PC3236

Computational methods in physics

Jia Xiaodong

May 7, 2021

1 Miscellaneous

1.1 Gram-Schmidt orthogonalization

Define the inner product of a set of polynomials $\{\phi_m(x)\}$ as

$$\int_a^b w(x)\phi_m(x)\phi_n(x)\mathrm{d}x.$$

Suppose now we want to construct a orthonormal set ϕ out of $U = \{1, x, x^2, \dots\}$. We may apply Gram-Schmidt orthogonalization to it. Start with $U_0 = 1$. Normalisation means $\phi_0 = \frac{1}{\sqrt{2}}$. The next element ϕ_1 by imposing orthonormality condition of all existing orthonormal vectors and a linear combination of U_1 and ϕ_0. Orthogonality:

$$\begin{aligned} 0 &= \int_{-1}^1 \phi_0(x)(U_1 + \alpha_{10}\phi_0(x))\mathrm{d}x \\ &= \int_{-1}^1 \frac{1}{\sqrt{2}} x\mathrm{d}x + \alpha_{10}\int_{-1}^1 \phi_0(x)\phi_0(x)\mathrm{d}x \\ &= 0 + \alpha_{10} \end{aligned}$$

Normalisation:

$$\begin{aligned} 1 &= \int_{-1}^1 (cx)^2\mathrm{d}x \\ &= c^2\frac{2}{3} \end{aligned}$$

So $\phi_1(x) = \sqrt{\frac{3}{2}}x$. We can continue with the linear combination $U_2 + \alpha_{21}\phi_1(x) + \alpha_{20}\phi_0(x)$. Orthogonality:

$$\begin{aligned} 0 &= \int_{-1}^1 \phi_0(x)(U_2 + \alpha_{21}\phi_1(x) + \alpha_{20}\phi_0(x))\mathrm{d}x \\ &= \int_{-1}^1 \frac{1}{\sqrt{2}} x^2\mathrm{d}x + \int_{-1}^1 \alpha_{21}\frac{\sqrt{3}}{2}x\mathrm{d}x + \int_{-1}^1 \alpha_{20}\phi_0(x)\phi_0(x) \\ &= \frac{1}{\sqrt{2}}\frac{2}{3} + 0 + \alpha_{20} \end{aligned}$$

$$\begin{aligned} 0 &= \int_{-1}^{1} \phi_1(x)(U_2 + \alpha_{21}\phi_1(x) - \frac{\sqrt{2}}{3}\phi_0(x))\mathrm{d}x \\ &= 0 + \alpha_{21} + 0 \end{aligned}$$

Normalisation:

$$1 = \int_{-1}^{1} \left[c\left(x^2 - \frac{1}{3}\right)\right]^2 \mathrm{d}x$$

We find that $\phi_2(x) = \sqrt{\frac{5}{2}}\frac{3x^2-1}{2}$.

1.2 Random number generators

1.2.1 Linear congruential method

Given a seed X_0,

$$X_{i+1} = aX_i + c \bmod m$$

This generates a uniform distribution. Example values are $a = 7^5$, $c = 0$, and $m = 2^{31} - 1$.

1.2.2 Inverse transform method

Suppose we have a probability density function $f(x)$ of a random variable x. We want to generate a sequence of random number with the same density function?

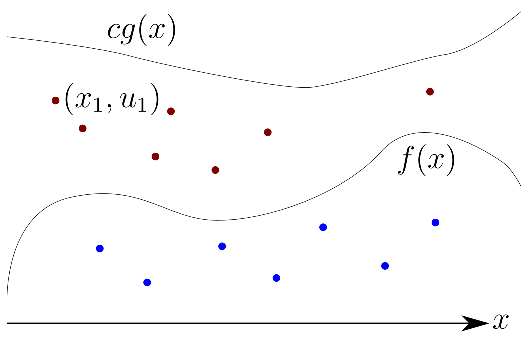
The cumulative distribution function is defined as

$$F_x(x) = \int_{-\infty}^{x} f(x)\mathrm{d}x.$$

Next, generate a sample $u \in U(0,1)$. Then, calculate $x = F_x^{-1}(u)$.

1.2.3 Acceptance rejection method

This is another alternative to the inverse method. Suppose we have a method to simulate a random variable with probability density $g(X)$ such that $f(x) \leq cg(x)$ for all x. Now generate x from g, and generate $u \in U(0,1)$. Accept x if $u \leq \frac{f_X(x)}{cg(x)}$. Otherwise we reject it. This can be visualised as dart throwing:



2 Root finding

In this section we will explore some methods in finding roots of functions numerically.

2.1 Bisection method

The bisection scheme proceeds similarly to binary search. It is most suitable for functions that are continuous in some interval.In a bisection scheme we must first require that the root be bracketed. That is, we must find some interval where there is a change of sign for the function. For example for $f(x) = \cos x - x$, a good bracket can be $[0, \frac{\pi}{2}]$. This can be achieved through graphical means or some other methods.

To bisect the search space, we need to find which side of the interval the search should proceed in. For a given interval $[a, b]$ with midpoint $m = \frac{a+b}{2}$, we calculate the product $f(a)f(m)$. If the product is negative, we proceed recursively in the interval $[a, m]$, otherwise we proceed in $[m, b]$. The reason for this test is that we wish to find the sub-interval that possesses a zero-crossing. The intermediate value theorem guarantees us a root in the sub-interval with the zero-crossing.

It can be seen that since the interval is halved at every iteration, so will the error. The error after n iterations is given by $\varepsilon = \frac{b-a}{2^n}$. Solving for n, we obtain the desired number of iterations:

$$n = \log_2 \left(\frac{|b - a|}{\varepsilon} \right).$$

If the errors at every step are related to the previous step by the following relation

$$\varepsilon_{i+1} = A\varepsilon_i^R,$$

then if $R = 1$, we say the convergence is linear, if $R = 2$, it is quadratic, if $1 < R < 2$ then it is super-linear. For the bisection method, the rate of convergence is linear.

If the function is not continuous or if it is undetermined, then we can set a tolerance value and make sure $|f(m)|$ does not explode during each iteration.

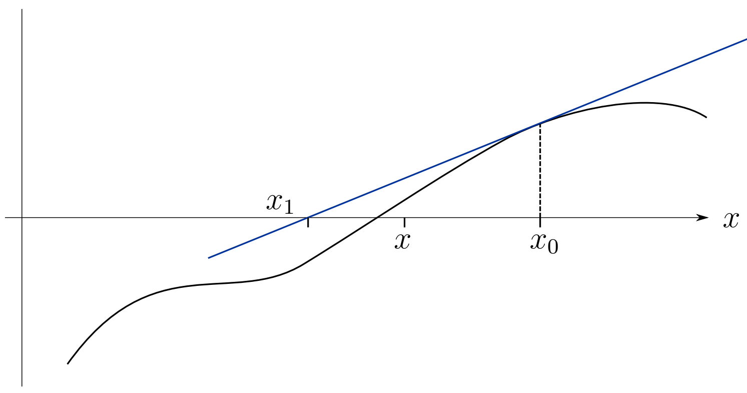
2.2 Newton-Raphson method

The Newton-Raphson method is an iterative scheme to find the root of a function using a trial value. If we have a reasonable guess x_0 of the root (i.e. $x - x_0$ is small), taking a Taylor expansion about $x = x_0$ we have

$$f(x) \approx f(x_0) + (x - x_0)f'(x_0).$$

Since we want to find the value of x where $f(x) = 0$, we have

$$\begin{aligned} 0 &= f(x_0) + (x - x_0)f'(x_0) \\ x &= x_0 - \frac{f(x_0)}{f'(x_0)}. \end{aligned}$$



Continue on the next iteration with x as our new trial value.

The geometrical intuition behind the Newton-Raphson method is as follows. The equation above is actually the tangent line at point x_0. The solution to this linear equation is the x-intercept of the tangent line, which is a better guess at the root of the original function.

The above is also an example of when the Newton-Raphson method fails. The success of the algorithm depends on some factors, such as the distance of our guess from the real root (and hence also the accuracy of the Taylor expansion), the first derivative being non-zero, and so on.

If everything works out and the method converges, we can stop our iteration based on some error tolerance ε, i.e.

$$\left|\frac{f(x)}{f'(x)}\right| < \varepsilon.$$

Of course we also have to check for singularities doing our search, so we also need to check $|f(x)|$ does not become too large.

Now let us determine the rate of convergence. Define the error at the i-th step as $\varepsilon_i = x - x_i$ where x is the actual root and x_i is the trial value. Then

$$\varepsilon_{i+1} = \varepsilon_i + \frac{f(x_i)}{f'(x_i)}.$$

From the Taylor series, keeping the first three terms and with $f(x) = 0$, we get

$$\begin{aligned} f(x) &\approx f(x_i) + (x - x_i)f'(x_i) + \frac{(x-x_i)^2}{2}f''(x_i) \\ f(x_i) &= -\varepsilon_i f'(x_i) - \frac{\varepsilon_i^2}{2}f''(x_i). \end{aligned}$$

Substituting,

$$\begin{aligned} \varepsilon_{i+1} &= \varepsilon_i + \frac{-\varepsilon_i f'(x_i) - \frac{\varepsilon_i^2}{2}f''(x_i)}{f'(x_i)} \\ &= -\frac{\varepsilon_i^2 f''(x_i)}{2f'(x_i)}. \end{aligned}$$

which demonstrates that convergence is quadratic.

In the bisection method, if the root has been bracketed, it always converges, albeit slowly. Newton's method converges rapidly, but it can fail at times. We can combine these two methods into one:

1. Bracket the root, let the interval be $[a, b]$. Let $x_0 = \frac{a+b}{2}$, the midpoint.
2. By computing the product $f(a)f(x_0)$ and comparing to 0, determine if the root lies on the left or right half interval.
3. Using x_0 as the trial value, apply the Newton-Rhapson method to obtain x_1.
   3.1. If x_1 lies in the correct sub-interval, then we accept it and use it for the next Newton-Rhapson iteration.
   3.2. If x_1 does not lie in the correct sub-interval, we reject it and apply the bijection method using the correct bracket.

Algorithm 2.1: Newton-Rhapson method with bisection method.

```
function newton-rhapson($f$):
    $a, b \leftarrow$ bracket values;
    $x \leftarrow (a + b)/2$;
    $dx \leftarrow \infty$;
    while $|dx| >$ tolerance do
        if $f(a)f(x) < 0$ then                       // Tightening brackets
            $b \leftarrow x$;
        else
            $a \leftarrow x$;
        if $|f'(x)| \neq 0$ then                     // Try Newton-Rhapson step
            $dx \leftarrow -f(x)/f'(x)$;
        else
            $dx \leftarrow b - a$;
        $x \leftarrow x + dx$;
        if $(b - x)(x - a) < 0$ then                 // Use bisection if wrong bracket
            $x \leftarrow (a + b)/2$;
            $dx \leftarrow (b - a)/2$;
    return $x$;
```

2.3 Method of false position

Given a function $f(x)$ and a interval $[a, b]$ with a root in it, we can approximate the root by drawing a straight line through a and b and taking the $x - intercept$.

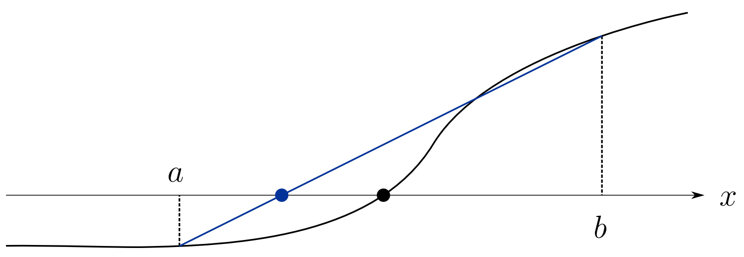
The line can be determined using the Lagrange interpolating polynomial. In our case, where we just want a straight line, we can express the equation of the line as

$$P(x) = A(x - b) + B(x - a)$$

where A and B are constants to be determined. By substituting the appropriate values and solving for the x-intercept, we find that

$$x = \frac{af(b) - bf(a)}{f(b) - f(a)}.$$

After the guess is made, we determine which sub-interval ($[a, x]$ or $[x, b]$) the root lies in, then perform another iteration with the new bounds.



Algorithm 2.2: Method of false position

```
function falseposition($f$):
    $a, b \leftarrow$ initial bounds;
    $xprev \leftarrow \infty$;
    while $\varepsilon >$ Tolerance do
        $x \leftarrow (af(b) - bf(a))/(f(b) - f(a))$;
        $\varepsilon \leftarrow |(xprev - x)/x|$;
        if $f(a)f(x) < 0$ then
            $b \leftarrow x$;
        else
            $x \leftarrow x$;
        $xprev \leftarrow x$;
    return $x$;
```

2.4 Secant Method

The secant method is based on the same principle as the Newton-Rhapson method. However instead of requiring the first derivative of the function, we instead require two initial guesses of the root. Instead of using the exact derivative, we instead approximate it with

$$f'(x_i) \approx \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$

Then substituting this back into the Newton-Rhapson equation, we get the following:

$$x_i = x_{i-1} - f(x_{i-1})\frac{x_{i-1} - x_{i-2}}{f(x_{i-1}) - f(x_{i-2})}.$$

Algorithm 2.3: Secant method

```
function secant($f$):
    $x_1, x_2 \leftarrow$ appropriate guesses;
    while $|dx| >$ tolerance do
        $dx \leftarrow -f(x_2)(x_2 - x_1)/(f(x_2) - f(x_1))$;
        $x_1 \leftarrow x_2$;
        $x_2' \leftarrow x_2$;
        $x_2 \leftarrow x_3$;
        $x_3 \leftarrow x_2' + dx$;
    return $x_3$;
```

If evaluating the function is expensive, we can improve the algorithm by caching the previous function evaluation, so we only evaluate the function once per iteration (except the first iteration). This is an advantage over Newton's method. Although the secant method converges slower than Newton's method (super-linear vs. quadratic), very often its better performance still makes it a competitive choice.

2.5 Brent's method

Brent's method is the method of choice for general one-dimensional root finding problems where we do not have the first derivative. It is similar to the combined Newton and bisection scheme, just that the Newton-Rhapson method is changed into a quadratic interpolation algorithm.

First, we find a bracket for the root as usual, $[x_1, x_2]$. The first point x_3 is obtained through bisection. We then approximate the root using inverse quadratic interpolation. The idea behind inverse quadratic interpolation is to use quadratic interpolation to approximate the inverse of the function in question. The interpolation can be carried out using the Lagrange interpolation polynomial (see example 3.1). Given three points x_1, x_2, and x_3, and their evaluated values $f_1 = f(x_1)$, $f_2 = f(x_2)$, $f_3 = f(x_3)$, the Lagrange formula yields

$$f^{-1}(y) = \frac{(y-f_2)(y-f_3)}{(f_1-f_2)(f_1-f_3)}x_1 + \frac{(y-f_1)(y-f_3)}{(f_2-f_1)(f_2-f_3)}x_2 + \frac{(y-f_1)(y-f_2)}{(f_3-f_1)(f_3-f_2)}x_3.$$

Solving for the root at $y = 0$ gives

$$x = \frac{f_2 f_3}{(f_1-f_2)(f_1-f_3)}x_1 + \frac{f_1 f_3}{(f_2-f_1)(f_2-f_3)}x_2 + \frac{f_1 f_2}{(f_3-f_1)(f_3-f_2)}x_3.$$

Next, if the approximate root x lies in the correct sub-interval, we accept the root and use the endpoints of the sub-interval together with x for our next iteration. Otherwise we apply the bisection method to find a new approximation.

Algorithm 2.4: Brent's method

```
function brent($f$):
    $a, b \leftarrow$ initial bracket;
    $x_1, x_2 \leftarrow a, b$;
    $x_3 \leftarrow (x_1 + x_2)/2$;
    while $\varepsilon >$ tolerance do
        if $f(x_1)f(x_3) < 0$ then                         // Tighten brackets
            $b \leftarrow x_3$;
        else
            $a \leftarrow x_3$;
        if $f(x_1) \neq f(x_2) \neq f(x_3)$ then     // Check to prevent division by 0
            $x \leftarrow$ see formula above;
        else                                         // If div. by 0, force bisection.
            $x \leftarrow x_3 + b$;
        if $(b - x)(x - a) < 0$ then                  // If out of bracket, use bisect
            $x \leftarrow (a + b)/2$;
        $\varepsilon \leftarrow |x - x_3|$;
        if $x < x_3$ then                        // New bounds, maintain $x_1 < x_3 < x_2$
            $x_2 \leftarrow x_3$;
        else
            $x_1 \leftarrow x_3$;
        $x_3 \leftarrow x$;
    return $x$;
```

By caching previous function evaluations, we only need to perform one function call per iteration. Furthermore, it also converges super-linearly and so is quite a good general purpose method.

2.6 Laguerre's method

Laguerre's method provides a way to find all roots to a polynomial. One may use methods like Newton's method, but Laguerre's method is a dedicated technique for polynomials.

Two tools are required for this method. First we need an efficient algorithm for evaluating a polynomial and its derivatives. Second, we need a way to deflate polynomials. This involves calculating $P(x)/(x - r)$ after finding a root r.

2.6.1 Evaluating polynomials and their derivatives

A polynomial of degree n (real or complex) takes the form of

$$P(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$$

The naive way of calculating $P(x)$ incurs $O(n^2)$ multiplications. More efficiently, we evaluate it from right to left to avoid wasting previous multiplications:

$$P(x) = a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + xa_n)))$$

We can thus evaluate $P(x)$ in $O(n)$ evaluations with the following recurrence:

$$\begin{aligned} P_0(x) &= a_n \\ P_i(x) &= a_{n-i} + xP_{i-1}(x). \end{aligned}$$

Now the derivatives may be obtained directly from the above recurrence. It is clear that finding the derivative of P_i involves differentiating P_{i-1}. We get another set of recurrences,

$$\begin{aligned} P_0'(x) &= 0 \\ P_i'(x) &= P_{i-1}(x) + xP_{i-1}'(x) \\ P_0''(x) &= 0 \\ P_i''(x) &= 2P_{i-1}'(x) + xP_{i-1}''(x). \end{aligned}$$

2.6.2 Deflating

After a root r of polynomial $P(x)$ has been determined, we wish to factor it out. Suppose $P(x) = Q(x)(x-r)$, and

$$\begin{aligned} P(x) &= a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 \\ Q(x) &= b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \cdots + b_1 x + b_0. \end{aligned}$$

Then, equating the two,

$$a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 = (x-r)(b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \cdots + b_1 x + b_0).$$

Comparing coefficients give us

$$a_n = b_{n-1} \qquad\qquad a_k = b_{k-1} - rb_k$$

2.6.3 Derivation

Suppose if we know a polynomial has a root at $x = r$ and $n-1$ roots at $x = q$. Then we can write

$$P_n(x) = (x-r)(x-q)^{n-1}.$$

Our objective is to determine r, given the polynomial in its general form $P_n(x) = a_1 x^n + a_2 x^{n-1} + \cdots + a_{n+1}$.

Differentiating once yield

$$\begin{aligned} P_n'(x) &= (x-q)^{n-1} + (n-1)(x-r)(x-q)^{n-2} \\ &= P_n(x)\left(\frac{1}{x-r} + \frac{n-1}{x-q}\right) \\ \frac{P_n'(x)}{P_n(x)} &= \frac{1}{x-r} + \frac{n-1}{x-q}. \end{aligned}$$

Differentiating this expression again yields

$$\frac{P_n''(x)}{P_n(x)} - \left(\frac{P_n'(x)}{P_n(x)}\right)^2 = -\frac{1}{(x-r)^2} - \frac{n-1}{(x-q)^2}.$$

Now, let

$$G = \frac{P_n'(x)}{P_n(x)} \qquad H = G^2 - \frac{P_n''(x)}{P_n(x)}.$$

Solving the simultaneous equations

$$\begin{aligned} G &= \frac{1}{x-r} + \frac{n-1}{x-q} \\ H &= \frac{1}{(x-r)^2} + \frac{n-1}{(x-q)^2}. \end{aligned}$$

for $x-q$, we obtain a quadratic equation in $x-r$. The solution of this equation is Laguerre's formula,

$$x - r = \frac{n}{G \pm \sqrt{(n-1)(nH-G^2)}}.$$

The formula is exact for our special case. However, it turns out that it works well as an iterative method for any polynomial. We will start with a first trial value $x = x_0$. Our next value will then be $r = x_1$, and so on. To ensure convergence, we should pick the denominator with a larger absolute value. This will minimize the difference $x-r$. We stop when $|x-r|$ is below some error threshold.

Algorithm 2.5: Root finding with Laguerre's method.

```
   // $a$ is a list of coefficients of the polynomial $a_1x^n + a_2x^{n-1} + \cdots + a_{n+1}$.
   function lroot($x$):
1      for $coeff_i$ in $a$ do
2          $r \leftarrow$ laguerre($a$, $n$);
3          $roots_i \leftarrow r$;
4          $a \leftarrow$ deflate($a$, $r$);
```

Algorithm 2.6: Deflation algorithm.

```
1  function deflate($a, r$):
2      $b_1 \leftarrow a_1$;
3      for $i$ in $[2..(|a|-1)]$ do
4          $b_i \leftarrow a_i + r * b_{i-1}$;
5      return $b$;
```

Algorithm 2.7: Laguerre's method.

```
function laguerre($a$):                                  // Note that $n = |a| - 1$.
    $x \leftarrow$ random guess; $dx \leftarrow \infty$;
    while $|dx| > tolerance \lor |p| > tolerance$ do
        $(p, dp, ddp) \leftarrow evalpoly(a, x)$;
        $G \leftarrow \frac{dp}{p}$;
        $H \leftarrow G^2 - \frac{ddp}{p}$;
        $s \leftarrow \sqrt{(n-1)(nH - G^2)}$;
        if $|g + f| < |g - f|$ then
            $dx \leftarrow \frac{n}{g-f}$
        else
            $dx \leftarrow \frac{n}{g+f}$
        $x \leftarrow x - dx$;
```

2.7 Systems of equations

So far we have only dealt with solving a single equation. Now let us consider solving an n-dimensional problem using Newton's method.

Given a set of equations

$$\mathbf{f}(\mathbf{x}) = \mathbf{0},$$

Taylor expansion of f_i about $\mathbf{x}$ gives

$$f_i(x_i + \Delta x_i) = f_i(x_i) + \sum_{j=1}^{n} \frac{\partial f_i}{\partial x_j} \Delta x_j + O(\Delta x^2).$$

Dropping the squared terms give

$$\mathbf{f}(\mathbf{x} + \Delta\mathbf{x}) = \mathbf{f}(\mathbf{x}) + \mathbf{J}(\mathbf{x})\Delta\mathbf{x}$$

where $\mathbf{J}(\mathbf{x}) = (j_{ij}) = \frac{\partial f_i}{\partial x_j}$ is the Jacobian.

Hence, if $\mathbf{x}$ is our current approximation of the root of $\mathbf{f}$, we can find the correction term $\Delta\mathbf{x}$ by setting $\mathbf{f}(\mathbf{x} + \Delta\mathbf{x}) = 0$, and then solving

$$\mathbf{J}(\mathbf{x})\Delta\mathbf{x} = -\mathbf{f}(\mathbf{x}).$$

Then we repeat the process using $\mathbf{x} + \Delta\mathbf{x}$ as our new approximation.

To calculate the partial derivatives, we may use a finite difference approximation with a small perturbation δ and unit vector $\mathbf{e}$ in the direction of x_j:

$$\frac{\partial f_i}{\partial x_j} \approx \frac{f_i(\mathbf{x} + \delta\mathbf{e}_j) - f_i(\mathbf{x})}{\delta}.$$

Algorithm 2.8: Solving systems of equations using Newton's method.

```
function simult(f, x):      // Input list of functions f and parameters x.
    $\mathbf{dx} \leftarrow [\infty, \ldots, \infty]$;
    while $\|dx\| >$ tolerance do
        $\mathbf{J} \leftarrow$ jacobian(f, x);
        $dx \leftarrow \mathbf{J}^{-1}(-\mathbf{f}(\mathbf{x}))$;
        $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{dx}$;
```

Algorithm 2.9: Finding the Jacobian using the finite difference approximation.

```
function jacobian(f, x):
    $\delta \leftarrow$ small perturbation;
    $\mathbf{f}_0 \leftarrow \mathbf{f}(\mathbf{x})$;
    for $x_j \in \mathbf{x}$ do
        $temp \leftarrow x_j$;
        $x_j \leftarrow x_j + \delta$;
        $\mathbf{f}_1 \leftarrow \mathbf{f}(\mathbf{x})$;
        $x_j \leftarrow temp$;
        $J_{ij} \leftarrow \left[\frac{\mathbf{f}_1 - \mathbf{f}_0}{\delta}\right]_i$;
```

3 Interpolation and extrapolation

In interpolation we are given a set of points in a region and are asked to estimate an unknown point in the region. This is used for many other numerical techniques such as integration and differentiation.

Interpolation however is usually unsuitable for predicting points outside the given data points since they do not use information about trends in the data. That is where extrapolation comes in.

3.1 Lagrange interpolation

Given a function $f(x)$, consider two points (x_1, y_1) and (x_2, y_2). Taylor's series gives

$$f(x_i) = f(x) + (x_i - x)f'(x) + \cdots$$

Approximating $f(x)$ by an unknown function $p(x)$ this can be made exact

$$f(x_i) = p(x) + (x_i - x)p'(x).$$

Solving for p, we have

$$\begin{aligned} p(x) &= \frac{x - x_2}{x_1 - x_2} f(x_1) + \frac{x - x_1}{x_2 - x_1} f(x_2) \\ &= A(x - x_2) + B(x - x_1). \end{aligned}$$

This is a straight line joining the two points, and is known as the Lagrange interpolating polynomial. It is easy to calculate the constants by plugging in $x = x_1$ and $x = x_2$.

If we were to add another point, we can do the same

$$f(x_i) = p(x) + (x_i - x)p'(x) + \frac{(x_i - x)^2}{2}p''(x)$$

which gives a system of equation which when solved gives

$$\begin{aligned} p(x) &= \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)}f(x_1) + \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)}f(x_2) + \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)}f(x_3) \\ &= A(x - x_2)(x - x_3) + B(x - x_1)(x - x_3) + C(x - x_1)(x - x_2). \end{aligned}$$

Following this trend, for four unknowns we have

$$\begin{aligned} p(x) = &\frac{(x - x_2)(x - x_3)(x - x_4)}{(x_1 - x_2)(x_1 - x_3)(x - x_4)}f(x_1) + \frac{(x - x_1)(x - x_3)(x - x_4)}{(x_2 - x_1)(x_2 - x_3)(x_2 - x_4)}f(x_2) \\ &+ \frac{(x - x_1)(x - x_2)(x - x_4)}{(x_3 - x_1)(x_3 - x_2)(x_3 - x_4)}f(x_3) + \frac{(x - x_1)(x - x_2)(x - x_3)}{(x_4 - x_1)(x_4 - x_2)(x_4 - x_3)}f(x_4) \end{aligned}$$

Example 3.1. In Brent's method we perform the inverse quadratic interpolation. Given (x_1, y_1), (x_2, y_2), (x_3, y_3), we plot x against y, then the inverse quadratic polynomial can be written as

$$x = \frac{(y - y_2)(y - y_3)}{(y_1 - y_2)(y_1 - y_3)}x_1 + \frac{(y - y_1)(y - y_3)}{(y_2 - y_1)(y_2 - y_3)}x_2 + \frac{(y - y_1)(y - y_2)}{(y_3 - y_1)(y_3 - y_2)}x_3.$$

$\Diamond$

In general the Lagrange interpolating polynomial of order $n - 1$ is given by

$$p_{n-1}(x) = \sum_{j=1}^{n} l_{j,n}(x)f(x_j)$$

where the coefficients

$$l_{j,n}(x) = \prod_{\substack{k=1 \\ k\neq j}}^{n} \frac{x - x_k}{x_j - x_k}.$$

Note that sometimes increasing the number of polynomials can actually increase the error. Sometimes a kind of oscillation errors may arise. This can be mitigated using *Chebychev nodes* or just by simply subdividing the region of interest to avoid high order polynomials.

3.2 Newton interpolation

Although Lagrange's method is simple, it is computationally inefficient. A more efficient method is Newton's method, where the interpolating polynomial is written as

$$p_{n-1}(x) = a_1 + (x - x_1)a_2 + (x - x_1)(x - x_2)a_3 + \dots$$

Example 3.2. Consider the case $n = 4$. Then

$$\begin{aligned}p_3(x) &= a_1 + (x - x_1)a_2 + (x - x_1)(x - x_2)a_3 + (x - x_1)(x - x_2)(x - x_3)a_4 \\ &= a_1 + (x - x_1)(a_2 + (x - x_2)[a_3 + (x - x_3)a_4])\end{aligned}$$

This can be evaluated with the recurrence

$$\begin{aligned}p_0(x) &= a_4 \\ p_1(x) &= a_3 + (x - x_3)p_0(x) \\ p_2(x) &= a_2 + (x - x_2)p_1(x) \\ p_3(x) &= a_1 + (x - x_1)p_2(x)\end{aligned}$$

$\Diamond$

From the example, for arbitrary n we have

$$\begin{aligned}p_0(x) &= a_n \\ p_k(x) &= a_{n-k} + (x - x_{n-k})p_{k-1}(x).\end{aligned}$$

The coefficients of p_{n-1} are determined by forcing the polynomial to pass through each data point we have, in other words, solving the following set of equations

$$\begin{aligned}y_1 &= a_1 \\ y_2 &= a_1 + (x_2 - x_1)a_2 \\ y_3 &= a_1 + (x_3 - x_1)a_2 + (x_3 - x_1)(x_3 - x_2)a_3 \\ &\vdots\end{aligned}$$

Define

$$\begin{aligned}\nabla y_i &= \frac{y_i - y_1}{x_i - x_1} && i = 2, 3, \ldots \\ \nabla^2 y_i &= \frac{\nabla y_i - \nabla y_2}{x_i - x_2} && i = 3, 4, \ldots \\ &\vdots \\ \nabla^{n-1} y_i &= \frac{\nabla^{n-2} y_i - \nabla^{n-2} y_{i-1}}{x_i - x_{i-1}}\end{aligned}$$

When solving, we see that first

$$\begin{aligned}a_2 &= \frac{y_2 - y_1}{x_2 - x_1} \\ &= \nabla y_2 \\ a_3 &= \frac{y_3 - y_1 - (x_3 - x_1)\nabla y_2}{(x_3 - x_1)(x_3 - x_2)} \\ &= \frac{\nabla y_3 - \nabla y_2}{x_3 - x - 2} \\ &= \nabla^2 y_3 \\ &\vdots \\ a_n &= \nabla^{n-1} y_n\end{aligned}$$

This form makes the algorithm very suitable for optimization through memoization.

Example 3.3. Fit a polynomial onto the given data points

x	1	-1	3	2	4	-2
y	-2	-14	18	1	61	-47

.

We tabulate our results as follows:

i	x	y	Δ	Δ^2	Δ^3	Δ^4	Δ^5
1	1	-2					
2	-1	-24	6				
3	3	18	10	1			
4	2	1	3	-1	2		
5	4	61	21	3	2	0	
6	-2	-47	15	-9	2	0	0

Newton's interpolating polynomial is written as

$$y = -2 + (x-1)6 + (x-1)(x+1) + (x-1)(x+1)(x-3)2.$$

$\Diamond$

3.3 Hermite interpolation

In the case of the Hermite interpolating polynomials, we also assume the availability of first derivative values at the data points.

Example 3.4. We desire the interpolation polynomial

$$p(x) = ax^3 + bx^3 + cx + d$$

given that $p^{(i)}(x_1) = f(x_1)$ and $p^{(i)}(x_2) = f(x_2)$ where $i = 0, 1$ (zero and first derivatives). Solving for the coefficients we have the Hermite cubic interpolating polynomial:

$$\begin{aligned} p(x) = &\frac{(1 - 2\frac{x-x_1}{x_1-x_2})(x-x_2)^2}{(x_1-x_2)^2} f(x_1) + \frac{(1 - 2\frac{x-x_2}{x_2-x_1})(x-x_1)^2}{(x_1-x_2)^2} f(x_2) \\ &+ \frac{(x-x_1)(x-x_2)^2}{(x_1-x_2)^2} f'(x_1) + \frac{(x-x_2)(x-x_1)^2}{(x_1-x_2)^2} f'(x_2) \end{aligned}$$

$\Diamond$

The general Hermite interpolating polynomial is

$$p(x) = \sum_{j=1}^{n} h_{j,n}(x) f(x_j) + \sum_{j=1}^{n} \bar{h}_{j,n}(x) f'(x_j).$$

We want to obtain an expression for h and $\bar{h}$. We can start from the relations $p(x_i) = f(x_i)$. This alone tells us

$$h_{j,n}(x_i) = \delta_{ij} \qquad\qquad \bar{h}_{j,n}(x_i) = 0.$$

Next, differentiating we have

$$p'(x) = \sum_{j=1}^{n} h'_{j,n}(x) f(x_j) + \sum_{j=1}^{n} \bar{h}'_{j,n}(x) f'(x_j).$$

and using the relation $p'(x_i) = f'(x_i)$, we obtain

$$h'_{j,n}(x_i) = 0 \qquad \bar{h}'_{j,n}(x_i) = \delta_{ij}$$

We make the following hypothesis

$$\begin{aligned} h_{j,n}(x_i) &= (a_j x + b_j) l^2_{j,n}(x) & \bar{h}_{j,n}(x_i) &= (c_j x + d_j) l^2_{j,n}(x) \\ h'_{j,n}(x_i) &= (a_j x + b_j) 2 l_{j,n}(x) l'_{j,n}(x) + a_j l^2_{j,n}(x) & \bar{h}'_{j,n}(x_i) &= (c_j x + d_j) 2 l_{j,n}(x) l'_{j,n}(x) + c_j l^2_{j,n}(x) \end{aligned}$$

where $l_{j,n}$ is the same as in Lagrange interpolation, but reproduced here:

$$l_{j,n}(x) = \prod_{\substack{k=1 \\ k\neq j}}^{n} \frac{x - x_k}{x_j - x_k}.$$

Together with our observations, we are able to solve and obtain the following

$$\begin{aligned} a_j x_j + b_j &= 1 & c_j x_j + d_j &= 0 \\ a_j + 2 l'_{j,n}(x_j) &= 0 & c_j &= 1. \end{aligned}$$

Thus, substituting and simplifying we finally get

$$\begin{aligned} h_{j,n}(x) &= (1 - 2(x - x_j) l'_{j,n}(x_j)) l^2_{j,n}(x) \\ \bar{h}_{j,n}(x) &= (x - x_j) l^2_{j,n}(x) \end{aligned}$$

3.4 Cubic splines

Splines are piecewise polynomials in which the component polynomials satisfy some continuity conditions.

Suppose we have a function and we have n samples. Between every two points x_j and x_{j+1} we can define a cubic polynomial as

$$p(x) = a_j (x - x_j)^3 + b_j (x - x_j)^2 + c_j (x - x_j) + d_j.$$

We require the approximation to be exact at $x = x_j$ and $x = x_{j+1}$, which means

$$\begin{gathered} p(x_j) = d_j \\ p(x_{j+1}) = a_j (x_{j+1} - x_j)^3 + b_j (x_{j+1} - x_j)^2 + c_j (x_{j+1} - x_j) + p(x_j) \end{gathered}$$

We want the final polynomial to be differentiable, so the pieces have to be twice differentiable to ensure a continuous derivative so they can be joined without kinks. The derivatives are given by

$$\begin{aligned} p'(x) &= 3 a_j (x - x_j)^2 + 2 b_j (x - x_j) + c_j \\ p''(x) &= 6 a_j (x - x_j) + 2 b_j. \end{aligned}$$

For the second derivative, we have

$$\begin{aligned} p''(x_j) &= 2b_j \\ p''(x_{j+1}) &= 6a_j h_j + 2b_j. \end{aligned}$$

We can now solve to get

$$\begin{aligned} d_j &= p(x_j) \\ b_j &= \frac{p''(x_j)}{2} \\ a_j &= \frac{1}{6}\frac{p''(x_{j+1}) - p''(x_j)}{x_{j+1} - x_j} \\ c_j &= \frac{p(x_{j+1}) - p(x_j)}{x_{j+1} - x_j} - \frac{(x_{j+1} - x_j)p''(x_{j+1}) + 2(x_{j+1} - x_j)p''(x_j)}{6}. \end{aligned}$$

Now we impose the continuity of first derivatives. In other words,

$$p_j(x_j) = c_j = p_{j-1}(x_j) = 3a_{j-1}(x_j - x_{j-1})^2 + 2b_{j-1}(x_j - x_{j-1}) + c_{j-1}$$

Solving this very large equation (skipped), we obtain

$$\begin{aligned} &(x_{j+1} - x_j)p''(x_{j+1}) + 2p''(x_j)(x_{j+1} - x_{j-1}) + (x_j - x_{j-1})p''(x_{j-1}) \\ &\quad = 6\left(\frac{p(x_{j+1}) - p(x_j)}{x_{j+1} - x_j} - \frac{p(x_j) - p(x_{j-1})}{x_j - x_{j-1}}\right). \end{aligned}$$

This is valid for $j = 2, \ldots, n-1$. We need two more equations to fully solve for the p'''s. The additional equations come from specifying the derivatives at the endpoints x_1 and x_n. This is also known as a *clamped* spline. Substituting these into our expression for $p'(x)$ gives

$$\begin{aligned} (x_2 - x_1)(2p''(x_1) + p''(x_2)) &= 6\frac{p(x_2) - p(x_1)}{x_2 - x_1} - 6p'(x_1) \\ (x_n - x_{n-1})(p''(x_{n-1}) + 2p''(x_n)) &= -6\frac{p(x_n) - p(x_{n-1})}{x_n - x_{n-1}} + 6p'(x_n) \end{aligned}$$

If the first derivatives at the endpoints are not known, we can set the second derivatives to zero at the endpoints. This is known as a *natural* spline. We can express all these as a simple matrix equation

$$\begin{pmatrix} 2(x_2-x_1) & x_2-x_1 & & & & & \\ x_2-x_1 & 2(x_3-x_1) & x_3-x_2 & & & & \\ & x_3-x_2 & 2(x_4-x_2) & x_4-x_3 & & & \\ & & & \ddots & & & \\ & & & & x_{n-1}-x_{n-2} & 2(x_n-x_{n-2}) & x_n-x_{n-1} \\ & & & & & x_n-x_{n-1} & 2(x_n-x_{n-1}) \end{pmatrix} \begin{pmatrix} p''(x_1) \\ p''(x_2) \\ p''(x_3) \\ \vdots \\ p''(x_{n-1}) \\ p''(x_n) \end{pmatrix}$$

$$= \begin{pmatrix} 6\frac{p(x_2)-p(x_1)}{x_2-x_1} - 6p'(x_1) \\ 6\frac{p(x_3)-p(x_2)}{x_3-x_2} - 6\frac{p(x_2)-p(x_1)}{x_2-x_1} \\ 6\frac{p(x_4)-p(x_3)}{x_4-x_3} - 6\frac{p(x_3)-p(x_2)}{x_3-x_2} \\ \vdots \\ 6\frac{p(x_n)-p(x_{n-1})}{x_n-x_{n-1}} - 6\frac{p(x_{n-1})-p(x_{n-2})}{x_n-x_{n-1}} \\ -6\frac{p(x_n)-p(x_{n-1})}{x_n-x_{n-1}} - 6p'(x_n) \end{pmatrix}$$

For the natural spline, we have

$$\begin{pmatrix} 1 & & & & & \\ 2(x_3-x_1) & x_3-x_2 & & & & \\ x_3-x_2 & 2(x_4-x_2) & x_4-x_3 & & & \\ & & \ddots & & & \\ & & & x_{n-1}-x_{n-2} & 2(x_n-x_{n-2}) & \\ & & & & & 1 \end{pmatrix} \begin{pmatrix} p''(x_1) \\ p''(x_2) \\ p''(x_3) \\ \vdots \\ p''(x_{n-1}) \\ p''(x_n) \end{pmatrix}$$

$$= \begin{pmatrix} 0 \\ 6\frac{p(x_3)-p(x_2)}{x_3-x_2} - 6\frac{p(x_2)-p(x_1)}{x_2-x_1} \\ 6\frac{p(x_4)-p(x_3)}{x_4-x_3} - 6\frac{p(x_3)-p(x_2)}{x_3-x_2} \\ \vdots \\ 6\frac{p(x_n)-p(x_{n-1})}{x_n-x_{n-1}} - 6\frac{p(x_{n-1})-p(x_{n-2})}{x_n-x_{n-1}} \\ 0 \end{pmatrix}$$

3.5 Padé approximation

We use rational functions to approximate a function $f(x)$ over a small portion of its domain. A rational function is a ratio of two polynomials $R(x) = \frac{P(x)}{Q(x)}$. To make the approximation unique, We ask that $Q(0) = 1$, so the constant term for Q be set to 1. In other words define

$$R_{n,m}(x) = \frac{P_n(x)}{Q_m(x)} = \frac{\sum_{i=0}^{n} p_i x^i}{1 + \sum_{j=1}^{m} q_j x^j}$$

We require that $f(x)$ and $R_{n,m}(x)$ and their $n+m$ derivatives agree at x_0. This gives rise to $n+m+1$ equations that allow us to solve for the coefficients. Consider the Taylor expansion of $f(x) - R_{n,m}(x)$:

$$\begin{aligned} f(x) - R_{n,m}(x) &= f(x_0) + (x-x_0)f'(x_0) + \cdots + c_1(x-x_0)^{n+m+1} + \ldots \\ &\qquad - [R_{n,m}(x_0) + (x-x_0)R'_{n,m}(x_0) + \cdots + c_2(x-x_0)^{n+m+1} + \ldots] \\ &= (c_1 - c_2)(x-x_0)^{n+m+1} + \ldots \end{aligned}$$

where the second step arises due to our condition of $R^{(k)}_{n,m}(x_0) = f^{(k)}(x_0)$. So the lowest power is $n+m+1$. We can write

$$\begin{aligned} f(x) - R_{n,m}(x) &= \frac{f(x)Q_m(x) - P_n(x)}{Q_m(x)} \\ &= \frac{\sum_{i=0}^{\infty} a_i x^i \sum_{j=0}^{m} q_j x^j - \sum_{i=0}^{n} p_i x^i}{Q_m(x)} \end{aligned}$$

We know the numerator has no terms of of decree lower than $n-m$ inclusive. Hence,

$$\sum_{i=0}^{n+m} a_i q_{k-i} = p_k \qquad\qquad k = 0, 1, \ldots n+m.$$

This gives a system of equations (note that $q_0 = 1$ and $p_0 = a_0$).

Example 3.5. Let $n = 3$ and $m = 2$. The function in question is $f(x) = e^{-x} = \sum_{k=0}^{\infty} \frac{(-1)^k}{k!} x^k$. The first few coefficients are

$$a_0 = 1 \qquad a_1 = -1 \qquad a_2 = \frac{1}{2} \qquad a_3 = -\frac{1}{6} \qquad a_4 = \frac{1}{24} \qquad a_5 = -\frac{1}{120}$$

Then, the coefficients of the Padé approximant can be determined by solving the following matrix equation:

$$\begin{pmatrix} a_0 & 0 & -1 & 0 & 0 \\ a_1 & a_0 & 0 & -1 & 0 \\ a_2 & a_1 & 0 & 0 & -1 \\ a_3 & a_2 & 0 & 0 & 0 \\ a_4 & a_3 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} q_1 \\ q_2 \\ p_1 \\ p_2 \\ p_3 \end{pmatrix} = - \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix}.$$

This gives us

$$R_{3,2}(x) = \frac{1 - \frac{3}{5}x + \frac{3}{20}x^2 - \frac{1}{60}x^3}{1 + \frac{2}{5}x + \frac{1}{20}x^2}.$$

◊

3.6 Approximating derivatives

From Taylor's expansion

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(x) + \cdots$$

we solve for $f'(x)$

$$\begin{aligned} f'(x) &= \frac{1}{h}\left[f(x+h) - f(x) - \frac{h^2}{2!}f''(x) + \cdots\right] \\ &= \frac{f(x+h) - f(x)}{h} + O(h) \end{aligned}$$

This expression for $f'(x)$ is known as the *forward difference expression.* Starting with a different Taylor expansion

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2!}f''(x) - \frac{h^3}{3!}f'''(x) + \cdots$$

we get the *backward difference approximation*

$$f'(x) = \frac{f(x) - f(x-h)}{h} + O(h).$$

To get a better approximation, we can subtract the two Taylor expansions to obtain

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2).$$

This is known as the *central difference formula.* We can also add the two Taylor expansions to obtain

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + O(h^2)$$

To develop a forward difference formula for the second derivative, we can use another expansion

$$f(x+2h) = f(x) + 2hf'(x) + \frac{4h^2}{2!}f''(x) + \frac{8h^3}{3!}f'''(x) + \cdots$$

which gives us

$$f''(x) = \frac{f(x) - 2f(x+h) + f(x+2h)}{h^2} + O(h).$$

A smaller h decreases the truncation error, however it should also be noted that for extremely small h due to finite precision (float, double, etc.) the error can actually increase from round off errors.

Example 3.6. Suppose we are calculating the second derivative using the central difference formula. Assume we incur the same round off error e for computing $f(x+h)$, $f(x)$, and $f(x-h)$. Therefore the total round off error is their sum, $\frac{4e}{h^2}$. Now the truncation error is given by the $O(h^2)$ term in the formula. It has a leading term of $-\frac{h^2}{12}f^{(4)}(x)$. So, we seek the minima for the total error:

$$\begin{aligned}
0 &= \frac{\mathrm{d}}{\mathrm{d}h}\left(\frac{4e}{h^2} + \frac{h^2}{12}f^{(4)}(x)\right) \\
&= -\frac{8e}{h^3} + \frac{h}{6}f^{(4)}(x) \\
h &= \left(\frac{48e}{f^{(4)}(x)}\right)^{\frac{1}{4}}
\end{aligned}$$

$\Diamond$

3.7 Richardson extrapolation

Richardson extrapolation is a method to generate high accuracy results from low order formulas.

Let $D(h)$ be an approximation to a derivative f' with step size h. Then $f' = D(h) + E(h)$ where $E(h) = ch^p$ is the error. Then if we perform the calculation with two step sizes,

$$\begin{aligned}
F &= D(h_1) + ch_1^p \\
F &= D(h_2) + ch_2^p.
\end{aligned}$$

We can eliminate c and obtain

$$F = \frac{h_2^p D(h_1) - h_1^p D(h_2)}{h_2^p - h_1^p}$$

Example 3.7. Starting from the central difference formula

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{6}f'''(x) + \ldots$$

we want to increase its accuracy. This means finding another formula that allows us to remove the trailing term. Using a step size of $2h$, we get

$$f'(x) = \frac{f(x+2h) - f(x-2h)}{4h} - \frac{4h^2}{6}f'''(x) + \ldots.$$

Now combining the two equations we are able to obtain

$$f'(x) = \frac{f(x-2h) - 8f(x-h) + 8f(x+h) - f(x+2h)}{12h} + O(h^4).$$

$\Diamond$

Example 3.8. Now consider the forward difference formula

$$f'(x) = \frac{f(x+h) - f(x)}{h} + O(h).$$

A better approximation is obtained with $h_1 = 2h_2$

$$f'(x) = \frac{-3f(x) + 4f(x+h) - f(x+2h)}{2h} + O(h^2).$$

$\Diamond$

Richardson extrapolation can be used for many other numerical methods other than derivatives. We will see an use for it in integration later on.

3.8 Curve fitting

Consider the least squares error

$$S(a_1, \dots a_m) = \sum_{k=1}^{N} (y_k - Y(x_k, a_1, \dots a_m))^2,$$

where y_k is the provided data points at point x_k and Y is the proposed function we are trying to fit. The goal is to minimize S. The question can be rephrased as finding the point at which $\frac{\partial S}{\partial a_i} = 0$ for $i = 1, \dots, m$.

This gives rise to a system of equations which can be solved by Newton's method. We perform the iteration

$$\mathbf{J}(\mathbf{x})\Delta\mathbf{x} = -\mathbf{f}(\mathbf{x})$$

where $\mathbf{J}_{ij}\mathbf{x}_j = \sum_j \frac{\partial f_j}{\partial x_j}\Delta x_j$ and $\mathbf{f} = f_i$. Only that here, the system in question is $f_i = \frac{\partial S}{\partial a_i}$, so we have

$$\sum_j \frac{\partial^2 S}{\partial a_i \partial a_j}\Delta a_j = -\frac{\partial S}{\partial a_i}.$$

The terms $S_{ij} = \frac{\partial^2 S}{\partial a_i \partial a_j}$ form the Hessian matrix which we denote by $\mathbf{S}$. Hence the solution to our problem is obtained by iterating the following equation

$$\sum_{j=1}^{m} S_{ij}(a_1, a_2, \dots, a_m)\Delta a_j = -\frac{\partial S(a_1, a_2, \dots, a_m)}{\partial a_i}$$

with trial values $a_1, a_2, \dots a_m$. The derivatives can be approximated with repeated applications of the finite difference approximation.

$$\begin{aligned}
S_{ii} &\approx \frac{S(a_i + \delta_i, \dots) - 2S(a_i, \dots) + S(a_i - \delta_i, \dots)}{\delta_i^2} \\
S_{ij} &\approx \frac{1}{2\delta_i}\left[\frac{S(a_i + \delta_i, a_j + \delta_j, \dots) - S(a_i + \delta_i, a_j - \delta_j, \dots)}{2\delta_j}\right. \\
&\qquad \left. - \frac{S(a_i - \delta_i, a_j + \delta_j, \dots) - S(a_i - \delta_i, a_j - \delta_j, \dots)}{2\delta_j}\right]
\end{aligned}$$

Algorithm 3.1: Non-linear curve fitting.

```
function nonlinear($x$):
    $\mathbf{X} \leftarrow x$ data points;
    $\mathbf{Y} \leftarrow y$ data points;
    $\mathbf{A} \leftarrow$ initial trial values;
    while $\|dx\| > tolerance$ do
        $\mathbf{S} \leftarrow$ hessian($\mathbf{A}$, $\delta$);
        $dx \leftarrow \mathbf{S}^{-1}(-\mathbf{f}(x))$;
        $\mathbf{A} \leftarrow \mathbf{A} + dx$;
```

4 Numerical integration

4.1 Newton-Cotes formula

Consider an integral

$$I = \int_a^b f(x)\mathrm{d}x.$$

We seek a quadrature

$$I \approx \sum_{i=1}^{n} W_i f_i$$

where W_i's are the weights and $f_i = f(x_i)$ at evaluation points x_i. In Newton-Cotes we assume the evaluation points are evenly distributed in the interval of concern $[a, b]$ with distance $d = \frac{b-a}{n-a}$.

In our quadrature the unknowns are the weights. We can approximate $f(x)$ by a polynomial of degree $n-1$ that intersects all x_i. The Lagrange form of this polynomial is

$$P_{n-1}(x) = \sum_{i=1}^{n} f(x_i) l_{i,n}(x).$$

Therefore,

$$\begin{aligned} I &= \int_a^b P_{n-1}(x)\mathrm{d}x \\ &= \sum_{i=1}^{n} \left(f(x) \int_a^b l_{i,n}(x)\mathrm{d}x \right) \end{aligned}$$

A comparison tells us the weights are given by

$$W_i = \int_a^b l_{i,n}\mathrm{d}x$$

4.1.1 Trapezoidal approximation

Using two data points ($n = 2$), we get a simple trapezoidal approximation. We have $l_{1,2} = \frac{x-b}{a-b}$. Therefore

$$W_1 = \int_a^b \frac{x-b}{a-b}\mathrm{d}x = \frac{b-a}{2}$$

and for $l_{2,2} = \frac{x-a}{b-a}$ we have

$$W_2 = \frac{b-a}{2}.$$

Therefore

$$I \approx (f(a) + f(b))\frac{b-a}{2}.$$

The error term is $-\frac{(b-a)^3}{12} f''(x)$. This error term can be obtained from a Taylor series expansion which we skip.

4.1.2 Simpson's one-third rule

Newton-Cotes with $n = 3$ gives us Simpson's one-third rule. Our points are located at

$$x_1 = a \qquad x_2 = \frac{a+b}{2} \qquad x_3 = b.$$

separated by $d = \frac{b-a}{2}$. Lagrange interpolation gives

$$l_{1,3}(x) = \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)} \quad l_{2,3}(x) = \frac{(x-x_2)(x-x_3)}{(x_2-x_1)(x_2-x_3)} \quad l_{3,3}(x) = \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)}$$

A change of variables makes our life easier. We center the origin around $\xi = x_2$, so

$$\xi = x - x_2 \qquad \xi_1 = -h \qquad \xi_2 = 0 \qquad \xi_3 = h.$$

Then,

$$W_i = \int_a^b l_{i,n}(x)\mathrm{d}x = \int_{-h}^h l_{i,n}(\xi)\mathrm{d}\xi.$$

We can plug in the new variables to obtain

$$\begin{aligned} W_1 &= \int_{-h}^h \frac{(\xi-0)(\xi-h)}{(-h)(-2h)}\mathrm{d}\xi = \frac{h}{3} \\ W_2 &= \int_{-h}^h \frac{(\xi+h)(\xi-h)}{(-h)(-h)}\mathrm{d}\xi = \frac{4h}{3} \\ W_3 &= \int_{-h}^h \frac{(\xi+h)(\xi-0)}{(2h)(h)}\mathrm{d}\xi = \frac{h}{3}. \end{aligned}$$

This gives

$$I = (f(a) + 4f(-h) + f(b))\frac{h}{3}$$

with error $-\frac{h^5}{90} f''''(\xi)$. It might be noted that this method is exact for cubic polynomials.

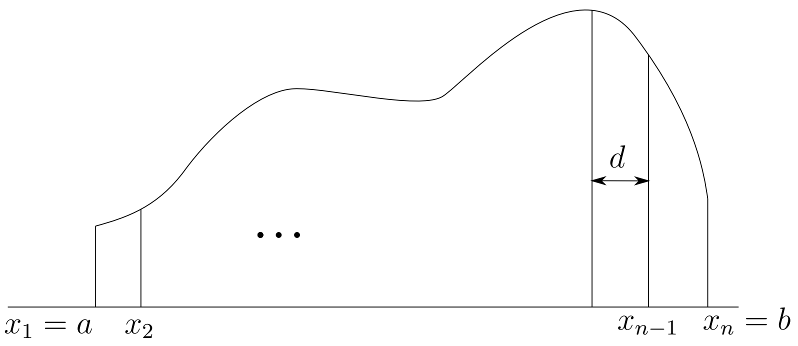
For $n = 4$ we get the Simpson's 3/8 rule, give by

$$\int_{x_1}^{x_4} f(x)\mathrm{d}x \approx \frac{3d}{8}(f(x_1) + 3f(x_2) + 3f(x_3) + f(x_4))$$

with error $-\frac{3d^5}{80} f''''(x)$.

4.2 Composite rules

In interpolation we avoid high degree polynomials to avoid oscillation. We subdivide the intervals and use low degree polynomials in them instead. Numerical integration is similar. We may subdivide a curve into panels:



Then we apply the trapezoidal rule in a piecewise fashion. The approximate area of the i-th panel is

$$I_i = (f(x_i) + f(x_{i+1}))\frac{d}{2}$$

and the total area is

$$I = \sum_{i=1}^{n-1} I_i = (f(x_1) + 2f(x_2) + 2f(x_3) + \cdots + 2f(x_{n-1}) + f(x_n))\frac{d}{2}.$$

For Simpson's one-third rule, the area of two adjacent panels (d is still the width of a single panel) is

$$I_i = (f(x_i) + 4f(x_{i+1}) + f(x_{i+2}))\frac{d}{3}.$$

Thus the total area is approximately

$$I = \sum_{i=1,3,\dots}^{n-2} I_i = (f(x_1) + 4f(x_2) + 2f(x_3) + 4f(x_4) + \cdots + 2f(x_{n-1}) + 4f(x_{n-1}) + f(x_n))\frac{d}{3}.$$

This formula only works for an even number of panels. For odd panels we have to combine this with another method. For example we can leave out the last three panels and use the 3/8 rule.

4.3 Euler-McClaurin

Consider the integral of $f(x)$ as an integral over its Taylor series about $x = a$ instead:

$$\begin{aligned} I &= \int_a^b f(x)\mathrm{d}x \\ &= \int_a^b \sum_{n=0}^{\infty} \frac{(x-a)^n}{n!} f^{(n)}(a) \\ &= \sum_{n=0}^{\infty} \frac{(b-a)^{n+1}}{(n+1)!} f^{(n)}(a). \end{aligned}$$

Do the same thing but expanding about $x = b$ and we obtain

$$I = \sum_{n=0}^{\infty} -\frac{(a-b)^{n+1}}{(n+1)!} f^{(n)}(a).$$

Add them together and divide by two:

$$\begin{aligned} I &= \sum_{n=0}^{\infty} \frac{1}{2} \frac{(b-a)^{n+1} - (a-b)^{n+1}}{(n+1)!} f^{(n)}(a) \\ &= \frac{b-a}{2}(f(a)+f(b)) - \frac{(b-a)^2}{4}(f'(a) - f'(b)) + \cdots . \end{aligned}$$

The first term we recognize as the trapezoidal rule. Notice that odd derivatives appear as differences while even derivatives appear as sums. Consider the Taylor expansion of $f'(x)$ about $x = a$ and then evaluating at b:

$$f'(b) = f'(a) + (b-a)f''(a) + \frac{(b-a)^2}{2} f'''(a) + \cdots .$$

Similarly for expanding about $x = b$ and evaluating at a.

$$f'(a) = f'(b) - (b-a)f''(a) + \frac{(b-a)^2}{2} f'''(a) + \cdots .$$

Subtracting the second from the first gives

$$f''(a) + f''(b) = \frac{2}{b-a}(f'(b) - f'(a)) - \frac{b-a}{2}(f'''(a) - f'''(b)) - \cdots .$$

We can also do the same thing for $f'''(x)$, and get

$$f^{(4)}(a) + f^{(4)}(b) = \frac{2}{b-a}(f'''(b) - f'''(a)) - \cdots ,$$

Using these equations allow us to express the integral in terms of odd derivatives.

$$I = \frac{b-a}{2}(f(a)+f(b)) + \frac{(b-a)^2}{12}(f'(a) - f'(b)) - \frac{(b-a)^4}{720}(f'''(a) - f'''(b)) + \dots .$$

This is the trapezoidal rule with corrections.

Now consider two *equally spaced* panels $[a, b]$ and $[b, c]$. Then

$$\begin{aligned} \int_a^c f \mathrm{d}x &= \int_a^b f \mathrm{d}x + \int_b^c f \mathrm{d}x \\ &= \frac{b-a}{2}[f(a) + f(b) + f(b) + f(b)] + \frac{(b-a)^2}{12}[f'(a) - f'(b) + f'(b) - f'(c)] + \cdots . \end{aligned}$$

We can see that a general formula for n equally spaced panels is given by

$$\int_{x_1}^{x_n} f \mathrm{d}x = \frac{d}{2}(f(x_1) + \cdots + f(x_n)) + \frac{d^2}{12}(f'(x_1) - f'(x_n)) - \frac{d^4}{720}(f'''(x_1) - f'''(x_n)) + \cdots$$

where $d = x_2 - x_1$ is the distance between panels. This is the Euler-Maclaurin integration rule.

4.4 Romberg Intergration

Here we combine the trapezoidal rule with Richardson extrapolation. Denote the result obtained from the trapezoidal rule with $n = 2^{m-1}$ panels as $T_{m,1}$. We know this has $O(d^2)$ error. If we use an interval half as large, Richardson extrapolation gives

$$T = \frac{2^p T(\frac{d}{2}) - T(d)}{2^p - 1}$$

For example

$$T_{m,2} = \frac{4T_{m,1} - T_{m-1,1}}{3}$$

gives us an $O(d^4)$ error. Applied once more we get

$$T_{m,3} = \frac{16T_{m,2} - T_{m-1,2}}{15}$$

with an error of $O(d^6)$.

In general, we have

$$T_{m,k} = \frac{4^{k-1}T_{m,k-1} - T_{m-1,k-1}}{4^{k-1} - 1}.$$

4.5 Improper integrals

An improper integral is one where there are singularities along the path of integration or if one of the limits are at infinity. For singularities, since we can always split the interval of integration, we will only consider the case where singularities appear at one of the limits of integration.

An easy technique we can try first is substitution. For example we can try $x = \frac{1+y}{1-y}$ which maps $[0, \infty] \to [-1, 1]$ or $x = \frac{y}{1-y}$ which maps $[0, \infty] \to [0, 1]$.

Another technique is to rewrite $\int f \mathrm{d}x$ as $\int \frac{f}{g} g \mathrm{d}x$ with an appropriate g such that $\frac{f}{g}$ is easier to integrate, then we use the substitution $\mathrm{d}y = g\mathrm{d}x$.

Example 4.1. Consider $\int_0^\infty \frac{1}{(1+x)\sqrt{x}} \mathrm{d}x$. Let $g(x) = \frac{1}{\sqrt{x}}$ and $\mathrm{d}y = \frac{\mathrm{d}x}{g(x)}$. This means $y = 2\sqrt{x}$. Then

$$\begin{aligned}\int_0^\infty \frac{1}{(1+x)\sqrt{x}} &= \int_0^\infty \frac{1}{(1+x)\sqrt{x}} \sqrt{x} \frac{\mathrm{d}x}{\sqrt{x}} \\ &= \int_0^\infty \frac{1}{1+\frac{y^2}{4}} \mathrm{d}y\end{aligned}$$

Next, let $y = \frac{z}{1-z}$.

$$\int_0^\infty \frac{1}{1+\frac{y^2}{4}} \mathrm{d}y = \int_0^1 \frac{4}{4(1-z)^2 + z^2} \mathrm{d}z$$

which is easy to evaluate numerically. ♢

Example 4.2. Consider $\int_1^\infty x^{-\frac{3}{2}} \sin \frac{1}{x} \mathrm{d}x$. Let $g(x) = x^{-\frac{3}{2}}$ and $\mathrm{d}y = \frac{\mathrm{d}x}{g(x)}$. This means $y = \frac{-2}{\sqrt{x}}$. Note that this changes the limits of integration. Then

$$\begin{aligned} \int_1^\infty x^{-\frac{3}{2}} \sin \frac{1}{x} \mathrm{d}x &= \int_1^\infty \sin \frac{1}{x} \frac{\mathrm{d}x}{x^{-\frac{3}{2}}} \\ &= \int_{-2}^0 \sin \frac{y^2}{4} \mathrm{d}y. \end{aligned}$$

◊

4.6 Gaussian integration

Consider integrals of the form

$$\int_a^b f(x)w(x)\mathrm{d}x = \sum_{m=1}^n W_m f(x_m).$$

where $w(x)$ is a positive definite weighting function. This is similar to Newton-Cotes, but in this case the abscissas x_m's are not evenly spaced. Trapezoidal approximation is exact for linear polynomials and Simpsons's is exact for cubic polynomials. Gaussian quadrature is exact for polynomials of degree $2n-1$.

Let $f(x)$ be a polynomial of degree $2n-1$ and $\{\phi_i(x)\}$ be a complete set of orthogonal polynomials. We can write

$$f_{2n-1}(x) = q_{n-1}(x)\phi_n(x) + r_{n-1}(x).$$

where q_{n-1} and r_{n-1} are polynomials of order at most $n-1$. We can also express q_{n-1} as a linear combination of ϕ's, $q_{n-1}(x) = \sum_{i=0}^{n-1} \alpha_i \phi_i(x)$. Then we have

$$\begin{aligned} \int_a^b q_{n-1}(x)\phi_n(x)w(x)\mathrm{d}x &= \sum_{i=0}^{n-1} \alpha_i \int_a^b \phi_i(x)\phi_n(x)w(x)\mathrm{d}x \\ &= \sum_{i=0}^{n-1} \alpha_1 \delta_{in} C_n \\ &= 0. \end{aligned}$$

Using the quadrature formula,

$$\int_a^b q_{n-1}(x)\phi_n(x)w(x)\mathrm{d}x = \sum_{m=1}^n W_m q_{n-1}(x_m)\phi_n(x_m) = 0.$$

Therefore this means that $\phi_n(x_m) = 0$ for all x_m, since f_{2n-1} and hence q_{n-1} are arbitrary functions. This means that x_m are chosen to be the zeros of ϕ_n. What is left is to determine the weights W_m.

Consider

$$l_{j,n}(x) = \prod_{\substack{k=1 \\ k\neq j}}^n \frac{x - x_k}{x_j - x_k}.$$

Note that

$$l_{i,n}(x_j) = \delta_{ij}.$$

Thus, using the quadrature formula,

$$\int_a^b l_{j,n}(x)w(x)\mathrm{d}x = \sum_{m=1}^{n} W_m l_{j,n}(x_m) = W_j$$

Compare with Netwon-Cotes where $W_j = \int_a^b l_{j,n}(x)\mathrm{d}x$.

4.6.1 Gauss-Legendre quadrature

Gauss-Legendre is the most used Gaussian quadrature. We shall develop an way to integrate integrals of the form

$$I = \int_{-1}^{1} f(x)\mathrm{d}x.$$

Legendre polynomials form an appropriate basis set. The abscissas are the zeros of ϕ_n. For example for $n = 2$ (evaluating at two points) we have the abscissas

$$x_1 = -\frac{1}{\sqrt{3}} \qquad\qquad x_2 = \frac{1}{\sqrt{3}}$$

The weights are given by $W_i = \int_{-1}^{1} l_{i,n}(x)\mathrm{d}x$, so we have for $n = 2$

$$\begin{aligned} W_1 &= \int_{-1}^{1} \frac{x - x_2}{x_1 - x_2}\mathrm{d}x \\ &= 1 \end{aligned}$$

and

$$\begin{aligned} W_2 &= \int_{-1}^{1} \frac{x - x_1}{x_2 - x_1}\mathrm{d}x \\ &= 1. \end{aligned}$$

Thus we have

$$I = \int_{-1}^{1} f(x)\mathrm{d}x = f\left(-\frac{1}{\sqrt{3}}\right) + f\left(\frac{1}{\sqrt{3}}\right).$$

The quadrature is exact for polynomials of degree $2n - 1$, in this case, cubic polynomials.

For an arbitrary integration interval, we can use a change of variables

$$x = \frac{b+a}{2} + \frac{b-a}{2}\xi$$

$$\int_a^b f(x)\mathrm{d}x = \frac{b-a}{2}\int_{-1}^{1} f(x)\mathrm{d}\xi \approx \frac{b-a}{2}\sum_{i=1}^{n} W_i f(x_i).$$

When computing the quadrature, take care that ξ takes up the values of the abscissas, and no longer x, despite the notation!

Example 4.3.

$$\begin{aligned} \int_0^1 x^3\mathrm{d}x &= \frac{1}{2}\left[\left(\frac{1}{2} - \frac{1}{2\sqrt{3}}\right)^3 + \left(\frac{1}{2} + \frac{1}{2\sqrt{3}}\right)^3\right] \\ &= \frac{1}{4} \end{aligned}$$

◊

4.7 Multidimensional integration

Consider a double integral

$$I = \int_a^b \int_c^d f(x,y)\mathrm{d}x\mathrm{d}y.$$

We can simply compute

$$I = \int_a^b F(y)\mathrm{d}y \qquad\qquad F(y) = \int_c^d f(x,y)\mathrm{d}x.$$

It is not even a problem if the limits are not constants. We just evaluate the nested integral like any other function.

Example 4.4. Simpson's one-third rule:

$$\begin{aligned}
\int_0^1 \int_0^y \frac{x}{x+y}\mathrm{d}x\mathrm{d}y &= \int_0^1 \frac{y/2}{3}\left[0 + 4\frac{y/2}{y/2+y} + \frac{y}{y+y}\right]\mathrm{d}y \\
&= \int_0^1 \frac{11}{36} y\mathrm{d}y \\
&= \frac{11}{36}\frac{1/2}{3}(0 + \frac{4}{2} + 1) \\
&= \frac{11}{72}.
\end{aligned}$$

$\diamondsuit$

4.8 Monte Carlo integration

Since we have the average

$$\langle f \rangle = \frac{1}{b-a}\int_a^b f(x)\mathrm{d}x$$

we can find the integral if we have a way of finding the average. We can do this by generating random values uniformly in the interval $[a,b]$, and evaluating f there, then taking the average.

To reduce the error, we can use importance sampling. If we can find a $g(x) \approx f(x)$ that we know how to integrate, then we can write

$$\int_a^b f(x)\mathrm{d}x = \int_a^b \frac{f(x)}{g(x)}(g(x)\mathrm{d}x).$$

Setting $\mathrm{d}y = g(x)\mathrm{d}x$, where $y = \int g(x)\mathrm{d}x$. Instead of uniformly sampling x to integrate $f(x)$, we uniformly sample y and integrate $\frac{f(x)}{g(x)}$. This makes the integrand flatter and thus increases the accuracy of the randomly sampled average.

Example 4.5. Consider $\int_0^1 e^x \mathrm{d}x$. From the Taylor series we have $e^x \approx 1 + x$. We have

$$y = \int 1 + x\mathrm{d}x = x + \frac{x^2}{2}.$$

and

$$\int_0^1 \frac{e^x}{1+x}(1+x)\mathrm{d}x = \int_0^{\frac{3}{2}} \frac{e^{\sqrt{1+2y}-1}}{\sqrt{1+2y}}\mathrm{d}y$$
$$\approx \frac{3}{2}\frac{1}{n}\sum_{i=1}^{n}\frac{e^{\sqrt{1+2y_i}-1}}{\sqrt{1+2y_i}}.$$

$\Diamond$

5 Ordinary Differential Equations

5.1 Lowering order of ODEs

We can lower any higher order ODEs to first order ODEs by adding auxiliary variables. This means we only need to know how to solve first order ODEs.

Consider $y'' = f(x, y, y')$. Let $y_1 = y$ and $y_2 = y'$. Then, we can rewrite the second order ODE in terms of coupled first order equations

$$\begin{cases} y_1' = y_2 \\ y_2' = f(x, y_1, y_2) \end{cases}.$$

This can be convenient expressed in the matrix equation

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix}' = \begin{pmatrix} y_2 \\ f(x, y_1, y_2). \end{pmatrix}$$

For a third order ODE $y''' = f(x, y, y', y'')$, we can rewrite it as

$$\begin{cases} y_1' = y_2 \\ y_2' = y_3 \\ y_3' = f(x, y, y_2, y_3) \end{cases}.$$

5.2 Euler Methods

5.2.1 Forward, backward Euler methods

Consider the ODE

$$y' = f(x, y)$$

with initial conditions $y(x_0) = y_0$. The second derivative is given by

$$y'' = \frac{\partial}{\partial x}f + f\frac{\partial}{\partial y}f.$$

Therefore the second order Taylor expansion of y about x_0 is

$$y(x) = y_0 + (x - x_0)f(x_0, y_0) + \frac{(x-x_0)^2}{2!}\left[\frac{\partial f(x_0, y_0)}{\partial x} + f(x_0, y_0)\frac{\partial f(x_0, y_0)}{\partial y}\right] + \frac{(x-x_0)^3}{3!}y'''(\xi)$$

where $x_0 \leq \xi \leq x$, and if known, allows us to obtain the error directly. Letting $h = x - x_0$, we summarise it as follows

$$y(x_0 + h) = y(x_0) + hf(x_0, y(x_0)) + O(h^2)$$

and this is known as the *forward Euler method.* One may think of it as a linear extrapolation of the gradient to predict the value of y. Similarly the *backward Euler method* arises from the Taylor expansion

$$y(x_0) = y(x) - hy'(x) + \frac{h^2}{2!}y''(x) + \cdots$$

and some rearrangement gives

$$y(x) = y(x_0) + hy'(x) - \frac{h^2}{2!}y''(x) + \cdots$$

and so

$$y(x_0 + h) = y(x_0) + hf(x_0 + h, y(x_0 + h)) + \cdots$$

The two methods both have the same accuracy, but the forward Euler method is an explicit scheme whereas backward Euler is implicit. Explicit schemes calculate the state of the system at a future time using initial conditions. Implicit schemes solve equations involving the future and current state of the system. The solution of these equations cause an overhead, but they are usually more numerically stable. Explicit schemes are usual numerically unstable unless an extremely small step size is used.

Example 5.1. Consider the ODE

$$y' = -100y.$$

Using the forward Euler method we have the recurrence

$$y_{i+1} = y_i + hf_i = y_i(1 - 100h).$$

Therefore,

$$y_n = (1 - 100h)^n y_0$$

which means that a forward Euler method is stable only when $|1 - 100h| \leq 1$. Using a backward Euler method we have

$$y_{i+1} = y_i + hf_{i+1} = y_i + h(-100y_{i+1})y_{i+1} = \frac{y_i}{1 + 100h}.$$

Therefore

$$y_n = \frac{y_0}{(1 + 100h)^n}.$$

This method is unconditionally stable. ◊

5.2.2 Modified Euler method

In the modified Euler method, we take the derivative at the midpoint instead of at the beginning of the interval. Let $x_m = x_0 + \frac{h}{2}$. From $y' = f(x, y)$, integrate both sides

$$\begin{aligned} y(x_0 + h) - y(x_0) &= \int_{x_0}^{x_0+h} f(\xi, y)\mathrm{d}\xi \\ &\approx \int_{x_0}^{x_0+h} f(x_m, y_m) + (\xi - x_m)f' + \cdots \mathrm{d}\xi \\ &= hf(x_m, y_m) + O(h^3) \end{aligned}$$

where y_m is approximated using a Taylor expansion:

$$y_m \approx y(x_0) + \frac{h}{2} f(x_0, y_0).$$

In summary:

$$y(x_0 + h) \approx y(x_0) + hf\left(x_0 + \frac{h}{2} + y_0 + \frac{h}{2} f_0\right).$$

5.2.3 Improved Euler method

An improvement to the Euler method, also known as Heun's method, is to average the slopes at the start and the end of the interval. Since we do not know the value for y_1, we make a guess $y_1 \approx y_0 + hf_0$.

$$m = \frac{f(x_0, y_0) + f(x_1, y_0 + hf(x_0 + y_0))}{2}$$

This average slope is then used to extrapolate linearly from y_0 to y_1 using the forward Euler method,

$$y_1 = y_0 + hm.$$

The Crank-Nicolson method is obtained by converting this to an implicit scheme:

$$y(x_0 + h) = y(x_0) + h\frac{f(x_0, y_0) + f(x_0 + h, y_0 + h)}{2}$$

Example 5.2. Consider the ODE

$$y' = -y \qquad\qquad y(0) = 1$$

We have via the Crank-Nicolson method

$$\begin{aligned} y_1 &= y_0 + \frac{h}{2}(f_0 + f_1) \\ &= y_0 + \frac{h}{2}(-y_0 - y_1) \\ &= \frac{1 - \frac{h}{2}}{1 + \frac{h}{2}} y \end{aligned}$$

It is clear that this is unconditionally stable. ◊

5.3 Runge-Kutta Methods

The Runge-Kutta method is a widely used method for solving initial value problems. The basic idea behind it is to approximate the integral by a weighted average of slopes and to approximate the slopes at a number of points.

First we note that the Euler methods can be written as a linear combination of slopes.

$$y(x_0 + h) = y(x_0) + h[\alpha f_0 + \beta f(x_0 + \gamma g, y_0 + \delta h f_0)].$$

For example in the modified Euler method we have $\alpha = 0$, $\beta = 1$, $\gamma = \delta = \frac{1}{2}$.

This can be written as a Taylor expansion

$$\begin{aligned} y_1 &= y_0 + h\alpha f_0 + h\beta\left[f(x_0, y_0) + \gamma h\frac{\partial f_0}{\partial x} + \delta h f_0\frac{\partial f_0}{\partial y} + O(h^2)\right] \\ &= y_0 + h(\alpha+\beta)f_0 + h^2\beta\left[\gamma\frac{\partial f_0}{\partial x} + \delta f_0\frac{\partial f_0}{\partial y}\right] + O(h^3). \end{aligned}$$

Comparing this to the Taylor series expansion of y,

$$y(x) = y_0 + hf_0 + \frac{h^2}{2!}\left[\frac{\partial f_0}{\partial x} + f_0\frac{\partial f_0}{\partial y}\right] + O(h^3)$$

we see that $\alpha + \beta = 1$, $\beta\gamma = \beta\delta = \frac{1}{2}$ makes the original expression agree with the Taylor series. Following are some popular choices:

- Modified Euler: $\alpha = 0$, $\beta = 1$, $\gamma = \delta = \frac{1}{2}$.
- Heun's: $\alpha = \frac{1}{2}$, $\beta = \frac{1}{2}$, $\gamma = \delta = 1$.
- Ralston's: $\alpha = \frac{1}{3}$, $\beta = \frac{2}{3}$, $\gamma = \delta = \frac{3}{4}$.

Ralston's method is the best among the second order Runge-Kutta methods.

The most well known method among the Runge-Kutta methods is the fourth order Runge-Kutta method. As with the second order case, there are an infinite number of versions. The most common however is defined with the intermediate quantities

$$f_0 = f(x_0, y_0) \qquad f_1 = f\left(x_0 + \frac{h}{2}, y_0 + \frac{h}{2}f_0\right)$$

$$f_2 = f\left(x_0 + \frac{h}{2}, y_0 + \frac{h}{2}f_1\right) \qquad f_3 = f(x_0 + h, y_0 + hf_2)$$

and the solution is expressed as

$$y(x_0 + h) = y(x_0) + \frac{h}{6}(f_0 + 2f_1 + 2f_2 + f_3).$$

5.4 Verlet algorithm

One inefficiency of the Runge-Kutta method is the need for repeated evaluation of the function every time step. For example in the fourth order Runge-Kutta method, four evaluations are needed every step. This can be expensive. Verlet integration is a method frequently used to give numerical solutions for Newton's equations of motion. Consider the equations of motion

$$\ddot{\mathbf{r}} = \mathbf{a}(\mathbf{r}) \qquad \dot{\mathbf{r}} = \mathbf{v}.$$

Using the central difference formula, we have

$$\frac{\mathbf{r}_{n+1} + r_{n-1} - 2\mathbf{r}_n}{h^2} + O(h^2) = \mathbf{a}_n \qquad \frac{\mathbf{r}_{n+1} - \mathbf{r}_{n-1}}{2h} + O(h^2) = \mathbf{v}_n$$

Solving for $\mathbf{r}_{n+1}$, we have

$$\mathbf{r}_{n+1} = 2\mathbf{r}_n - \mathbf{r}_{n-1} + h^2\mathbf{a}_n + O(h^4) \qquad \mathbf{v}_n = \frac{\mathbf{r}_{n+1} - \mathbf{r}_{n-1}}{2h} + O(h^2)$$

which only requires one function evaluation per step. This iterative scheme is not self starting (since $\mathbf{r}_{-1}$ does not exist). We need another method like Runge-Kutta to provide the first value $\mathbf{r}_1$. Another problem is the expression for $\mathbf{v}$ involves the difference between two quantities of similar magnitude, which may lead to the loss of precision. A more common method is the velocity Verlet method. Adding r_{n+1} on both sides, gives

$$\begin{aligned} 2\mathbf{r}_{n+1} &= 2\mathbf{r}_n + \overbrace{\mathbf{r}_{n+1} - \mathbf{r}_{n-1}}^{2h\mathbf{v}_n + O(h^3)} + h^2\mathbf{a}_n + O(h^4) \\ \mathbf{r}_{n+1} &= \mathbf{r}_n + h\mathbf{v}_n + \frac{1}{2}h^2\mathbf{a}_n. \end{aligned}$$

Now consider the original Verlet method expression to one obtained by replacing $n+1$ by n.

$$\begin{aligned} \mathbf{r}_{n+1} &= 2\mathbf{r}_n - \mathbf{r}_{n-1} + h^2\mathbf{a}_n + O(h^4) \\ \mathbf{r}_n &= 2\mathbf{r}_{n-1} - \mathbf{r}_{n-2} + h^2\mathbf{a}_{n-1} + O(h^4). \end{aligned}$$

We add them and rearrange to obtain

$$\begin{aligned} \overbrace{\mathbf{r}_{n+1} - \mathbf{r}_{n-1}}^{2h\mathbf{v}_n} &= \overbrace{\mathbf{r}_n - \mathbf{r}_{n-2}}^{2h\mathbf{v}_{n-1}} + h^2(\mathbf{a}_{n-1} + \mathbf{a}_n) \\ \mathbf{v}_n &= \mathbf{v}_{n-1} + \frac{h}{2}(\mathbf{a}_{n-1} + \mathbf{a}_n). \end{aligned}$$

In summary, the velocity Verlet method is given by

$$\mathbf{r}_{n+1} = \mathbf{r}_n + h\mathbf{v}_n + \frac{1}{2}h^2\mathbf{a}_n \qquad \mathbf{v}_{n+1} = \mathbf{v}_n + \frac{h}{2}(\mathbf{a}_n + \mathbf{a}_{n+1}).$$

Note that it is self-starting.

5.5 Bulirsch-Stoer method

The Burlirsch-Stoer method gives high accuracy but only works for smooth functions. Consider $y'(x) = f(x, y)$. Integrating gives

$$\int_{x+h} x - hy'(t)\mathrm{d}y = \int_{x+h} x - hf(u, y)\mathrm{d}u.$$

We can approximate this integral at the midpoint of the integration range, which gives

$$y(x+h) - y(x-h) = 2hf(x, y).$$

Consider advancing the solution of the problem from x_0 to $x_0 + H$. We divide the interval into n steps of equal length $h = \frac{H}{n}$. Also note that the scheme is not self starting (there is no $x_0 - h$), so we shall use the forward Euler method for the first step. The iterative scheme is as follows:

$$\begin{aligned} y_1 &= y_0 + hf_0 \\ y_2 &= y_0 + 2hf_1 \\ y_3 &= y_1 + 2hf_2 \\ &\vdots \\ y_n &= y_{n-2} + 2hf_{n-1} \end{aligned}$$

Lastly we use the backward Euler to find the value of y_n

$$y_n = y_{n-1} + hf_n$$

and take the average of the two y_n values obtained to get the final result. In summary

$$y(x_0 + H) = \frac{1}{2}(y_n + y_{n-1} + hf_n).$$

The interesting part about this method is that the error term consists of only even power terms of h. Therefore, we can use Richardson extrapolation to eliminate leading error terms and this gives us very high accuracy. For example, we can repeat the process first with h and then again with $\frac{h}{2}$. Let the results obtained be g_1 and g_2 respectively, and a better approximation would be

$$y(x_0 + H) = \frac{4g_2 - g_1}{3}$$

5.6 Applications

5.6.1 Projectile motion

Consider a flight of a projectile. It has mass m, position $\mathbf{r}(t)$ and velocity $\mathbf{v}(t)$. They are related by

$$\frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = \frac{1}{m}\mathbf{F}_a(\mathbf{v}) - g\hat{\mathbf{y}} \qquad\qquad \frac{\mathrm{d}\mathbf{r}}{\mathrm{d}t} = \mathbf{v}.$$

Here $\mathbf{F}_a$ is air resistance. We will assume

$$\mathbf{F}_a = -\frac{1}{2}C_d\rho A|\mathbf{v}|^2\hat{\mathbf{v}}.$$

The drag coefficient $C_d < 1$ for a streamlined object. The whole problem takes the form of the following matrix equation

$$\begin{pmatrix}\mathbf{r}\\ \mathbf{t}\end{pmatrix}' = \begin{pmatrix}\mathbf{v}\\ \frac{1}{m}\mathbf{F}_a(\mathbf{v}) - g\hat{\mathbf{y}}\end{pmatrix}$$

Using the forward Euler gives us the following recurrence:

$$\begin{pmatrix}\mathbf{r}\\ \mathbf{t}\end{pmatrix}_{n+1} = \begin{pmatrix}\mathbf{r}\\ \mathbf{t}\end{pmatrix}_n + \tau\begin{pmatrix}\mathbf{v}\\ \frac{1}{m}\mathbf{F}_a(\mathbf{v}) - g\hat{\mathbf{y}}\end{pmatrix}_n$$

We loop over this until the projectile hits the ground.

5.6.2 Kepler

Consider a small satellite orbiting the sun. The force on the comet is

$$m\ddot{\mathbf{r}} = \mathbf{F} = -\frac{GmM}{|\mathbf{r}|^2}\hat{\mathbf{r}}$$

where $\mathbf{r}$ is the position of the satellite, m is its mass, and M is the mass of the sun. This is expressed as the following matrix equation:

$$\begin{pmatrix} \mathbf{r} \\ \mathbf{v} \end{pmatrix}' = \begin{pmatrix} \mathbf{v} \\ -G\frac{M}{|\mathbf{r}|^3}\hat{\mathbf{r}} \end{pmatrix}$$

Or more verbosely:

$$\begin{pmatrix} r_x \\ r_y \\ v_x \\ v_y \end{pmatrix}' = \begin{pmatrix} v_x \\ v_y \\ \frac{F_x}{m} \\ \frac{F_y}{m} \end{pmatrix}.$$

5.6.3 Driven pendulum

Consider a pendulum consisting of a rod of length l and a point mass m attached on one end. The pendulum is driven by a driving force f_d and experiences a resistive force f_r. Newton's laws of motions states that

$$ma_t = f_g + f_d + f_r$$

where $f_g = -mg\sin\theta$ is the contribution by gravity, and $a_t = l\ddot{\theta}$ is the tangential acceleration.

Assume that the driving force is periodic, given as $f_d(t) = f_0 \cos\omega t$, and the resistive force is $f_r = -kv$ where $v = l\dot{\theta}$ is the tangential velocity and k is some constant parameter. The problem can be expressed as the following equation

$$\ddot{\theta} + q\dot{\theta} + \sin\theta = b\cos\omega t$$

where q and b are constants. We can rewrite it in matrix form

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix}' = \begin{pmatrix} y_2 \\ -qy_2 - \sin y_1 + b\cos\omega t \end{pmatrix}.$$

5.7 Relaxation methods

We will now be considering boundary value and eigenvalue problems rather than initial value problems. Consider the differential equation

$$y''(x) = f(y', y, x) \qquad y(a) = \alpha \qquad y(b) = \beta.$$

The value of the dependent variable y is fixed at the boundaries and this is known as a *Dirichlet problem.* We gave seen how we can approximate derivatives. Divide the domain into n equally spaced nodes, $x_1, x_2, \ldots, x_n$. The central difference formula gives

$$y_i' \approx \frac{y_{i+1} - y_{i-1}}{2h} \qquad y_i'' \approx \frac{y_{i+1} - 2y_1 + y_{i-1}}{h^2}$$

We simply substitute these equations in and solve for y_i. Then, we will solve this with a relaxation method.

Example 5.3.

$$y''(x) - 5y'(x) + 10y = 10x \qquad\qquad y(0) = 0 \qquad\qquad y(1) = 100.$$

Substituting the central difference formula gives

$$\frac{y_{i+1} - 2y_1 + y_{i-1}}{h^2} - 5\frac{y_{i+1} - y_{i-1}}{2h} + 10y_i = 10x_i$$

and solving for y_i gives

$$y_i = \frac{1}{2 - 10h^2}\left[\left(1 - \frac{5h}{2}\right)y_{i+1} + \left(1 + \frac{5h}{2}\right)y_{i-1} - 10h^2 x_i\right]$$

◊

5.7.1 Jacobi scheme

Next, we perform the Jacobi scheme. In the Jacobi scheme the old values of y are used to obtain new ones. We make an initial guess for all $y_i^{(0)}$. Then we will evaluate the first iteration $y_i^{(1)}$ using our guessed $y_{i+1}^{(0)}$ and $y_{i-1}^{(0)}$ (and possibly others). Then after evaluating for all i, we will iterate again for $y_i^{(2)}$.

Example 5.4. For the same example above,

$$y_i^{(j)} = \frac{1}{2 - 10h^2}\left[\left(1 - \frac{5h}{2}\right)y_{i+1}^{(j-1)} + \left(1 + \frac{5h}{2}\right)y_{i-1}^{(j-1)} - 10h^2 x_i\right].$$

◊

5.7.2 Gauss-Seidel scheme

The Gauss-Seidel scheme is an improved version of the Jacobi scheme. Instead of always using previous iteration values, whenever an updated value becomes available, it is immediately used.

Example 5.5. Using the same example as above, in determining $y_i^{(j)}$, only the $y_{i+1}^{(j-1)}$ term is not available. So we modify the equation to make use of the most recent values:

$$y_i^{(j)} = \frac{1}{2 - 10h^2}\left[\left(1 - \frac{5h}{2}\right)y_{i+1}^{(j-1)} + \left(1 + \frac{5h}{2}\right)y_{i-1}^{(j)} - 10h^2 x_i\right].$$

◊

In general, convergence can be improved if we employ the *successive over-relaxation* (SOR) method. Express a weighted average of the dependent variable obtained from its current and previous iterations:

$$\overline{y}_i^{(j)} = \alpha y_i^{(j)} + (1 - \alpha)y_i^{(j-1)}.$$

Here α is called the relaxation factor. If $\alpha = 1$, we get the original Gauss-Seidel scheme, and if $1 < \alpha < 2$, we have SOR where convergence is accelerated.

The bounds for α are found in the following way. We can write

$$y_i^{(j+1)} - y_i^{(j)} = \beta\left(y_i^{(j)} - y_i^{(j-1)}\right).$$

with $0 < \beta < 1$ as the rate of convergence. Therefore, this gives us $\alpha = 1 + \beta$, and hence the bound $1 < \alpha < 2$.

5.8 Shooting method

For a second order boundary value problem

$$y''(x) = f(y', y, x) \qquad y(a) = \alpha \qquad y(b) = \beta$$

we can turn it into a initial value problem

$$y''(x) = f(y', y, x) \qquad y(a) = \alpha \qquad y'(a) = u.$$

What we need to do is to find the correct value of u. This can be done by trial and error. We guess u, then solve the initial value problem from $x = a$ to $x = b$. Then we adjust u depending on how close we come to the boundary condition $y(b) = \beta$. We note that the value of y at b is in fact a function that depends on u, i.e. $y(b) = \gamma(u)$. And our goal is to find a u such that $\gamma(u) = \beta$. Now define the function $r(u) = \gamma(u) - \beta$. We have changed this to a root finding problem.

5.9 Finite elements

Consider a simple differential equation $y'' - 6x = 0$ subject to the boundary conditions $y(0) = 0$ and $y(1) = 1$. We guess a quadratic solution $p(x) = \alpha x^2 + \beta x + \gamma$. Plugging in the boundary conditions give us $p(x) = \alpha x^2 + (1 - \alpha)x$. Substituting the trial solution, we get a residual error of $R = p'' - 6x = 2\alpha - 6x$. We can then minimize α. To achieve this, we can consider the integral of the square of the error $I = \int_0^1 R^2 \mathrm{d}x$ and minimize it:

$$\frac{\partial I}{\partial \alpha} = 2\int_0^1 R\frac{\partial R}{\partial \alpha}\mathrm{d}x = 0$$

Evaluating, we get $\alpha = \frac{3}{2}$. Thus $p(x) = \frac{1}{2}(3x^2 - x)$ is the best quadratic approximation to the differential equation. The exact solution is actually x^3.

In general the last step can be expressed as $\int_a^b Rw\mathrm{d}x = 0$, where w is the weighting function. For instance the choice $w = \frac{\partial R}{\partial \alpha}$, we get the *least squares* finite element method.

Now, instead of using high degree polynomials, it might be easier to break the region of interest into smaller intervals, and use low degree polynomials in these subregions. It is thus useful to define basis functions, which the final solution will be a linear combination of. We shall only consider piecewise linear trial functions. Define the basis functions ϕ_i associated with the node x_i as

$$\phi_i(x) = \begin{cases} 0, & \text{if } x \leq x_{i-1} \\ \frac{x - x_{i-1}}{x_i - x_{i-1}}, & \text{if } x_{i-1} \leq x \leq x_i \\ \frac{x_{i+1} - x}{x_{i+1} - x_i}, & \text{if } x_i \leq x \leq x_{i+1} \\ 0, & \text{if } x_{i+1} \leq x_i \end{cases}.$$

It takes up a kind of a triangular shape with a peak at x_i.

Write the trial function as

$$p(x) = \sum_{i=0}^{n} \alpha_i \phi_i(x).$$

The boundary conditions $p(a) = y(a)$ and $p(b) = y(b)$ give $\alpha_0 = y(a)$ and $\alpha_n = y(b)$. To determine the other coefficients, we require that the error

$$\int_a^b R(x)w_i(x)\mathrm{d}x = 0.$$

The *Galerkin method* is widely used in finite element methods, where we set $w_i = \phi_i$. For example, in our previous problem we will get

$$\int_0^1 \frac{\mathrm{d}^2p(x)}{\mathrm{d}x^2}\phi_i(x) - 6x\phi_i(x)\mathrm{d}x = 0.$$

Integrating by parts, the first term gives

$$\begin{aligned}\int_0^1 \frac{\mathrm{d}^2p}{\mathrm{d}x^2}\phi_i(x)\mathrm{d}x &= \frac{\mathrm{d}p}{\mathrm{d}x}\phi_i\bigg|_0^1 - \int_0^1 \frac{\mathrm{d}p}{\mathrm{d}x}\frac{\mathrm{d}\phi_i}{\mathrm{d}x}\mathrm{d}x \\ &= -\sum_{j=0}^{n}\alpha_j\int_0^1 \frac{\mathrm{d}\phi_j(x)}{\mathrm{d}x}\frac{\mathrm{d}\phi_i(x)}{\mathrm{d}x}\mathrm{d}x \\ &= -\sum_{j=0}^{n}\alpha_j \begin{cases}\frac{2}{h}, & \text{if } |i-j| = 0 \\ -\frac{1}{h}, & \text{if } |i-j| = 1 \\ 0, & \text{if } |i-j| > 1.\end{cases} \\ &= \frac{-\alpha_{i-1} + 2\alpha_i - \alpha_{i+1}}{h}.\end{aligned}$$

We can solve the other part of the integral easily to obtain

$$\frac{-\alpha_{i-1} + 2\alpha_i - \alpha_{i+1}}{h} = 6hx_i$$

to get a system of equations and solved for the coefficients.

5.10 Applications

5.10.1 Schrödinger equation

Consider the one dimensional Schrödinger equation

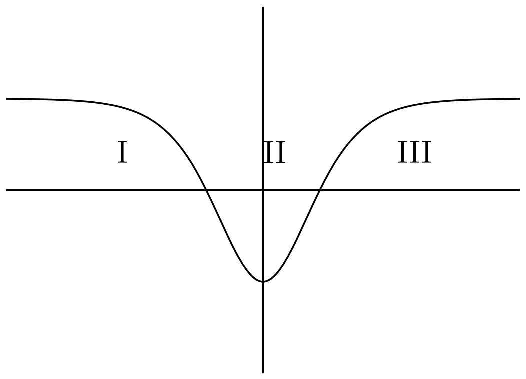
$$\psi''(x) + \frac{2m}{\hbar^2}[E - V(x)]\psi(x) = 0.$$

We want to solve for a particle bound in a potential well

$$V(x) = \frac{6\hbar^2}{m}\left[\frac{1}{2} - \frac{1}{\cosh^2 x}\right]$$

The boundary conditions are $\psi(x) \to 0$ for $x \to \pm\infty$. However there is a small problem here, being that E is also an unknown.

The following is a plot of the potential function. We see that it is made up of two classically forbidden regions I and III. Deep inside region I, we can make the assumption $\psi(x_0) = 0$ and $\psi'(x_0) = \psi_0'$, where ψ_0' is a small positive number. Then, we integrate from x_0 to a point deep



inside region III. However, we will find that we do not get a solution that satisfies the equation regardless of which E we use, even for the exact solution $E = 1$. There is a simple reason for this. Recall that for a square well, in the forbidden region the solution is a linear combination of two solutions

$$\psi = Ce^{\beta x} + De^{-\beta x}.$$

Analytically we are able to set one of the coefficients to 0, but numerically we do not have this ability. Thus, at sufficiently large x, one of the exponentials will still end up dominating and ruin the solution.

The solution to this is to integrate along the direction of the physical solution. In other words, we want to integrate from region I to III but stop, and likewise from III to I. This way, the non-physical solution does not end up dominating the physical solution. Now, with two solutions, they have to satisfy the continuity conditions at the point where they meet, x_m:

$$\begin{aligned}\psi_{\text{left}}(x_m) &= \psi_{\text{right}}(x_m)\\ \psi'_{\text{left}}(x_m) &= \psi'_{\text{right}}(x_m).\end{aligned}$$

We can combine them into

$$\frac{\psi'_{\text{left}}(x_m)}{\psi_{\text{left}}(x_m)} = \frac{\psi'_{\text{right}}(x_m)}{\psi_{\text{right}}(x_m)}.$$

We can replace the derivatives by the central difference approximation, and turn it into a root searching problem:

$$f(E) = \frac{[\psi_l(x_m+h) - \psi_l(x_m-h)] - [\psi_r(x_m+h) - \psi_r(x_m-h)]}{2h\psi(x_m)} = 0.$$

5.10.2 Quantum tunnelling

A potential occupies a region of space from 0 to a. A particle approaches from the left. The general solution outside the potential region is given by

$$\psi(x) = \begin{cases}\psi_1(x) = e^{ikx} + Ae^{-ikx}, & \text{if } x < 0\\ \psi_3(x) = Be^{ik(x-a)}, & \text{if } x > a\end{cases}$$

The boundary conditions are

$$\begin{aligned}\psi_2(0) &= \psi_1(0) = 1 + A & \psi_2(a) &= \psi_3(0) = B\\ \psi'_2(0) &= \psi'_1(a) = ik(1-A) & \psi'_2(a) &= \psi'_3(a) = ikB\end{aligned}$$

Define $y_1 = \psi_2$ and $y_2 = \psi_2'$. For a guess of A, looking at the boundary conditions, we see that in general we get two values of B. However if A was chosen correctly, they coincide and so $y_2(a) = iky_1(a)$. This means we can define a function $g(A) = |y_2(a) - iky_1(a)|^2$ (the square comes since the functions are complex valued), and minimize g.

6 Partial differential equations

6.1 Classes of PDEs

Consider a general linear homogeneous second order PDE in two variables x and t:

$$A\frac{\partial^2 u}{\partial x^2} + B\frac{\partial^2 u}{\partial x \partial t} + C\frac{\partial^2 u}{\partial t^2} + D\frac{\partial u}{\partial x} + E\frac{\partial u}{\partial t} + Fu = 0$$

where the coefficients are functions of x and t.

If

$$B^2(x_0, t_0) - 4A(x_0, t_0)C(x_0, t_0) > 0$$

at (x_0, t_0), then we say that the PDE is *hyperbolic* at the point (x_0, t_0). If the PDE is hyperbolic at all points in the domain of interest, then it is said to be a hyperbolic equation. An example is the one dimensional wave equation

$$\frac{\partial^2 u}{\partial t^2} - c^2\frac{\partial^2 u}{\partial x^2} = 0.$$

A hyperbolic equation generally describe time-dependent conservative physical processes that are not evolving towards a steady state. To have a solution, we need two initial conditions and two boundary conditions.

If

$$B^2(x_0, t_0) - 4A(x_0, t_0)C(x_0, t_0) = 0$$

at (x_0, t_0), then we say that the PDE is *parabolic* at the point (x_0, t_0). An example is the diffusion equation

$$\frac{\partial u}{\partial t} - k\frac{\partial^2 u}{\partial x^2} = 0.$$

Parabolic equations describe time-dependent dissipative physical processes that are evolving to a steady state. For a solution, we need one initial condition and two boundary conditions.

If

$$B^2(x_0, t_0) - 4A(x_0, t_0)C(x_0, t_0) < 0$$

at (x_0, t_0), then we say that the PDE is *elliptic* at the point (x_0, t_0). An example is the Laplace equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

or the Poisson equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y).$$

Elliptic equations describe processes that have already reached a steady state. Only boundary conditions are needed for a solution.

6.2 Wave equation

Example 6.1 (Vibrating string)**.** Consider an ideal string stretched between two supports. For a small string element between x and $x+\Delta x$, there are two forces acting on it, namely the tensions at both ends.

$$F_x = T(x+\Delta x)\cos\beta - T(x)\cos\alpha \qquad F_y = T(x+\Delta x)\sin\beta - T(x)\sin\alpha$$

Small angle approximation gives $\sin\theta \approx \tan\theta$ and $\cos\theta \approx 1$. First of all, we will assume that the string element only moves in the transverse direction. In other words, $F_x \approx T(x) - T(x-\Delta x) \approx 0$. Next, we note that the tangent of an angle is simply the slope at that point, so

$$F_y \approx -T(x)\frac{\partial u}{\partial x}\bigg|_x + T(x+\Delta x)\frac{\partial u}{\partial x}\bigg|_{x+\Delta x}$$

From our previous discussion, $T(x) \approx T(x-\Delta x)$ and also using a Taylor series expansion we get

$$\begin{aligned} F_y &\approx -T(x)\frac{\partial u}{\partial x}\bigg|_x + T(x)\frac{\partial u}{\partial x}\bigg|_{x+\Delta x} \\ &= -T(x)\frac{\partial u}{\partial x}\bigg|_x + T(x)\left[\frac{\partial u}{\partial x}\bigg|_x + \Delta x \frac{\partial}{\partial x}\frac{\partial u}{\partial x}\bigg|_x + \cdots\right] \\ &\approx T\Delta x \frac{\partial^2 u}{\partial x^2}. \end{aligned}$$

Let μ be the mass density of the string. Then Newton's laws gives

$$\begin{aligned} \mu\Delta x \frac{\partial^2 u}{\partial t^2} &= T\Delta x \frac{\partial^2 u}{\partial x^2} \\ \frac{\partial^2 u}{\partial t^2} &= c^2 \frac{\partial^2 u}{\partial x^2} \end{aligned}$$

where $c = \sqrt{\frac{T}{\mu}}$. ◊

6.2.1 Finite differences

Consider the one-dimensional wave equation derived in the previous section. In two dimensions, we impose an evenly spaced rectangular grid in space and time such that $x_i = ih$ and $t_j = j\tau$ for $i = 0, 1, \ldots, n_x$ and $j = 0, 1, \ldots, n_t$. This allows us to discretise the equation. The second derivatives can be approximated with the central difference approximation

$$\frac{u_i^{j+1} - 2u_i^j + u_i^{j-1}}{\tau^2} - c^2 \frac{u_{i+1}^j - 2u_i^j + u_{i-1}^j}{h^2} = 0.$$

where $u_i^j = u(x_i, t_j)$. Solving for u_i^{j+1}, we find

$$u_i^{j+1} = \left(\frac{\tau c}{h}\right)^2 \left(u_{i+1}^j + u_{i-1}^j\right) + 2\left(1 - \left(\frac{\tau c}{h}\right)^2\right) u_i^j - u_i^{j-1}.$$

This equation is not self starting. To solve this problem, we can make use of the initial values given. Using the central difference approximation, we can write the first derivative as

$$\frac{\partial u}{\partial t}\bigg|_{t=0} \approx \frac{u_i^1 - u_i^{-1}}{2\tau}$$

or after some rearrangement

$$u_i^{-1} = u_i^1 - 2\tau \frac{\partial u}{\partial t}\bigg|_{t=0}.$$

Substituting this back to the original equation gives for the first time step:

$$\begin{aligned} u_i^1 &= \left(\frac{\tau c}{h}\right)^2 \left(u_{i+1}^0 + u_{i-1}^0\right) + 2\left(1 - \left(\frac{\tau c}{h}\right)^2\right) u_i^0 - u_i^1 + 2\tau \frac{\partial u}{\partial t}\bigg|_{t=0} \\ &= \frac{\tau^2 c^2}{2h^2}\left(u_{i+1}^0 + u_{i-1}^0\right) + (1 - \frac{\tau^2 c^2}{h^2}) u_i^0 + \tau \frac{\partial u}{\partial t}\bigg|_{t=0} \end{aligned}$$

6.3 Schrödinger equation

6.3.1 Forward time centred space

Consider the Schrödinger equation in one dimension

$$i\hbar \frac{\partial \psi}{\partial t} = -\frac{\hbar^2}{2m}\frac{\partial^2 \psi}{\partial x^2} + V\psi.$$

We discretise time and space in increments of h and τ. Let $\psi_j^n = \psi(x_j, t_n)$. We use the forward Euler method for the left hand side, and the central difference approximation for the right hand side to get

$$i\hbar \frac{\psi_j^{n+1} - \psi_j^n}{\tau} = -\frac{\hbar^2}{2m}\frac{\psi_{j+1}^n + \psi_{j-1}^n - 2\psi_j^n}{h^2} + V_j \psi_j^n$$

where $V_j = V(x_j)$. Since the Hamiltonian is a linear operator, we can write

$$i\hbar \frac{\psi_j^{n+1} - \psi_j^n}{\tau} = \sum_{i=1}^{n} H_{jk}\psi_k^n$$

where

$$H_{jk} = -\frac{\hbar^2}{2m}\frac{\delta_{j+1,k} + \delta_{j-1,k} - 2\delta_{j,k}}{h^2} + V_j \delta_{j,k}.$$

Computing for each j we get a system of equations

$$\psi_j^{n+1} = \psi_j^n - \frac{i\tau}{\hbar}\sum_{k=1}^{n} H_{jk}\psi_k^n.$$

In matrix notation,

$$\boldsymbol{\psi}^{n+1} = (\mathbf{I} - \frac{i\tau}{\hbar}\mathbf{H})\boldsymbol{\psi}^n.$$

This is the explicit forward time centred space scheme for solving the one dimensional Schrödinger equation.

However, this might not always be numerically stable. From the above, we see that eigenvalues take the form of $1 - \frac{i\tau}{\hbar}\lambda$. The complex magnitude is strictly larger than 1, since the eigenvalues λ of the Hamiltonian are all real. Therefore the scheme is unstable.

We can try switching to the backward Euler method, and after the same manipulations we get

$$\boldsymbol{\psi}^{n+1} = \boldsymbol{\psi}^{n} - \frac{i\tau}{\hbar}\mathbf{H}\boldsymbol{\psi}^{n+1}.$$

The eigenvalues take a form of $(1 + \frac{i\tau}{\hbar}\lambda)^{-1}$. The complex magnitude is smaller than 1, so this method is unconditionally stable.

6.4 Crank-Nicolson

A more accurate and stable scheme uses the Crank-Nicolson method. Recall that for the ODE $y' = f$, the Crank-Nicolson method gives

$$y_{n+1} = y_n + \frac{h}{2}(f_n + f_{n+1}).$$

The Schrödinger equation is already in this form, $\frac{\partial\psi}{\partial t} = -\frac{i}{\hbar}\mathcal{H}\psi$. Thus,

$$\boldsymbol{\psi}^{n+1} = \boldsymbol{\psi}^{n} - \frac{i\tau}{2\hbar}\mathbf{H}(\boldsymbol{\psi}^{n} + \boldsymbol{\psi}^{n+1}).$$

Rearranging, we have

$$\boldsymbol{\psi}^{n+1} = \frac{\left(\mathbf{I} - \frac{i\tau}{2\hbar}\mathbf{H}\right)}{\left(\mathbf{I} + \frac{i\tau}{2\hbar}\mathbf{H}\right)}\boldsymbol{\psi}^{n}$$

6.5 Laplace equation

Consider the heat equation

$$\nabla^2 u = \frac{1}{k}\frac{\partial u}{\partial t}$$

where $k = \frac{K}{\sigma\rho}$, K is the conductivity, σ the specific heat, ρ the density, and u the temperature. If we are looking for a steady state distribution across a thin plate then this reduces to the Laplace equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0.$$

Discretise the plate by imposing a rectangular grid such that $x_i = ih_x$ and $y_j = jh_y$. We will solve this using the Gauss-Seidel scheme. Using the central difference approximation we have

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h_x^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h_y^2} = 0.$$

where $u_{i,j} = u(x_i, y_j)$. Solving for $u_{i,j}$, we have

$$u_{i,j} = \frac{h_x^2 h_y^2}{2h_x^2 + 2h_y^2}\left(\frac{u_{i+1,j} + u_{i-1,j}}{h_x^2} + \frac{u_{i,j+1} + u_{i,j-1}}{h_y^2}\right).$$

Setting $h_x = h_y$ makes the equation much simpler,

$$u_{i,j} = \frac{1}{4}(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1}),$$

which is just the average of its surrounding neighbours.

We can also solve the heat equation at a non steady state, as a function of time.

It may happen that the physical system does not match the grid. For a disc, we could try polar coordinates. But for an arbitrarily shaped plate, we might not be able to fit it perfectly onto any sensible grid. Take for example the case where all the neighbours of (i, j) lie on the grid except for $(i + 1, j)$. Further assume that the boundary of the plate intersects the grid at $(x_i + \varepsilon, y_j)$. The Taylor expansion about x_i gives

$$\begin{aligned} f(x_i + \varepsilon) &= f(x_i) + \varepsilon f'(x_i) + \frac{\varepsilon^2}{2} f''(x_i) + \frac{\varepsilon^3}{3!} f'''(x_i) + \cdots \\ f(x_i - h) &= f(x_i) - h f'(x_i) + \frac{h^2}{2} f''(x_i) - \frac{h^3}{3!} f'''(x_i) + \cdots \end{aligned}$$

Combining these two equations give

$$f''(x_i) = 2 \cdot \frac{h f(x_i + \varepsilon) - (h + \varepsilon) f(x_i) + \varepsilon f(x_i - h)}{h\varepsilon(h + \varepsilon)} + \frac{h - \varepsilon}{3} f'''(x_i) + \cdots .$$

This expression can then be used, for example in the central difference approximation scheme above:

$$2 \cdot \frac{h u(x_i + \varepsilon, y_i) - (h + \varepsilon) u_{i,j} + \varepsilon u_{i-1,j}}{h\varepsilon(h + \varepsilon)} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2} = 0.$$

Some simplification gives

$$u_{i,j} = \frac{\varepsilon}{2(h + \varepsilon)} \left(\frac{2h^2}{\varepsilon(h + \varepsilon)} u(x_i + \varepsilon, y_i) + \frac{2h}{h + \varepsilon} u_{i-1,j} + u_{i,j+1} + u_{i,j-1} \right).$$

6.6 Von Neumann stability

Consider a wave $y(x, t) = A(t)e^{ikx}$. In the discretised form, we have

$$y_{j,n} = A^n e^{ikjh}$$

where $x_j = jh$, $t_n = n\tau$, and $y_{j,n} = y(x_j, t_n)$. Advancing the solution by one step gives

$$y_{j,n+1} = A^{n+1} e^{ikjh} = \xi y_{j,n}$$

where ξ is called the *amplification factor.* The strategy here is to insert the trial solution (i.e. $y_{j,n}$) into a numerical scheme and solve for the amplification factor. If $|\xi| > 1$, then the scheme is said to be *unstable.*

Example 6.2. Consider the advection equation $\frac{\partial y}{\partial t} = -c\frac{\partial y}{\partial x}$. Using finite time central space, we have

$$\frac{y_{j,n+1} - y_{j,n}}{\tau} = -c \frac{y_{j+1,n} - y_{j-1,n}}{2h}$$

Solving for $y_{j,n+1}$,

$$y_{j,n+1} = y_{j,n} - \frac{c\tau}{2h}(y_{j+1,n} - y_{j-1,n}).$$

Using the same trial solution, substitution gives

$$\xi A^n e^{ikjh} = A^n e^{ikjh} - \frac{c\tau}{2h}(A^n e^{ik(j+1)h} - A^n e^{ik(j-1)h}.$$

and it is easy to solve for ξ:

$$\xi = 1 - i\frac{c\tau}{h}\sin(kh).$$

For non-zero k we have $|\xi| > 1$, so the solution is unstable. $\Diamond$

Example 6.3. The Lax scheme is where we replace the $y_{j,n}$ term with the average of its neighbours. For the above example, this gives us

$$y_{j,n+1} = \frac{1}{2}(y_{j+1,n} + y_{j-1,n}) - \frac{c\tau}{2h}(y_{j+1,n} - y_{j-1,n}).$$

Inserting the same trial solution and solving we get

$$\xi = \cos(kh) - i\frac{c\tau}{h}\sin(kh).$$

$|\xi| \leq 1$ iff $\left|\frac{c\tau}{h}\right| \leq 1$. $\Diamond$

6.7 Neumann boundary conditions

Neumann boundary conditions, simply put, give us the values of first derivatives on the boundary. For example, consider a square metal plate and we supply heat to one edge, and heat flows out of the other edges.

Consider a point on the left edge of the plate. Using the central difference approximation on the boundary condition gives

$$\left.\frac{\partial u}{\partial x}\right|_{x=0} = A_j \approx \frac{u_{i,j} - u_{-1,j}}{2h_x}.$$

We already have an expression for the discretised steady state heat equation in an earlier section. We substitute this in, and solve for $u_{0,j}$ to get

$$u_{0,j} = \frac{h_x^2 h_y^2}{2h_x^2 + 2h_y^2}\left(\frac{2u_{1,j} - 2h_x A_j}{h_x^2} + \frac{u_{0,j+1} + u_{0,j-1}}{h_y^2}\right).$$

For $h_x = h_y = h$

$$u_{0,j} = \frac{2u_{1,j} - 2hA_j + u_{0,j+1} + u_{0,j-1}}{4}$$

We can do the same thing for the other sides. The last thing to consider are the corners. Consider $u_{0,0}$. We have two boundary conditions

$$\left.\frac{\partial u}{\partial x}\right|_{x=0} = A_j \approx \frac{u_{1,j} - u_{-1,j}}{2h}$$
$$\left.\frac{\partial u}{\partial y}\right|_{y=0} = C_i \approx \frac{u_{i,1} - u_{i,-1}}{2h}.$$

Thus, substituting into the discretised Laplace equation gives

$$\begin{aligned} u_{0,0} &= \frac{1}{4}(u_{-1,0} + u_{0,-1} + u_{1,0} + u_{0,1}) \\ &= \frac{1}{2}(u_{1,0} - hA_0 + u_{0,1} - hC_0). \end{aligned}$$