomials. For simplicity, we restrict to piecewise linear functions, but keep in mind that it

is possible to construct vector spaces of piecewise polynomials of higher degree (just google *Lagrange finite elements* or *B-splines*).
What we want to have is a basis $\{b_i\}_{i=0}^n$ of the space of functions pp whose restriction $pp|_{[x_i,x_i+1]}$ is affine on every interval $[x_i, x_i + 1]$. The answer is simple, just set

$$b_0(x) = \begin{cases} 1 - \frac{x-x_0}{x_1-x_0} & \text{for } x_0 \le x \le x_1\,, \\ 0 & \text{otherwise}\,, \end{cases}$$

$$b_i(x) = \begin{cases} \frac{x-x_{i-1}}{x_i-x_{i-1}} & \text{for } x_{i-1} \le x \le x_i\,, \\ 1 - \frac{x-x_i}{x_{i+1}-x_i} & \text{for } x_i \le x \le x_{i+1}\,, \\ 0 & \text{otherwise}\,, \end{cases} \qquad \text{for } i = 1, \ldots, n-1,$$

$$b_n(x) = \begin{cases} \frac{x-x_{n-1}}{x_n-x_{n-1}} & \text{for } x_{n-1} \le x \le x_n\,, \\ 0 & \text{otherwise}\,. \end{cases}$$

The nice thing about piecewise polynomial interpolation is that it generalize more easily to arbitrary domains in higher dimension. For instance, let $\mathbf{P} \subset \mathbb{R}^2$ be a polygon in 2D and let $f : \mathbf{P} \to \mathbb{R}$. To compute an interpolant of f, we first compute a triangulation (mesh) of $\mathbf{P}$. A mesh (that has m triangles and n nodes) is uniquely described by an $n \times 2$ matrix of nodes coordinates and an $m \times 3$ matrix that describes the connectivity of the mesh (each row contains the vertex indices of an oriented triangle of the mesh). On a mesh we can construct the vector space of bivariate functions that are piecewise affine on each triangle. We can construct a basis of this space associating to each node $\mathbf{x}_i$ of the mesh a function b_i that is itself piecewise affine and satisfies $b_i(\mathbf{x}_j) = \delta_{ij}$. Such functions are called hat functions.

4 Quadrature

Let $f : [-1, 1] \to \mathbb{R}$ be a scalar function. We want to compute $\int_{-1}^{1} f(x)\,\mathrm{d}\boldsymbol{x}$. (note that $\int_a^b f(x)\,\mathrm{d}x = \int_{-1}^{1} f(a + (b-a) * (x+1)/2)(b-a)/2\,\mathrm{d}x$).

Key idea: replace f with an interpolant $I_n(f)$ and integrate $I_n(f)$. The interpolant can be written as $I_n(f) = \sum_{i=0}^{n} \mu_i(f) b_i$. Since the coefficients $\mu_i(f)$ depend linearly on f , we have

$$\int_{-1}^{1} f(x)\,\mathrm{d}x \approx \int_{-1}^{1} I_n(f)(x)\,\mathrm{d}x = \sum_{i=0}^{n} \mu_i \int_{-1}^{1} b_i(x)\,\mathrm{d}x = \sum_{i=0}^{n} w_i f(x_i)\,.$$

where w_i are certain quadrature weights that are determined by the interpolation scheme used. For instance, if we use polynomials, by the barycentric formula we have

$$\int_{-1}^{1} I_n(f)(x)\,\mathrm{d}x = \sum_{i=0}^{n} \left(\int_{-1}^{1} \frac{\frac{\lambda_j}{x-x_i}}{\sum_{j=0}^{n} \frac{\lambda_j}{x-x_j}}\,\mathrm{d}x \right) f(x_i)$$

(note that this is not the most efficient way to compute the quadrature weights).

Composite rules: $I_n(f)$ is a piecewise polynomial interpolant of f on the points $\{x_i\}_{i=0}^{n}$. Let's start with piecewise linear: in each subinterval $[x_i, x_{i+1}]$, $I_n(f)|_{[x_i, x_{i+1}]}$ is an affine function, and

$$\int_{x_i}^{x_{i+1}} I_n(f)(x)\,\mathrm{d}x = (I_n(f)(x_i) + I_n(f)(x_{i+1}))\frac{x_{i+1} - x_i}{2} = (f(x_i) + f(x_{i+1})\frac{x_{i+1} - x_i}{2}\,.$$

Thus, summing up, we obtain the composite trapezium rule:

$$\int_{-1}^{1} f(x)\,\mathrm{d}x \approx \frac{1}{2} \sum_{i=0}^{n-1} (f(x_i) + f(x_{i+1}))(x_{i+1} - x_i)\,.$$

Similarly, we can consider the piecewise quadratic interpolant (using the additional interpolation nodes $(x_i + x_{i+1})/2$). Then,

$$\int_{x_i}^{x_{i+1}} I_n(f)(x)\,\mathrm{d}x = \left(f(x_i) + 4f\left(\frac{x_i + x_{i+1}}{2} \right) + f(x_{i+1}) \right) \frac{x_{i+1} - x_i}{6}\,.$$

Thus, summing up, we obtain the composite Simpson rule:

$$\int_{-1}^{1} f(x)\,\mathrm{d}x \approx \frac{1}{6} \sum_{i=0}^{n-1} \left(f(x_i) + 4f\left(\frac{x_i + x_{i+1}}{2} \right) + f(x_{i+1}) \right) (x_{i+1} - x_i)$$

Note that Simpson's rule requires only one function evaluation more per interval compared to the trapezium rule. Therefore, we can use these two scheme to develop an adaptive quadrature scheme.

Gauss quadrature: A quadrature scheme is identified by the pairs $\{w_i, x_i\}_{i=0}^n$, that is, by $2n+2$ degrees of freedom, and with $2n+2$ degrees of freedom you can devise a scheme that integrates exactly polynomials of degree at most $2n+1$. Gauss quadrature achieve exactly this. The quadrature nodes are the zeros of Legendre polynomials, which are orthogonal in $L^2([-1,1])$. In principle, Gauss quadrature weights and nodes can be computed in $\mathcal{O}(n^2)$ operations using the Golub-Welsh algorithm (and in practice you compute them just once).

Clenshaw-Curtis quadrature: This quadrature scheme is based on Chebishev interpolation. In principle it is exact for polynomials of degree n only, but very often it performs as well as Gauss quadrature. On top of that, it has the advantage that its weights and nodes can be computed in $\mathcal{O}(n \log n)$ time using FFT, and the nodes are nested (from n to $2n$), so that it can be used more efficiently for adaptive strategies. `http://epubs.siam.org/doi/pdf/10.1137/060659831`

2D quadrature: Let $\mathbf{P} \subset \mathbb{R}^2$ be a polygon in 2d and let $f : \mathbf{P} \to \mathbb{R}$. We can compute $\int_{\mathbf{P}} f(\mathbf{x}) \, \mathrm{d}\boldsymbol{x}$ using a triangulation of $\mathbf{P}$. Let $\{\mathbf{K}_i\}_{i=1}^M$ be such a triangulation. Indeed

$$\int_{\mathbf{P}} f(\mathbf{x}) \, \mathrm{d}\boldsymbol{x} = \sum_{i=1}^{M} \int_{\mathbf{K}_i} f(\mathbf{x}) \, \mathrm{d}\boldsymbol{x}$$

The only thing we need is a quadrature scheme on triangles. The simplest one is obtained replacing $f(\mathbf{x})|_{\mathbf{K}_i}$ with the constant function $\mathbf{K}_i \ni \mathbf{x} \mapsto f(\mathbf{x}_{\mathbf{K}_i})$, where $\mathbf{x}_{\mathbf{K}_i}$ is the barycenter of the triangle $\mathbf{K}_i$, and to integrate this constant function exactly. The quadrature scheme reads

$$\int_{\mathbf{K}_i} f(\mathbf{x}) \, \mathrm{d}\boldsymbol{x} \approx f(\mathbf{x}_{\mathbf{K}_i}) \boldsymbol{\Delta}_{\mathbf{K}_i} \, ,$$

where $\boldsymbol{\Delta}_{\mathbf{K}_i}$ denotes the area of the triangle $\mathbf{K}_i$. Note that this approximation integrate exactly both constant and affine functions (it is indeed similar to the 1D midpoint rule).

5 Extrapolation

We have an algorithm $\Pi(h)$ that depends on a parameter $h > 0$ and that approximates a certain value y. In particular, $\lim_{h\to 0} \Pi(h) = y$. However, evaluating $\Pi(h)$ might be very demanding (in particular for small h). The idea of extrapolation is the following:

1. pick a finite monotone decreasing sequence $\{h_0, h_1, \ldots, h_n\}$,
2. evaluate $\{\Pi(h_0), \Pi(h_1), \ldots, \Pi(h_n)\}$,
3. compute the polynomial interpolant p of degree n to $\{(h_i, \Pi(h_i))\}_{i=0}^n$
4. evaluate $p(0)$ to approximated y.

Such a strategy provides a better approximation of y provided that $\Pi(h)$ can be expanded in the series

$$\Pi(h) = y + a_1 h + a_2 h^2 + \cdots + a_n h^n + \mathcal{R}_{n+1}(h) h^{n+1} \,,$$

where the coefficients $a_1, \ldots, a_n$ are independent of h. To see why that works, we use the Aitken-Neville scheme to evaluate $p(0)$.

Aitken-Neville: Let $\{(x_i, y_i)\}_{i=0}^n$ be interpolation points. Let the family of polynomials $P_{ik} \in \mathbb{P}_k$ be given by

$$\begin{aligned} P_{i0}(x) &\coloneqq y_i && i = 0, \ldots, n \\ P_{ik}(x) &\coloneqq \frac{(x_i - x) P_{i+1,k-1}(x) - (x_{i+k} - x) P_{i,k-1}(x)}{x_i - x_{i+k}} && k = 1, \ldots, n,\ i = 0, \ldots, n-k \end{aligned}$$

Then, P_{0n} is the unique interpolatory polynomial of degree n to $\{(x_i, y_i)\}_{i=0}^n$.

In our case, with $x = 0$ and

$$\begin{aligned} P_{00} = \Pi(h_0) &= y + a_1 h_0 + a_2 h_0^2 + a_3 h_0^3 + a_4 h_0^4 + \ldots \,, \\ P_{10} = \Pi(h_1) &= y + a_1 h_1 + a_2 h_1^2 + a_3 h_1^3 + a_4 h_1^4 + \ldots \,, \\ P_{20} = \Pi(h_2) &= y + a_1 h_2 + a_2 h_2^2 + a_3 h_2^3 + a_4 h_2^4 + \ldots \,, \\ P_{30} = \Pi(h_3) &= y + a_1 h_3 + a_2 h_3^2 + a_3 h_3^3 + a_4 h_3^4 + \ldots \,. \end{aligned}$$

we have

$$\begin{aligned} P_{01} &= \frac{h_0 P_{10} - h_1 P_{00}}{h_0 - h_1} = y + a_2 h_0 h_1 \frac{h_1 - h_0}{h_0 - h_1} + a_3 h_0 h_1 \frac{h_1^2 - h_0^2}{h_0 - h_1} + a_4 h_0 h_1 \frac{h_1^3 - h_0^3}{h_0 - h_1} + \ldots \\ P_{11} &= \frac{h_1 P_{20} - h_2 P_{10}}{h_1 - h_2} = y + a_2 h_1 h_2 \frac{h_2 - h_1}{h_1 - h_2} + a_3 h_1 h_2 \frac{h_2^2 - h_1^2}{h_1 - h_2} + a_4 h_1 h_2 \frac{h_2^3 - h_1^3}{h_1 - h_2} + \ldots \\ P_{21} &= \frac{h_2 P_{30} - h_3 P_{20}}{h_2 - h_3} = y + a_2 h_2 h_3 \frac{h_3 - h_2}{h_2 - h_3} + a_3 h_2 h_3 \frac{h_3^2 - h_2^2}{h_2 - h_3} + a_4 h_2 h_3 \frac{h_3^3 - h_2^3}{h_2 - h_3} + \ldots \end{aligned}$$

$$P_{02} = \frac{h_0 P_{11} - h_2 P_{01}}{h_0 - h_2} = y + a_3 h_0 h_1 h_2 \frac{h_2 - h_0}{h_0 - h_2} + a_4 h_0 h_1 h_2 (h_2 + h_1 + h_0) + \dots$$
$$P_{12} = \frac{h_1 P_{21} - h_3 P_{11}}{h_1 - h_3} = y + a_3 h_1 h_2 h_3 \frac{h_3 - h_1}{h_1 - h_3} + a_4 h_1 h_2 h_3 (h_3 + h_2 + h_1) + \dots$$

$$P_{03} = \frac{h_0 P_{12} - h_3 P_{02}}{h_0 - h_3} = y + a_4 h_0 h_1 h_2 h_3 \frac{h_3 - h_0}{h_0 - h_3} + \dots$$

So we see that $|P_{03} - y| = \mathcal{O}(h_0 h_1 h_2 h_3)$.

Quadratic series expansion: What to do if

$$\Pi(h) = y + a_1 h^2 + a_2 h^4 + \dots + a_n h^{2n} + \mathcal{R}_{n+1}(h) h^{n+2} \, ?$$

The same extrapolation works, but you can gain much more by choosing the interpolation points $\{h_0^2, h_1^2, \dots, h_n^2\}$, because setting $\tilde{h} = h^2$ we see that

$$\Pi(\tilde{h}) = y + a_1 \tilde{h} + a_2 \tilde{h}^2 + \dots + a_n \tilde{h}^n + \mathcal{R}_{n+1}(h) \tilde{h}^{n+1} \, .$$

6 Initial Value Problems

We consider the initial value problem

$$\mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}), \quad \mathbf{y}(t_0) = \mathbf{y}_0 \in \mathbb{R}^d. \tag{6.1}$$

By Picard's theorem, we know that this problem has a unique stable solution provided that $\mathbf{f}$ is Lipschitz continuous in an adequate neighborhood of $(t_0, \mathbf{y}_0)$. To prove it, one could consider the map

$$\mathbf{y} \mapsto F(\mathbf{y})(t) \coloneqq \mathbf{y}_0 + \int_{t_0}^{t} \mathbf{f}(\tau, \mathbf{y}(\tau))\, \mathrm{d}\tau,$$

which maps functions that satisfies $\mathbf{y}(t_0) = \mathbf{y}_0$ to other functions. Note that

$$\begin{aligned} \|F(\mathbf{y}) - F(\mathbf{z})\|_{C^0} &\leq \int_{t_0}^{t} \|\mathbf{f}(\tau, \mathbf{y}(\tau)) - \mathbf{f}(\tau, \mathbf{z}(\tau))\|\, \mathrm{d}\tau \\ &\leq \int_{t_0}^{t} L\|\mathbf{y}(\tau) - \mathbf{z}(\tau)\|\, \mathrm{d}\tau \leq L(t - t_0)\|\mathbf{y} - \mathbf{z}\|_{C^0} \end{aligned}$$

and if $L(t - t_0) < 1$, then F looks like a contraction (and in fact it is under a proper description of the domain of definition of F). So, we can construct the solution $\mathbf{y}$ to (6.1) by starting with an initial function $\mathbf{y}^{(0)}$ (that satisfies $\mathbf{y}^{(0)}(t_0) = \mathbf{y}_0$) and construct a sequence of functions by repeating $\mathbf{y}^{(n+1)} = F(\mathbf{y}^{(n)})$ iteratively.

Picard's iteration: This procedure can be mimicked numerically. Consider a collection of finite times $\{t_i\}_{i=0}^{N}$. Then

$$\begin{aligned} \mathbf{y}(t_0) &= \mathbf{y}_0, \\ \mathbf{y}(t_1) &= \mathbf{y}_0 + \int_{t_0}^{t_1} \mathbf{f}(\tau, \mathbf{y}(\tau))\, \mathrm{d}\tau, \\ &\vdots \\ \mathbf{y}(t_N) &= \mathbf{y}_0 + \int_{t_0}^{t_N} \mathbf{f}(\tau, \mathbf{y}(\tau))\, \mathrm{d}\tau. \end{aligned}$$

Replacing $\mathbf{y}(t_i)$ with $\mathbf{y}_i$ and integrals with the composite trapezoidal rule

$$\int_{t_0}^{t_i} \mathbf{f}(\tau, \mathbf{y}(\tau)) \approx h \left(\frac{\mathbf{f}(t_0, \mathbf{y}_0) + \mathbf{f}(t_i, \mathbf{y}(t_i))}{2} + \sum_{j=1}^{i-1} \mathbf{f}(t_j, \mathbf{y}(t_j)) \right)$$

we obtain a system of nonlinear equations for the values $\{\mathbf{y}_i\}_{i=1}^{N}$, and this system can be solved with a fixed point iteration if t_N is not too large.

Runge-Kutta Collocation Scheme: Runge-Kutta methods are described by Butcher tables of the form

$$\begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b}^\top \end{array} .$$

Runge-Kutta collocation schemes are constructed choosing collocation points $c_1, \dots, c_s \in [0,1]$ and setting

$$a_{ij} = \int_0^{c_i} L_j(\tau)\,\mathrm{d}\tau\,, \quad b_i = \int_0^1 L_i(\tau)\,\mathrm{d}\tau\,, \tag{6.2}$$

where $\{L_i\}_{i=1}^s$ are the Lagrange polynomials associated to $c_1, \dots, c_s$, that is,

$$L_i(\tau) \coloneqq \prod_{j \neq i} \frac{\tau - c_j}{c_i - c_j}\,, \quad i = 1, \dots, s\,,$$

and $L_j(c_i) = \delta_{ij}$ for every pair $i, j = 1, \dots, s$.

Collocation polynomial (in 1D): To construct collocation one-step methods, we consider the space $\mathcal{P}_s$ of univariate polynomials of degree s. Note that $\mathcal{P}_s$ has dimension $s+1$. Then, we choose s (pairwise distinct) collocation points $c_1, \dots, c_s \in [0,1]$ and compute the unique polynomial $\tilde{y} \in \mathcal{P}_s$ that satisfies

$$\tilde{y}(0) = y(t) \quad \text{and} \quad \tilde{y}'(c_i h) = f(t + c_i h, \tilde{y}(c_i h))\,, \quad \text{for } i = 1, \dots, s\,. \tag{6.3}$$

Finally, we approximate $y(t+h)$ by evaluating $\tilde{y}(h)$.
This polynomial is a spectral approximation of y (note that (6.3) is a system of s nonlinear equations). Under certain assumptions, spectral approximation exhibits exponential convergence, that is

$$\|y - \tilde{y}\|_{C^0([t,t+h])} = \mathcal{O}(\rho^{-s}) \quad \text{for some } \rho > 1\,.$$

To discretize (6.3), it is convenient to use the representation

$$\tilde{y}'(t) = \mu_{s-1} t^{s-1} + \mu_{s-2} t^{s-2} + \cdots + \mu_0\,.$$

Then, the discrete counterpart of (6.3) is

$$\mathbf{M}\boldsymbol{\mu} = f(t + \mathbf{c}h, \mathbf{Q}\boldsymbol{\mu} + y(t))\,,$$

where $\mathbf{M}$ is a matrix such that $\mathbf{M}\boldsymbol{\mu} = \mathbf{y}'(\mathbf{c}h)$ and $\mathbf{Q}$ is a matrix such that $\mathbf{Q}\boldsymbol{\mu} + y(t) = \mathbf{y}(\mathbf{c}h)$.

7 Boundary Value Problems

We consider the Poisson equation in 1D with Dirichlet boundary conditions

$$u'' = f \quad \text{in } (0,1)\,, \qquad u(0) = a\,, u(1) = b\,, \tag{7.1}$$

where $u : [0,1] \to \mathbb{R}$, $f : [0,1] \to \mathbb{R}$, and $a, b \in \mathbb{R}$. Poisson's equation is the simplest test case of an elliptic boundary value problem (BVP = PDE + boundary conditions) and appears in several simple physical models like electrostatics or gravity. The term elliptic refers to certain properties of the differential operator involved.

Finite differences: The basic idea of finite differences is to approximate u'' with a numerical scheme. For $h > 0$, consider the following Taylor expansion

$$u(x \pm h) = u(x) \pm hu'(x) + \frac{h^2}{2}u''(x) \pm \frac{h^3}{3!}u'''(x) + \frac{h^4}{4!}u''''(x) + \dots\,. \tag{7.2}$$

We can use these Taylor expansion to derive formulas that approximate the derivatives of u. For instance, note that

$$u(x+h) = u(x) + hu'(x) + \mathcal{O}(h^2)\,.$$

Therefore, we can approximate $u'(x)$ with the *forward differentiation* formula

$$u'(x) = \frac{u(x+h) - u(x)}{h} + \mathcal{O}(h) \approx \frac{u(x+h) - u(x)}{h}\,.$$

Alternatively, the expansion

$$u(x-h) = u(x) - hu'(x) + \mathcal{O}(h^2)$$

leads to the *backward differentiation* formula

$$u'(x) = \frac{u(x) - u(x-h)}{h} + \mathcal{O}(h) \approx \frac{u(x) - u(x-h)}{h}\,.$$

Another alternative is the *central difference formula*

$$u'(x) = \frac{u(x+h) - u(x-h)}{2h} + \mathcal{O}(h^2) \approx \frac{u(x+h) - u(x-h)}{2h}\,,$$

which can be derived from the Taylor expansion of $u(x+h) - u(x-h)$. To derive an approximation of u'', we can used the Taylor expansion of $u(x+h) + u(x-h)$, and obtain

$$u''(x) = \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} + \mathcal{O}(h^2) \approx \frac{u(x+h) - 2u(x) + u(x-h)}{h^2}\,.$$

Using this formula, we can derive the following finite difference discretization of (7.6)

$$\frac{u(x+h) - 2u(x) + u(x-h)}{h^2} = f(x) \quad \text{for } x \in (0,1)\,, \quad u(0) = a\,, u(1) = b\,.$$

At this point, we introduce the grid $\{x_i = ih\}_{i=0}^N$ (with $N = 1/h$) and denote by $\{u_i\}_{i=0}^N$ the approximation of $\{u(x_i)\}_{i=0}^N$ obtained by solving the linear system

$$\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} = f(x_i) \quad \text{for } i = 1, \dots, N-1\,, \quad u_0 = a\,, u_N = b\,. \tag{7.3}$$

The matrix that arises from (7.3) is tridiagonal and can be generated quickly in MATLAB using the function `spdiags` (make sure you include the boundary conditions!).

Similar elliptic BVPs
The BVP

$$-u'' + cu = f \quad \text{in } (0,1)\,, \qquad u(0) = a\,, u(1) = b\,, \tag{7.4}$$

with $c \in \mathbb{R}^+$ can be treated in a similar fashion. To solve non nonlinear equations of the form

$$-u'' + g(u) = f \quad \text{in } (0,1)\,, \qquad u(0) = a\,, u(1) = b\,, \tag{7.5}$$

(where $g : \mathbb{R} \to \mathbb{R}$ is a nonlinear function), one can discretize in space to derive a nonlinear system of equations and solve this nonlinear system using Newton's method. The success of this approach and the existence/uniqueness of the solution to (7.5) depend on the function g.
Note that on the lefthand side of (7.4) the terms u'' and u have different sign. This is important to guarantee that the operator $u \mapsto -u'' + u$ has no trivial kernel (and is therefore injective), and follows from the fact that the eigenvalues of the operator $u \mapsto -u''$ (restricted to functions that satisfy $u(0) = u(1) = 0$) are strictly positive. To see this, let λ and u be such that $-u'' = \lambda u$ and $u(0) = u(1) = 0$. Then,

$$\lambda \int_0^1 u^2 \,\mathrm{d}x = \int_0^1 u'' u \,\mathrm{d}x = \int_0^1 (u')^2 \,\mathrm{d}x \geq 0.$$

Therefore, $\lambda \geq 0$. Finally, if $\lambda = 0$, then $u' = 0$ and thus u is constant and equal 0 (becase $u(0) = 0$). Note however that $\lambda = 0$ is possible if the boundary condition changes from $u(0) = u(1) = 0$ to $u'(0) = u'(1) = 0$.

Neumann boundary conditions
We consider the elliptic BVP in 1D with Neumann boundary conditions

$$-u'' + u = f \quad \text{in } (0,1)\,, \qquad u'(0) = a\,, u'(1) = b\,, \tag{7.6}$$

The PDE-part can be discretized in using the finite difference approximation of u''. To determine u_0 and u_N one must discretize the boundary conditions. We have the following options: backward/forward finite differences of appropriate order or ghost cells.

Option 1: Since the centered finite difference approximation of u'' is of second order, one should use backward/forward finite differences of order 2. In this case, the equations $u'(0) = a$ and $u'(1) = b$ become

$$2ha = -3u_0 + 4u_1 - u_2 \quad \text{and} \quad 2hb = u_{N-2} - 4u_{N-1} + 3u_N \,,$$

respectively.
Option 2: The idea is to extend the domain to $(-h, 1+h)$ and to introduce to additional components u_{-1} and u_{N+1}. Then, the boundary conditions are replaced with the central differences

$$2ha = u_1 - u_{-1} \quad \text{and} \quad 2hb = u_{N+1} - u_{N-1} \,,$$

respectively. The components u_{-1} and u_{N+1} are determined by imposing the condition $u'' = 0$ also on the boundary points 0 and 1, that is

$$-\frac{u_1 - 2u_0 + u_{-1}}{h^2} + u_0 = f(0) \quad \text{and} \quad -\frac{u_{N+1} - 2u_N + u_{N-1}}{h^2} + u_N = f(1) \,.$$

Eliminating the components u_{-1} and u_{N+1}, one obtains

$$(2+h^2)u_0 - 2u_1 = -2ha + h^2 f(0) \quad \text{and} \quad (2+h^2)u_N - 2u_{N-1} = -2hb + h^2 f(1) \,.$$