Math 361S Lecture notes
More topics

Jeffrey Wong

April 19, 2019

Topics covered

- (Continuous) minimization
  - Newton's method
  - Steepest descent
  - Variants
- Continuous least squares
  - Orthogonal polynomials
  - Fourier series
- General wrap up

1 Minimization (introduction)

We consider the problem of computing

$$\mathbf{x}^* = \text{argmin}_{\mathbf{x}} f(\mathbf{x})$$

i.e. the minimizer of a function $f : \mathbb{R}^n \to \mathbb{R}$. As with non-linear systems, the general problem is quite difficult due to the potential complexity of the problem; robust algorithms are hard to come by. We will assume here that

- f is smooth (in particular, at least the second derivatives are continuous)
- f has a unique minimum $\mathbf{x}^*$, or at least unique-in-practice in that we do not need to worry about other local minima that are not the right solution.
- f is bounded below (so the solver will never diverge to $-\infty$)

The conditions are guaranteed if f is smooth and **convex**, for instance.

1.1 Some calculus

First, recall that the **Hessian** of f is the matrix H with entries

$$H_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}.$$

Taylor's theorem says that for any direction $\mathbf{v}$ and (small) scalar h,

$$f(\mathbf{x} + h\mathbf{v}) = f(\mathbf{x}) + h(\nabla f(\mathbf{x}))^T \mathbf{v} + \frac{1}{2} h^2 \mathbf{v}^T H(\mathbf{x}) \mathbf{v} + O(h^3). \tag{1}$$

Around a critical point where $\nabla f(\mathbf{x}^*) = 0$, (1) says that

$$f(\mathbf{x}^* + h\mathbf{v}) \approx f(\mathbf{x}^*) + \frac{1}{2} h^2 \mathbf{v}^T H(\mathbf{x}) \mathbf{v}$$

so we must have $\mathbf{v}^T H(\mathbf{x}) \mathbf{v} > 0$ for all directions $\mathbf{v}$ (i.e. H is positive definite).

Theorem (minimum condition): For a smooth function $f : \mathbb{R}^n \to \mathbb{R}$,

$$\mathbf{x}^* \text{ is a local minimum } \iff \nabla f(\mathbf{x}^*) = 0 \text{ and } H(\mathbf{x}^*) \text{ is positive definite .}$$

As a reminder of how to prove (1), define $g(h) = f(\mathbf{x} + h\mathbf{v})$ (a scalar function) and apply Taylor's theorem in one variable, computing

$$g'(h) = \sum_{j=1}^{n} v_j \frac{\partial f}{\partial x_j}(\mathbf{x} + h\mathbf{v}), \quad g''(h) = \sum_{i,j} v_i v_j \frac{\partial^2 f}{\partial x_i \partial x_j}(\mathbf{x} + h\mathbf{v})$$

using the chain rule; then $g(h) = g(0) + hg'(0) + \frac{1}{2} h^2 g''(0) + O(h^3)$.

1.2 A first attempt: Newton

In particular, the minimum $\mathbf{x}^*$ must be a zero for the function $\mathbf{G}(\mathbf{x}) = \nabla f$. Thus, Newton's method can be applied to find a zero of the gradient:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - (D\mathbf{G}(\mathbf{x}_k))^{-1} \mathbf{G}(\mathbf{x}_k).$$

Note that the i-th row of DG is the gradient of $\partial f / \partial x_i$, so

$$(D\mathbf{G})_{ij} = \frac{\partial}{\partial x_j} \frac{\partial f}{\partial x_i} = \frac{\partial^2 f}{\partial x_i \partial x_j} = H_{ij}$$

i.e. $D\mathbf{G}$ is just the Hessian of f. This gives Newton's method for minimization:

Algorithm (Newton's method): To minimize $f(\mathbf{x})$, compute

$$\mathbf{x}_{k+1} = \mathbf{x}_k - (H(\mathbf{x}_k))^{-1}\nabla f(\mathbf{x}_k). \tag{2}$$

where ∇f and H are the gradient and Hessian of f.

This iteration has severe limitations. All information about the minimum has been discarded; really, (2) is just finding a critical point. Without a good initial guess, there is no guarantee the algorithm will converge or make progress at each step.

On the other hand, near the solution, the iteration will converge quite fast. But often a good guess is not available, and so we need a method that is more robust.

2 A brief look at descent methods

The function f provides a natural way to define progress: the value of $f(\mathbf{x}_k)$ should decrease. It would be good to have a method that guarantees

$$f(\mathbf{x}_{k+1}) < f(\mathbf{x}_k).$$

If there is enough decrease per step, it should follow that $\mathbf{x}_k$ converges to a (local) minimum, assuming that f is bounded below. This is the idea of a **descent method**. We consider iterations of the form

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k \tag{3}$$

where $\mathbf{p}_k$ is a **search direction** and $\alpha_k > 0$ is a scalar. At each iteration,

i) Compute a suitable search direction $\mathbf{p}_k$ (f decreases along this direction)

ii) Move a distance α_k in this direction so that f decreases.

These steps are the two key ingredients. Efficiency and robustness are at odds here. Efficiency encourages us to be greedy - take as large a step as possible to minimize the amount of steps required. Robustness encourages caution - take a step that is small enough to guarantee progress, or add safeguards to keep the algorithm in check.

For instance, the Newton direction

$$\mathbf{p}_k = -H_k^{-1}(\mathbf{x}_k)\nabla f(\mathbf{x}_k)$$

is good when close to the minimum, but may not a good direction to take in general - it could lead the iteration astray and increase f at a step (or diverge, or something unpredictable).

2.1 Steepest descent

A bit of calculus comes to the rescue here. Consider f around some point $\mathbf{x}$ that is not a critical point. From Taylor's theorem (1),

$$f(\mathbf{x}+h\mathbf{v}) = f(\mathbf{x}) + h(\nabla f(\mathbf{x}))^T\mathbf{v} + O(h^2) \tag{4}$$

for any direction $\mathbf{v}$. It follows that

$$f \text{ decreases (locally) along } \mathbf{v} \text{ if } \nabla f(\mathbf{x})^T\mathbf{v} < 0.$$

Note that 'along' means in the $+\mathbf{v}$ direction (so $h > 0$). In particular, the rate of decrease is maximized when $\mathbf{v}$ and the gradient are parallel, i.e. when

$$\mathbf{v} = -\nabla f(\mathbf{x})$$

which is the direction of **steepest descent**.

Definition (steepest/gradient descent): The iteration

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{p}_k, \qquad \mathbf{p}_k = -\nabla f(\mathbf{x}_k).$$

where α_k is chosen so that $f(\mathbf{x}_{k+1})$ is less than $f(\mathbf{x}_k)$ by a satisfactory amount.[1]

Observe that from (4) that if α_k is small enough, then

$$f(\mathbf{x}_{k+1}) \approx f(\mathbf{x}) - \alpha_k(\nabla f(\mathbf{x}))^T\nabla f(\mathbf{x}) < 0$$

i.e. the 'step length' α_k can always be chosen to make progress. This makes sure the iteration is well-defined, but we need a good strategy for choosing α_k to be reasonably efficient.

2.2 Choosing the step length

Taylor's theorem, being local, does not help much with finding α_k (besides a guarantee that finding α_k is not an impossible task). There are a number of ways to choose α_k; two of them will be discussed here. At iteration k, we can think of the selection of α_k as a minimization problem in 1d for

$$g(\alpha) = f(\mathbf{x}_k + \alpha\mathbf{p}_k). \tag{5}$$

The more effort we put into this sub-problem at each step, the more progress will be made (at the cost of more computation per step). We must have $g(\alpha_k) < g(0)$ but want to have as much decrease as possible with the right balance of effort.

Approach 1 (backtracking): First, pick a value of α (say, $\alpha = 1$). Then,

$$\texttt{while } f(\mathbf{x}_k + \alpha\mathbf{p}_k) \geq f(\mathbf{x}_k), \quad \alpha \leftarrow \alpha/2.$$

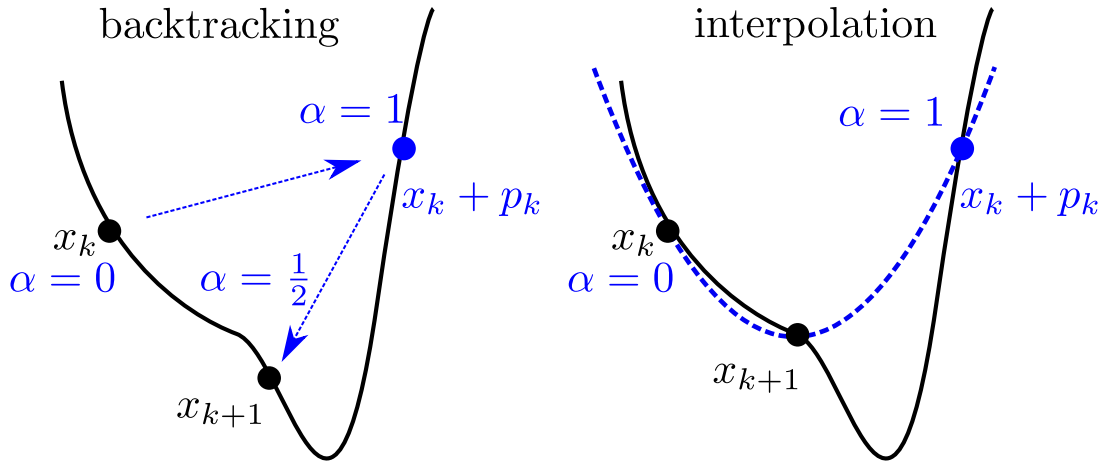


Figure 1: Two schemes for choosing α_k: backtracking (start at $\alpha = 1$, divide by two until satisfied) and interpolation using $g(0), g'(0)$ and $g(1)$.

As observed, this loop will terminate after a finite number of steps, and directly guarantees that $f(\mathbf{x}_{k+1}) < f(\mathbf{x}_k)$. It can be shown that under certain (reasonable) conditions, this simple strategy is enough to guarantee convergence.

Approach 2 (extrapolation): Let us pause for a moment and consider what information we have at $\mathbf{x}_k$. The value of $f(\mathbf{x}_k)$ and the gradient $\nabla f(\mathbf{x}_k)$ are known. Given this, how can we estimate the minimum of $g(\alpha)$? We have

$$g(0) = f(\mathbf{x}_k), \quad g'(0) = (\nabla f(\mathbf{x}_k))^T \mathbf{p}_k.$$

Now evaluate g also at some value of α, say

$$g(1) = f(\mathbf{x}_k + \mathbf{p}_k).$$

This gives us three pieces of data, so we can find a quadratic interpolant

$$g(\alpha) \approx g(0) + c_1\alpha + c_2\alpha^2 := p_2(\alpha)$$

and then take its minimum as the value of α:

$$\alpha_k = \alpha > 0 \text{ that minimizes } p_2.$$

It is not too hard to derive an explicit formula. This works so long as the critical point is a minimum ($c_2 > 0$). If not, we can instead use the more reliable backtracking algorithm.

When the function looks quadratic near the minimum, this extrapolation can perform quite well. However, it is not always true that the quadratic extrapolation gives a good estimate, nor is it always true that the negative gradient is a good search direction. Substantial effort is required to fix these deficiencies.

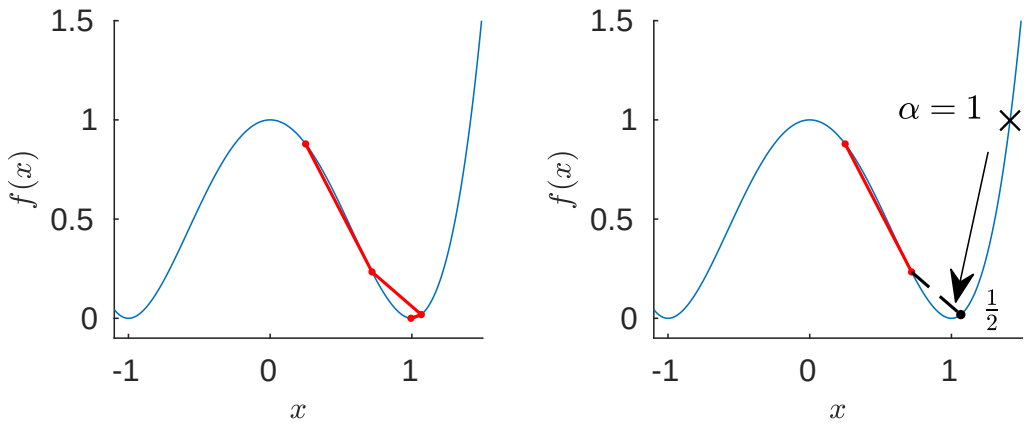


Figure 2: Finding a minimum in 1d using steepest descent. The iterates are shown on the left; calculation of x_2 (with backtracking) on the right, rejecting $\alpha = 1$ and accepting $\alpha = 1/2$.

2.3 In one dimension

The basic idea can be illustrated in one dimension. Consider

$$f(x) = (x^2 - 1)^2$$

and suppose we wish to find one of the minima (at $x = \pm 1$). Newton's method finds one of the critical points of

$$f'(x) = 4x(x^2 - 1)$$

at $x = \pm 1$ or at $x = 0$. The minimum is found if x is not too close to $x = 0$. However, if x_0 is near the peak at $x = 0$, it will instead converge to the local maximum there.

For steepest descent (see Figure 2), the descent 'direction' is a scalar; in the increasing direction if positive and decreasing direction if negative. Using backtracking, the method ensures that the step length is not too large and the sequence of x_k's has $f(x_k)$ decreasing, so x_k must go to a minimum and not the critical point at $x = 0$.

There is no guarantee that the iteration will not hop out of one of the basins and end up at the other minimum (e.g. go from positive to negative). However, it will be the case that once x_k is close enough to a minimum, it will stay close and actually converge.

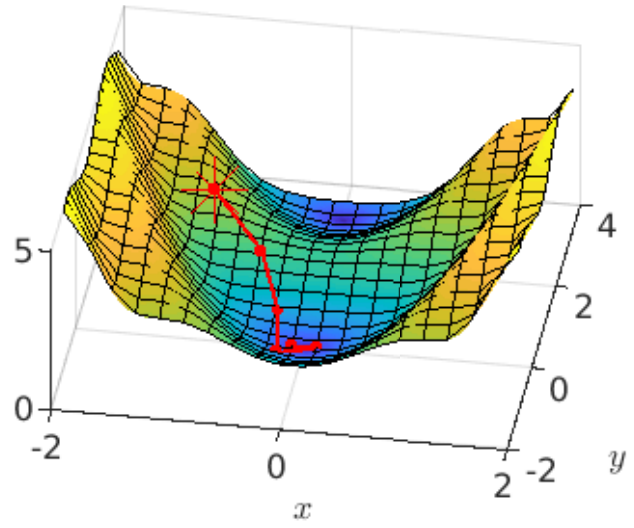


Figure 3: Finding a minimum in 2d using steepest descent (where it works well).

2.4 A straightforward example in 2d

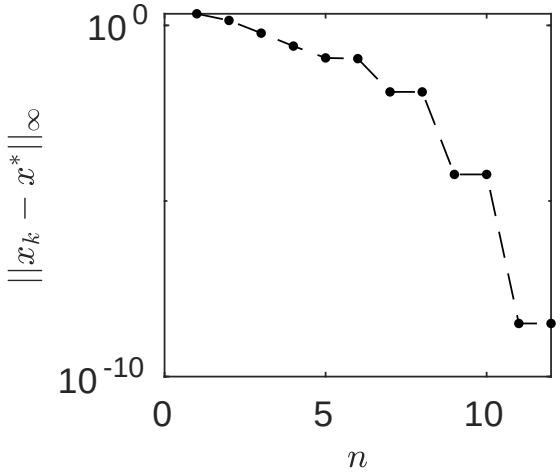
Consider

$$f(x, y) = x^2 + (\sin(y - x^2))^2$$

which has minima at

$$(x, y) = (0, n\pi), \qquad n \in \mathbb{Z}.$$

Steepest descent will converge to one of the minima (depending on the starting point). Nothing surprising occurs here, as shown in Figure 3. The convergence is better than linear here (shown below); typically the error for steepest descent is roughly linear, depending on the scheme used.



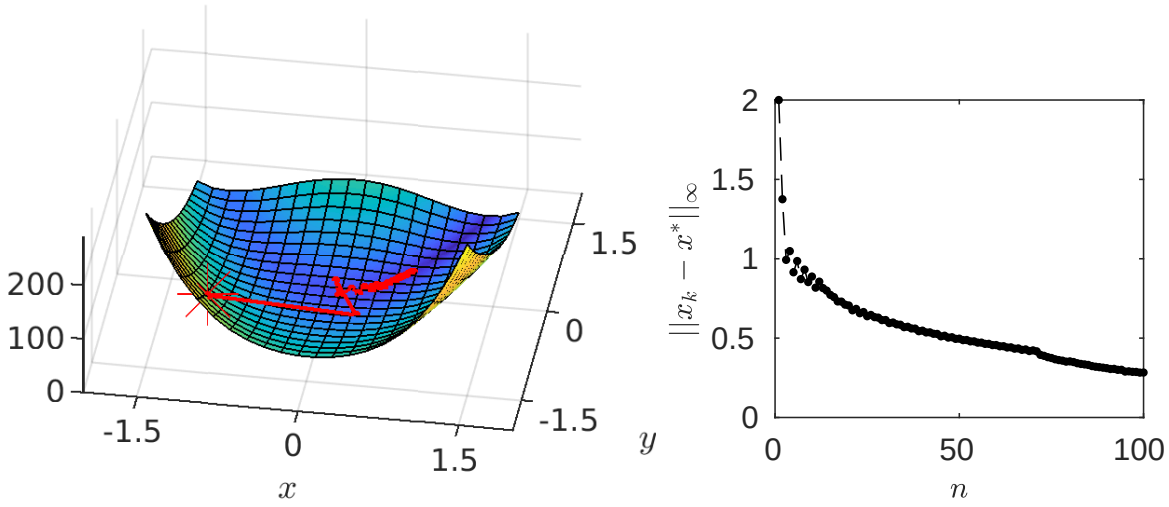


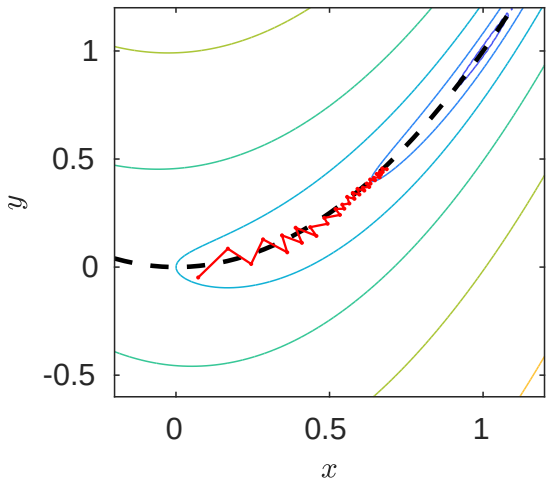
Figure 4: Poor convergence for steepest descent applied to the Rosenbrock function.

2.5 When it does not work well

The Rosenbrock function is

$$f(x, y) = (x - 1)^2 + b(y - x^2)^2$$

which has a unique minimum at $(x, y) = (1, 1)$. There is a valley along the parabola $y \approx x^2$ where the surface is quite shallow. The minimum is sought for $b = 20$ using steepest descent. The iterates are quickly funneled into the valley, but then oscillate back and forth, making little progress along it ($y \approx x^2$). Surface and contour plots are shown in Figure 4 and below.



Fixing the problem: The remedy here is to use a method that picks search directions more judiciously. We need to the sequence to move *along* the valley $y \approx x^2$ (progress there is quite slow for steepest descent). A method that alternates directions normal and parallel to the valley does much better; the (non-linear) **conjugate gradient method** is one approach.

2.6 Extensions

The search direction does not need to be the minus gradient; it can be any direction where progress can be made. More generally, a descent method takes the form

$$\mathbf{p}_k = -B_k^{-1} \nabla f(\mathbf{x}_k) \tag{6}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k \tag{7}$$

where B_k is a matrix. Observe that $\mathbf{p}_k$ is a descent direction if

$$0 < \nabla f(\mathbf{x}_k)^T \mathbf{p}_k = -\nabla f^T B_k^{-1} \nabla f$$

so we should choose B_k to be (symmetric) positive definite. Steepest descent uses

$$B_k = I$$

but any SPD matrix will work.

Notice that Newton's method has this form with B_k equal to the Hessian of f. However, it is **not** a descent method as derived earlier, since α_k is simply taken to be 1. We could imagine instead using a 'damped' Newton method

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k H(\mathbf{x}_k)^{-1} \nabla f(\mathbf{x}_k)$$

where $\alpha_k < 1$ is selected to ensure that progress is made, thus creating a version of Newton's method that falls into the descent method framework. Such methods are called 'Quasi-Newton' methods (e.g. the BFGS method). The nice thing about such methods is that once they get close to the minimum, the method can become Newton-like and have super-linear convergence. Far from the minimum, they will be less Newton-like but will be more robust (guaranteeing a decrease in f).

2.7 Non-linear least squares

Suppose we have a set of data (t_i, y_i) for $j = 0, \cdots, n$ and wish to fit it to a non-linear function. For example,

$$g(t) = x_1 t + e^{x_2 t},$$

where x_1, x_2 are unknown coefficients. The goal is to minimize

$$f(\mathbf{x}) = \frac{1}{2} \sum_{j=0}^{n} |g(x_j; \mathbf{x}) - y_j|^2.$$

Setting $\nabla E = 0$ does not yield a linear system of equations for $\mathbf{c}$ as we saw for linear least squares. Instead, we may define a bector $\mathbf{y}$ and function $\mathbf{G} : \mathbb{R}^m \to \mathbb{R}^n$ such that

$$\mathbf{G}_j(\mathbf{x}) = g(t_j; \mathbf{x}), \quad \mathbf{y}_j = y_j$$

so that the problem is to minimize

$$f(\mathbf{x}) = \frac{1}{2} \|\mathbf{G}(\mathbf{x}) - \mathbf{y}\|^2.$$

This is a minimization problem for which the methods of the previous section can be applied.

Another approach can be derived by solving local linear least squares problems. Suppose we have an approximate minimum $\mathbf{x}_k$. We look for a correction $\mathbf{p}_k$ such that

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k \text{ minimizes } \|\mathbf{G}(\mathbf{x}) - \mathbf{y}\|^2.$$

This is not useful in itself (it's another non-linear least squares problem). But we can look for 'close by' corrections by using a linear approximation

$$\mathbf{G}(\mathbf{x}_k + \mathbf{p}_k) \approx \mathbf{G}(\mathbf{x}_k) + J(\mathbf{x}_k)\mathbf{p}_k + \cdots.$$

where J is the Jacobian of $\mathbf{G}$. Then

$$\min_{\mathbf{x}} \|\mathbf{G}(\mathbf{x}) - \mathbf{y}\|^2 \approx \min_{\mathbf{p}} \|J(\mathbf{x}_k)\mathbf{p} - (\mathbf{b} - \mathbf{G}(\mathbf{x}_k))\|^2.$$

The right hand side is a linear least squares problem; the normal equations are

$$J^T J \mathbf{p}_k = J^T(\mathbf{b} - \mathbf{G}(\mathbf{x}_k)), \qquad J = J(\mathbf{x}_k).$$

We use this to find the correction vector $\mathbf{p}_k$ and then update. This is the **Gauss-Newton method**. It can be shown that this method is close to but not the same as Newton's method. In particular, it is a descent method whereas Newton is not (note that the minimization at each step ensures that progress is made).

See Section 9.2 of the textbook for further discussion.

3 Best approximation of functions

Note: some of the results here depend on Fourier series, which will be reviewed briefly. There is much more insight to be gained with some background in the theory. The main point here is to illustrate the value of **orthogonal bases** for numerical approximation.

3.1 Continuous least squares

Given a function f defined in an interval $[a, b]$ the **continuous least squares** problem is to find a suitable $g(x)$ in some space such that

$$\int_a^b |f(x) - g(x)|^2 \, dx \text{ is minimized.}$$

Previously, we looked at the example of polynomial approximation. Suppose f is defined on $[0, 1]$ and consider approximation by polynomials:

$$p_n(x; \mathbf{c}) = c_0 + c_1 x + \cdots + c_n x^n.$$

We seek to minimize

$$E(\mathbf{c}) = \int_0^1 |f(x) - p_n(x; \mathbf{c})|^2 \, dx.$$

Taking partial derivatives yields the **normal equations**

$$c_j \int_0^1 p_n(x; \mathbf{c}) x^i \, dx = \int_0^1 f(x) x^i \, dx \text{ for } i = 0, 1, \cdots, n.$$

Plugging in the definition of p_n,

$$\sum_{j=0}^{n} c_j \int_0^1 x^{i+j} \, dx = \int_0^1 f(x) x^i \, dx.$$

Defining

$$a_{ij} = \int_0^1 x^{i+j} \, dx = \frac{1}{i + j + 1}, \quad b_i = \int_0^1 f(x) x^i \, dx$$

we get a linear system $A\mathbf{c} = \mathbf{b}$ that can be solved. However, A is the Hilbert matrix, which is ill-conditioned. To get around this numerical difficulty, we need to represent the approximation (here, p_n) in a different way to yield a better A.

Example (linear approximation): Let $f(x) = e^x$. Suppose we wish to find the best 'least-squares' linear approximation to e^x. Consider first the 'naive' representation in terms of 1 and x:

$$p = c_0 + c_1 x.$$

The normal equations lead to the linear system

$$\begin{bmatrix} 1 & 1/2 \\ 1/2 & 1/3 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} = \begin{bmatrix} e - 1 \\ 1 \end{bmatrix}$$

which gives $c_0 = 4e - 10$ and $c_1 = -6e + 18$. Thus

$$p = 4e - 10 + (18 - 6e)x.$$

To get a better conditioned matrix, we should instead write

$$p = c_0 + c_1(x - 1/2).$$

Then $\int_0^1 1 \cdot (x - 1/2)\, dx = 0$ so the equations become

$$\begin{bmatrix} 1 & 0 \\ 0 & 1/12 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} = \begin{bmatrix} e - 1 \\ (3 - e)/2 \end{bmatrix}$$

which is a trivial (diagonal) system and results in

$$p = e - 1 + (18 - 6e)(x - 1/2).$$

The 2×2 Hilbert matrix is not too ill-conditioned, but for larger n the difference matters.

More generally, suppose we have a function f defined on some interval $[a, b]$ and a **basis** $\{\phi_k\}$ (eliding the theory; see details in the last section) to be used for approximation. We look for a function

$$g_n(x) = \sum_{j=0}^{n} c_j \phi_j$$

such that

$$E(\mathbf{c}) = \int_a^b |g_n(x; \mathbf{c}) - f(x)|^2\, dx \text{ is minimized.}$$

As with Fourier series, setting $\nabla E = 0$ gives the **normal equations**

$$\int_a^b f(x)\phi_i\, dx = \sum_{j=0}^{n} c_j \int_a^b \phi_i \phi_j\, dx$$

which have the form

$$A\mathbf{c} = \mathbf{b}, \qquad a_{ij} = \int_a^b \phi_i \phi_j\, dx, \qquad \mathbf{b}_i = \int_a^b f(x)\phi_i\, dx.$$

These equations have a solution so long as A is non-singular.

The equations are easy to solve if A is diagonal. Moreover, the system is best conditioned when A is diagonal, so this is desirable for numerical reasons, not just convenience. This property holds if and only if the ϕ_i's are **orthogonal**:

$$\int_a^b \phi_i(x)\phi_j(x)\, dx = 0 \text{ for } i \neq j.$$

3.2 Least-squares approximation: polynomials

The normal equations are easy to solve if A is diagonal. Moreover, the system is best conditioned when A is diagonal, so this is desirable for numerical reasons, not just convenience. This property holds if and only if the ϕ_i's are **orthogonal**:

$$\int_a^b \phi_i(x)\phi_j(x)\,dx = 0 \text{ for } i \neq j.$$

To construct an orthogonal basis from a non-orthogonal basis $\{\psi_i\}$ we use the **Gram-Schmidt** process. Let

$$\langle f, g\rangle = \int_a^b f(x)g(x)\,dx.$$

Define

$$\begin{aligned}
\phi_0 &= \psi_0,\\
\phi_1 &= \psi_1 - \frac{\langle \psi_1, \phi_0\rangle}{\langle \phi_0, \phi_0\rangle}\phi_0,\\
\phi_j &= \psi_j - \sum_{k=0}^{j-1} \frac{\langle \psi_j, \phi_k\rangle}{\langle \phi_k, \phi_k\rangle}\phi_k
\end{aligned}$$

and so on (subtract from ψ_j the projection of ψ_j onto all previous basis functions to get ϕ_j).

Let us consider the interval $[-1, 1]$ and look for **polynomials**:

$$\phi_i = \text{poly. of deg. } i.$$

Note that $1, x, x^2, \cdots$ does not work. The functions 1 and x are orthogonal but not 1 and x^2 and so on. We use Gram-Schmidt:

$$\phi_2 = x^2 - b_1 x - b_0, \qquad b_0 = \frac{\int_{-1}^1 x^2\,dx}{\int_{-1}^1 dx} = 1/3, \quad b_1 = \frac{\int_{-1}^1 x^2 \cdot x\,dx}{\int_{-1}^1 x \cdot x\,dx} = 0$$

so

$$\phi_2 = x^2 - 1/3.$$

Proceeding in this fashion, we obtain the **Legendre polynomials**. The first few are

$$\begin{aligned}
\phi_0 &= 1\\
\phi_1 &= x\\
\phi_2 &= x^2 - 1/3\\
\phi_3 &= x^3 - 3x/5\\
&\vdots
\end{aligned}$$

To find the best polynomial approximation to $f(x)$ in $[-1, 1]$ of degree n, write

$$g_n(x) = \sum_{j=0}^{n} c_j \phi_j(x).$$

where ϕ_j is the j-th Legendre polynomial. By orthogonality, the coefficients that minimize the least-squares error are

$$c_j = \frac{\int_{-1}^{1} f(x)\phi_j(x)\,dx}{\int_{-1}^{1} \phi_j^2\,dx}.$$

The function g_n is the same polynomial we would get by using $1, x, \cdots, x^n$ as a basis, but the orthogonal basis makes the normal equations trivial to solve, avoiding ill-conditioned matrices like the Hilbert matrix.

Example: Suppose we wish to find the best (least-squares) quadratic approximation to e^x in $[-1, 1]$. Use the Legendre polynomials as a basis:

$$p_2 = c_0 + c_1\phi_1 = c_2\phi_2 = c_0 + c_1 x + c_2(x^2 - 1/3).$$

From the conditions

$$c_j \int_{-1}^{1} \phi_j^2\,dx = \int_{-1}^{1} e^x \phi_j\,dx$$

we obtain

$$c_0 = \frac{\int_{-1}^{1} e^x\,dx}{\int_{-1}^{1} 1\,dx} = (e - 1/e)/2 = \frac{e^2 - 1}{2e},$$

$$c_1 = \frac{\int_{-1}^{1} xe^x\,dx}{\int_{-1}^{1} x^2\,dx} = (2/e)/(2/3) = \frac{3}{e},$$

$$c_2 = \frac{\int_{-1}^{1} (x^2 - 1/3)e^x\,dx}{\int_{-1}^{1} (x^2 - 1/3)^2\,dx} = (2(e^2 - 7)/(3e))/(8/45) = \frac{15(e^2 - 7)}{4e}.$$

Again, note that no linear systems need to be solved; ;problem condition is not a concern.

The process can be generalized to other bases as well, by using **weighted least squares**, where the orthgonality property is

$$\int_a^b w(x)\phi_i(x)\phi_j(x)\,dx = 0 \text{ for } i \neq j.$$

For instance, to approximate a function on $[0, \infty)$ of the form

$$f(x) = e^{-x}(\cdots)$$

we may consider minimizing

$$\int_0^\infty e^{-x} |f(x) - g_n(x)|^2\,dx.$$

When g_n is a polynomial, this leads to the **Laguerre polynomials** that satisfy

$$\int_0^\infty e^{-x}\phi_i\phi_j\,dx = 0 \text{ for } i \neq j.$$

These orthogonal bases have a number of nice properties in various cases due to the rather convenient orthogonality property.

3.3 Fourier series (review)

Periodic functions: A function $f : \mathbb{R} \to \mathbb{R}$ is **periodic** with period T if

$$f(x) = f(x+T), \qquad x \in \mathbb{R}.$$

It suffices to define such a function on any interval of length T.

Fourier series: The **Fourier series** for a 2π-periodic function is

$$f = \frac{a_0}{2} + \sum_{k=1}^{\infty} (a_k \cos kx + b_n \sin kx). \tag{8}$$

The coefficients are given by the formulas

$$(a_k, b_k) = \frac{1}{\pi} \int_{-\pi}^{\pi} (\cos kx, \sin kx)\, dx.$$

Orthogonality: Define the **inner product**

$$\langle f, g \rangle = \int_{-\pi}^{\pi} f(x) g(x)\, dx.$$

Two functions are said to be **orthogonal** (w.r.t this inner product) if $\langle f, g \rangle = 0$.

Fourier basis: Considering the interval $[-\pi, \pi]$ or 2π-periodic functions, let

$$\phi_k = \cos kx, \quad \psi_k = \sin kx.$$

In particular, we have

$$\langle \phi_m, \psi_n \rangle = 0 \text{ for all } m, n$$

$$\langle \phi_m, \phi_n \rangle = \begin{cases} 0 & \text{for } m \neq n \\ \pi & \text{for } m = n \neq 0 \\ 2\pi & \text{for } m = n = 0 \end{cases}$$

$$\langle \psi_m, \psi_n \rangle = \begin{cases} 0 & \text{for } m \neq n \\ \pi & \text{for } m = n \end{cases}$$

This set of functions is a basis for reasonable 2π-periodic functions f in that every such function has a representation (8).

3.4 Least-squares approximation: Fourier series

Let f be a 2π-periodic function and consider it on the interval $[-\pi, \pi]$. We approximate using **trigonometric polynomials**

$$T_n(x) = \frac{a_0}{2} + \sum_{j=1}^{n} (a_j \cos jx + b_n \sin jx).$$

Given n, we seek coefficients a_j and b_j such that

$$E(\mathbf{a}, \mathbf{b}) = \int_{-\pi}^{\pi} |f(x) - T_n(x; \mathbf{a}, \mathbf{b})|^2 \, dx \text{ is minimized.}$$

Taking partial derivatives, we find that

$$0 = \int_{-\pi}^{\pi} (f(x) - T_n(x)) \cos jx \, dx$$
$$0 = \int_{-\pi}^{\pi} (f(x) - T_n(x)) \sin jx \, dx.$$

Now we use the orthogonality property to obtain the simple formula

$$(a_j, b_j) = \frac{1}{\pi} \int_{-\pi}^{\pi} (\cos jx, \sin jx) \, dx.$$

Thus, the best trigonometric polynomial approximation to $f(x)$ in the least squares sense is the first n terms of its Fourier series (8). The point here is that the orthogonality of the basis functions allows the coefficients to be easily computed.

4 Wrap-up

Problems: what tools do we have?

- **Interpolation:** approximate a function using a finite set of data (f, f', $\cdots$)
- **Taylor series:** local approximation $\rightarrow$ methods (linearization, ...)
- **Direct methods for linear systems:** (reduce A to a simpler matrix)
- **Iterative methods** (for $Ax = b$; use convergent sequence)
- **Discretization:** replace 'continuous' calculus with 'discrete' version, e.g.

$$\int f(x) \, dx \approx \sum c_i f(x_i), \qquad f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}.$$

- **Orthogonal bases:** Represent functions as $f = \sum c_i \phi_i$ for orthogonal ϕ's (leads to nice structure for approximations)

What else is there?

- Spectral methods (Fourier-based methods; pseudo-spectral methods)
- More on orthogonal bases (e.g. finite element method)
- Stochastic methods (stochastic gradient descent,...)
- Methods for very large sparse linear systems

General concepts: what do we know?

- **Economize calculations:** The right representation of the problem helps to craft efficient methods and can provide shortcuts. Iterative methods for $Ax = b$ only need $x \to Ax$ and not A (good for large sparse systems). RK methods for $y' = f(t, y)$ avoid computing derivatives of $f(t, y)$. Aitken extrapolation lets one use less iterations to get sufficient accuracy.

- **Problems and algorithms can have different stability:** A problem's **condition** is distinct from the **numerical stability** of the algorithm. Ill-conditioned problems are hard to solve; unstable methods can be replaced by stable ones, sometimes.

  Non-stiff solvers do poorly on stiff equations, but stiff solvers do fine. LU factorization is numerically stable, but ill-conditioning causes problems.

- **We can make precise statements about error:** Expressions for error can be determined even when the actual error is not computable.

  $$\text{Linear/quadratic convergence: } |e_{n+1}| \sim r|e_n|, \qquad |e_{n+1}| \sim |e_n|^2, \cdots$$

  Linear is fast for some methods (integration) and slow for others (root-finding).

  $$\text{Sub-linear cases ('order } p\text{'): } |e_n| \sim \frac{C}{n^p}, \quad E \sim Ch^p, \quad \cdots$$

  Very slow for minimization, root-finding; typical for interpolation-based methods.

- **Error can be estimated (using theory):** e.g. using **asymptotic error series** like

  $$\frac{f(x+h) - f(x-h)}{2h} = f'(x) + c_2h^2 + c_4h^4 + \cdots$$

  and extrapolation. Knowing the structure of the error means it can be manipulated.

4.1 A few practical conclusions

- **The math matters:** Subtle changes can make a big difference. The midpoint formula is an order better than the left-hand rule, but just shifts the points by $h/2$. Symmetry makes a profound difference - but what kind and when it matters depends on the problem. High-degree interpolation can be terrible with equally spaced points but good with the a better distribution (Chebyshev points).

- **Order isn't everything:** A higher-order method is not always 'more accurate'. Convergence 'as $h \to 0$' is not the same as 'better error for practical values of h'. For a stiff ODE, Backwards Euler will perform better than RK45. Integrating a function with an adaptive method (e.g. the adaptive Simpson's rule) will outperform a higher order formula with a fixed step size.

- **Methods/algorithms are building blocks:** The various tools can be combined. Brent's method uses quadratic interpolation plus bisection plus the secant method to find roots. Steepest descent uses 'simpler' minimization tricks to find the optimal step length. A PDE can be reduced to a system of ODEs, which can be solved implicitly, which involves using Newton's method at each step; Newton's method uses a linear system solver at each of its steps. Identify the pieces helps to guide your understanding of algorithms with many moving parts (and implementation!).

- **Be careful, but use what you know:** Algorithms can go wrong, and it is up to you to recognize the signs. With knowledge, you can be cautious but not paranoid - you can make convincing arguments that results are correct, or state exactly why they are wrong (and what parts are right). If Newton's method is not converging quadratically, it could be that the zero is not simple, the function is not smooth, the function is not being computed accurately (spoiled by rounding error?), or the initial guess is wrong. Cases have different symptoms - so you can diagnose the problem with numerical tests.

- **There's always a trade-off:** No algorithm is perfect. Choose the one that suits your needs. Use understanding of the problem to make that decision and be willing to try multiple methods.