PHY 325 Notes

Computational Physics

Jaeden Bardati

Last modified January 31, 2022

1 Numerical Evaluation of Derivatives

Let $y = f(x)$, then the **derivative** is defined as

$$\frac{\mathrm{d}f}{\mathrm{d}x} \equiv \lim_{h\to 0} \frac{f(x+h) - f(x)}{h} \tag{1}$$

Now consider $y = f(x_1, ..., x_n)$, then the **partial derivative** is defined as

$$\frac{\partial f}{\partial x_i} \equiv \lim_{h\to 0} \frac{f(x_1, ..., x_i + h, ..., x_n) - f(x_1, ..., x_n)}{h} \tag{2}$$

We can numerically compute derivatives in three main ways: forwards, backwards and with an average of the two.

The **forward** derivative is defined as

$$\frac{\mathrm{d}f}{\mathrm{d}x} \approx \frac{f(x+h) - f(x)}{h} \tag{3}$$

The **backward** derivative is defined as

$$\frac{\mathrm{d}f}{\mathrm{d}x} \approx \frac{f(x-h) + f(x)}{h} \tag{4}$$

The **central** derivative is defined as

$$\frac{\mathrm{d}f}{\mathrm{d}x} \approx \frac{f(x+h) - f(x-h)}{2h} \tag{5}$$

The smaller n is, the more accurate this approximation is.

Note that, while the forward and backward derivatives take the same amount of time to compute, the central derivative has twice the number of calculations to do and so will take longer to run.

Furthermore, the central derivative can have divergent behaviour. To illustrate this, we can take the central derivative at the point (0, 0) on the absolute value function (see figure 1). Notice that the value of the derivative is 0 here, which could lead to undesired effects in the simulations of such systems.

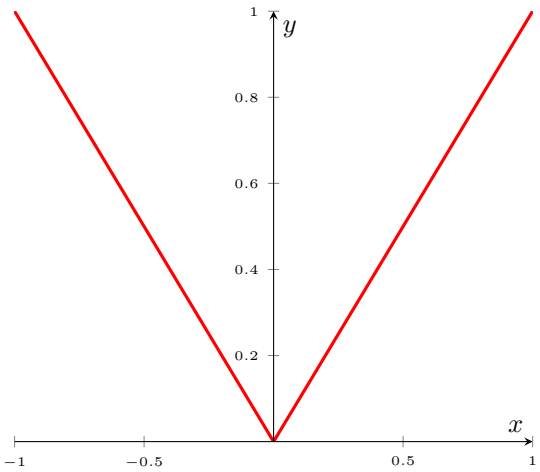

Figure 1: Absolute value function

2 Error in Derivative Calculation

When implementing a numerical derivative calculation, we would like to use a small value of h. But how small? One might think that we should try to make h as small as it can be. Of course, zero will not work since it would result in an error due to division by 0 (see equations 3, 4, and 5).

If we plot h by the error in the derivative, we will find that the derivative is inaccurate at both high and low values for h. There is a "sweet spot" in the middle (roughly around 10^{-8}) that minimizes the error. The error at low values of h is called **round-off error** and at high values of h is called **truncation error**.

2.1 Round-off Error

To understand this error, we must first ask: How are numbers represented by a computer? Specifically, we are interested in floats. The form is:

$$s \times M \times B^{e-E} \tag{6}$$

where s is the sign (0 if number is positive, 1 if negative), M is the Mantissa, B is the base, E is the bias, and e is the exponent.

This form is similar to scientific notation in principle. Namely, it is much more space-efficient to write 1.2×10^{-8} rather than 0.000000012. Note that here, 1.2 is the Mantissa, 10 is the base, the sign is 0, and $e - E = -8$.

For example, we will look at how 10.75 is stored. In binary, $(10)_{10} = (1010)_2$ and $(0.75)_{10} = (11)_2$, where the subscript indicates the base). So,

$$(10.75)_{10} = (1010.11)_2 = (1.01011)_2 \times 2^3$$

The bias term E is highly dependent on the particular machine you use. However, for a typical 32-bit computer, it is common to have a bias of $E = 2^{n-1} + 1$ with $n = 8$. This gives that $E = 129$. The base is, of course, $B = 2$.

Calculating e now, we know

$$e - E = 3 \implies e = 3 + E = 3 + 129 = 132$$

In binary, $(132)_{10} = (10000100)_2$.

Therefore, we would store the float value of 10.75 as

0	10000100	10101100000...000
s	$e - E$	M (23-bits)

In a 32-bit system, 1 bit is used to store the sign s, 8 bits for the exponent $e - E$, and 23 bits for the mantissa M. In a 64-bit system (double precision), 1 bit is used to store the sign s, 11 bits for the exponent $e - E$, and 52 bits for the mantissa M.

Since the mantissa is only a certain size, once a decimal value becomes lower than what can be stored in the mantissa, the value is lost. This is round-off error. The machine accuracy for a 32-bit system is around 10^{-8} and for a 64-bit system is 10^{-16}.

It is important when analyzing numerical systems to pick normalized units (close to unity). That is to say, choose units such that the numbers that are being used do not have this round-off error effect.

2.2 Truncation Error

Truncation error has to do with the sort of "rate of error decreasing with respect to h." Namely, if we Taylor expand $f(x+h)$, we obtain

$$f(x+h) = f(x) + hf'(x) + \frac{1}{2}h^2 f''(x) + \frac{1}{6}h^3 f'''(x) + ...$$

So, the calculation we use for the forward derivative (equation 3) is

$$\begin{aligned} & \frac{f(x+h)-f(x)}{h} = f'(x) + \frac{1}{2}hf''(x) + \frac{1}{6}h^2 f'''(x) + ... \\ \implies & \frac{f(x+h)-f(x)}{h} \approx f'(x) + O(h) \end{aligned}$$

We can see that the forward derivative is only accurate to the first order of h. This is why there is an increase in error when h is large. It is called **truncation error** since we truncate the infinite series when approximating.

3 Example: Projectile Motion

Consider a baseball thrown with air resistance. The equations of motion are

$$\frac{\mathrm{d}\vec{v}}{\mathrm{d}t} = \frac{1}{m}\vec{F}_{\text{air}}(v) - g\hat{y} \tag{7}$$

$$\frac{\mathrm{d}\vec{r}}{\mathrm{d}t} = \vec{v} \tag{8}$$

where $\vec{v}$ is the velocity, t is the time elapsed, m is the mass of the baseball, g is the gravitational acceleration, $\hat{y}$ is the upward direction, and $\vec{r}$ is the position vector. Note that the force of air friction here is

$$\vec{F}_{\text{air}}(v) = -\frac{1}{2}C_d\rho A|\vec{v}|\vec{v} \tag{9}$$

where C_d is the coefficient of air friction, ρ is the density of air, and A the area of the baseball (perpendicular to its direction of travel).

3.1 Euler Method

The Euler method is a numerical procedure for solve initial value problems of ordinary differential equations. Namely, if we have the form

$$\frac{\mathrm{d}\vec{v}}{\mathrm{d}t} = \vec{a}(\vec{r},\vec{v})$$

$$\frac{\mathrm{d}\vec{r}}{\mathrm{d}t} = \vec{v}$$

where $\vec{a}(\vec{r}, \vec{v})$ is the acceleration vector as a function of the position and velocity vectors.

Using the forward derivative where $\tau = h$ represents an increment in the time, then

$$\frac{\vec{v}(t+\tau) - \vec{v}(t)}{\tau} = \vec{a}(\vec{r}, \vec{v})$$
$$\frac{\vec{r}(t+\tau) - \vec{r}(t)}{\tau} = \vec{v}(t)$$

So,

$$\vec{v}(t+\tau) = \tau\vec{a}(\vec{r}, \vec{v}) + \vec{v}(t)$$
$$\vec{r}(t+\tau) = \tau\vec{v}(t) + \vec{r}(t)$$

We can write this as an iterative process

$$\vec{v}_{n+1} = \tau\vec{a}(\vec{r}_n, \vec{v}_n) + \vec{v}_n$$
$$\vec{r}_{n+1} = \tau\vec{v}_n + \vec{r}_n$$

This is what we use for the Euler method. Concretely, the steps for this method are

1. Specify the initial values $\vec{r}_1, \vec{v}_1$ at $t = 0$
2. Choose a time step τ
3. Calculate $\vec{a}$, given the current $\vec{r}$ and $\vec{v}$
4. Compute the new $\vec{v}_{i+1}$ and $\vec{r}_{i+1}$
5. Go to step 3

4 Example: Simple Pendulum

The equations of motion for the simple pendulum is

$$\frac{\mathrm{d}^2\theta}{\mathrm{d}t^2} = -\frac{g}{L}\sin\theta \tag{10}$$

For small angles $\theta \ll 1$, we have

$$\frac{\mathrm{d}^2\theta}{\mathrm{d}t^2} = -\frac{g}{L}\theta$$

The solutions of this approximation are

$$\theta(t) = C_1 \cos\left(\frac{2\pi t}{T_s} + C_2\right)$$

For arbitrary constants C_1 and C_2 determined by the initial conditions and where $T_s = \sqrt{\frac{L}{g}}$.

However, if we are interested in the behaviour of the pendulum in regions where the angle is not small, we must result to numerical approximation. If we wanted to use the Euler method here, we would split the second-order ODE into two first-order ODEs:

$$\frac{\mathrm{d}\omega}{\mathrm{d}t} = \alpha(\theta)$$
$$\frac{\mathrm{d}\theta}{\mathrm{d}t} = \omega$$

where $\alpha(\theta) = -\frac{g}{L} sin\Theta$. The Euler method would therefore be done using

$$\theta_{n+1} = \Theta_n + \tau\omega_n$$
$$\omega_{n+1} = \omega_n + \tau\alpha(\Theta_n)$$

However, because the Euler method is only accurate to the first-order, this implementation diverges to infinity in error very quickly. Instead, we must look to a new method.

4.1 Central Derivative Truncation Error

For the new scheme, we must understand the truncation error of the central derivative. We will start by Taylor expanding $f(t+\tau)$ and $f(t-\tau)$

$$f(t+\tau) = f(t) + \tau f'(t) + \frac{1}{2}\tau^2 f''(t) + \frac{1}{6}\tau^3 f'''(t) + ...$$
$$f(t-\tau) = f(t) - \tau f'(t) + \frac{1}{2}\tau^2 f''(t) - \frac{1}{6}\tau^3 f'''(t) + ...$$

So, using the formula for the central derivative (equation 5),

$$\frac{f(t+\tau) - f(t-\tau)}{2\tau} = f'(t) - \frac{1}{6}\tau^2 f'''(t) + ...$$
$$\implies \frac{f(t+\tau) - f(t-\tau)}{2\tau} \approx f'(t) + O(\tau^2)$$

The central derivative is therefore accurate to the second order of τ. Note that we can also find the second derivative by adding the Taylor expanded series instead of subtracting them.

4.2 Leap Frog Method

We will start with the general equations of motion where **the acceleration does not depend on velocity**. This is different from the Euler method, which allowed accelerations as a function of velocity.

$$\frac{\mathrm{d}\vec{v}}{\mathrm{d}t} = \vec{a}(\vec{r}(t))$$
$$\frac{\mathrm{d}\vec{r}}{\mathrm{d}t} = \vec{v}$$

where $\vec{a}(\vec{r}, \vec{v})$ is the acceleration vector as a function of the position and velocity vectors.

This time, we will evaluate using the central derivative instead of the forward one. We will evaluate the velocity at $t + \tau$ and $t - \tau$, and the position at $t + \tau$ and $t + 2\tau$,

$$\frac{\vec{v}(t+\tau) - \vec{v}(t-\tau)}{2\tau} + O(\tau^2) = \vec{a}(\vec{r}(t))$$
$$\frac{\vec{r}(t+2\tau) - \vec{r}(t)}{2\tau} + O(\tau^2) = \vec{v}(t+\tau)$$

We can rewrite this as

$$\frac{\vec{v}_{n+1} - \vec{v}_{n-1}}{2\tau} + O(\tau^2) = \vec{a}(\vec{r}_n)$$
$$\frac{\vec{r}_{n+2} - \vec{r}_n}{2\tau} + O(\tau^2) = \vec{v}_{n+1}$$

Therefore,

$$\vec{v}_{n+1} = \vec{v}_{n-1} + 2\tau\vec{a}(\vec{r}_n) + O(\tau^3)$$
$$\vec{r}_{n+1} = \vec{r}_n + 2\tau\vec{v}_{n+1} + O(\tau^3)$$

This is the **leap-frog method**. The solution is advanced in n steps of 2τ. Moreover, the position is evaluated at $\vec{r}_1$, $\vec{r}_3$, $\vec{r}_5$, etc., and the velocity is evaluated at $\vec{v}_2$, $\vec{v}_4$, $\vec{v}_6$, etc., hence the leap-frog.

4.3 Verlet Method

Now taking the 1st and 2nd derivatives, consider

$$\frac{\mathrm{d}\vec{r}}{\mathrm{d}t} = \vec{v}(t) \quad \frac{\mathrm{d}^2\vec{r}}{\mathrm{d}t^2} = \vec{a}(t)$$

Then,

$$\frac{\vec{r}_{n+1} - \vec{r}_{n-1}}{2\tau} + O(\tau^2) = \vec{v}_n$$
$$\frac{\vec{r}_{n+1} + \vec{r}_{n-1} - 2\vec{r}_n}{\tau^2} + O(\tau^2) = \vec{a}_n$$

Therefore,

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

Therefore, if we do not need the velocity, we can have accuracy to the 4-th order.

Note that both the leap-frog and the verlet methods are not self-starting. In other words, you need to use another method to get the first step or two to work.

4.4 Euler-Cromer Method

We can make an improvement to the regular Euler method without doing too much. Namely, we first compute the velocity of the current iteration and then use the current velocity to find the current position. Namely, instead of

$$\vec{r}_{n+1} = \vec{r}_n + \tau\vec{v}_n$$
$$\vec{v}_{n+1} = \vec{v}_n + \tau\vec{a}(\vec{r}_n, \vec{v}_n)$$

which is the Euler method, we do

$$\vec{v}_{n+1} = \vec{v}_n + \tau\vec{a}(\vec{r}_n, \vec{v}_n)$$
$$\vec{r}_{n+1} = \vec{r}_n + \tau\vec{v}_{n+1}$$

This is the Euler-Cromer method.

5 Integration of Ordinary Differential Equations

Consider the general equation

$$\frac{\mathrm{d}^2 y}{\mathrm{d}x^2} + q(x)\frac{\mathrm{d}y}{\mathrm{d}x} = r(x)$$

ODEs can always be written as sets of first order differential equations. Here, we can do this by substitution,

$$\frac{\mathrm{d}y}{\mathrm{d}x} = z(x)$$
$$\frac{\mathrm{d}z}{\mathrm{d}x} = r(x) - q(x)z(x)$$

The general problem in ODEs is thus reduced to the study of N coupled first-order differential equations.

$$\frac{\mathrm{d}y_i}{\mathrm{d}x} = f_i(x, y_1, ..., y_N)$$

where $i = 1, ..., N$.

We always need boundary conditions. These can be in the form of

- Initial boundary conditions
  - All y_i are given at some starting rate of xs (start)
  - Need to find y_i at x_f (finish)
- Two point boundary value problem
  - Conditions are specified at more than one x
  - Typically at x_f, x_s

For example, when modeling the Sun, we set the boundary conditions of the temperature, luminosity, mass and pressure at the edge. Then, we integrate inwards to find those properties in the center.

5.1 General Strategy

The general stategy for attacking these types of problems is to rewrite the dy's and dx's as Δx's and Δy's, and then multiply by Δx. The literal interpretation is the Euler method. Though, in general one should not use the Euler method.

There are two very common general methods that one can try:

- Runge-Kutta
- Bulinsch-Stoer (extrapolation method)

5.2 Example: Kepler Orbit

Ou running example will be a Kepler problem of a small object in orbit around the Run (e.g. a comet). The gravitational force is

$$\vec{F} = \frac{-GmM}{|\vec{r}|}\vec{r}$$

where $\vec{r}$ is the position of the comet, m is the mass of the comet, M is the mass of the Sun, G is the gravitational constant.

The natural units for this problem are in AU, years and $M_\odot$. So,

$$GM = \frac{4\pi^2 \text{AU}^3}{\text{years}^2}$$

We want to trace something to make sure the behaviour is correct. We will track the total energy:

$$E = \frac{1}{2}mv^2 - \frac{GMm}{r}$$

We could now implement the Euler or Euler-Cromer methods if we want, however they will not be very accurate. Instead, we will look at a new, more accurate method.

5.3 Runge-Kutta Method

The formula for the Euler method is

$$y_{n+1} = y_n + hf(x_n, y_n)$$

which advances the solution from x_n to x_{n+1}.

- The Euler method is $\text{O}(n^2)$
- Only uses derivative information at the beginning of the interval
- Euler is not accurate and not stable

Consider instead the use of Euler to make a trial step to the mid-point and use the midpoint to advance the next step.

$$\begin{aligned} k_1 &= hf(x_n, y_n) \\ k_2 &= hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1) \\ y_{n+1} &= y_n + k_2 + O(h^3) \end{aligned}$$

This is called **2nd order Runge-Kutta** or the **"midpoint" method**.

5.4 RK4 Method

The 4th order Runge-Kutta Method (RK4) is very commonly used. It goes as the following.

$$
\begin{aligned}
k_1 &= hf(x_n, y_n) \\
k_2 &= hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1) \\
k_3 &= hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2) \\
k_4 &= hf(x_n + h, y_n + k_3) \\
y_{n+1} &= y_n + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4 + O(h^5)
\end{aligned}
$$

The idea is to take a few intermediate steps to determine the next y to return.

Now that we have a general method to solve this problems, we need to find a way of setting the size of h.

5.5 Adaptive RK4 Method

Let us use the example of an elliptical orbit. In the perihelion (closest to the center body), we would like h to be small so that it is more accurate, since the orbiting body moves fast there. However, we could allow h to be larger in aphelion (furthest to the center body), where the orbiting body moves slower.

The way we will do this is with adapting h on the fly. We will do this by comparing the relative error of a big step $y_b(t+h)$ to a couple small steps $y_s(t+h)$ (result from two steps of $t + \frac{1}{2}h$).

We will compare y_b with y_s to understand the *local truncation error*.If the error is tolerable, then the step is accepted, and a larger value of h is used for the next step.

How can we code this Adaptive RK4 method?

5.5.1 Code Outline

- Loop over maximum number of attempts to satisfy error bound (user set)
  - Take two small steps
  - Take one large step
  - Compute truncation error

  - Estimate h
  - If acceptable, return updated solution

- Calculate $\Delta_1 = y_s - y_b$
- Calculate $Delta_0 = \text{err} \times \frac{1}{2}(|y_s| + |y_b|)$
- Calculate $\Delta_{\text{ratio}} = \left|\frac{\Delta_1}{\Delta_0 + \text{eps}}\right|$
- Estimate new h value: $h_{\text{new}} = h(\Delta_{\text{ratio}})^{-\frac{1}{5}}$

We need to be careful with h_{new} since this is a linear interpolation. So, we will correct it

$$h_{\text{new}} = S_1 \times h_{\text{new}}$$

where $S_1 < 1$, and typically, $S_1 \approx 0.9$. Also,

$$h_{\text{new}} = \max(h_{\text{new}}, \frac{h}{S_2})$$
$$h_{\text{new}} = \max(h_{\text{new}}, S_2 h)$$

where $S_2 > 1$, and typically, $S_2 \approx 4$.

We then check if this actually worked. If $\Delta_{\text{ratio}} < 1$, then we are done. Otherwise, we use h_{new} as our h and try again.

5.6 Current Method

The currently accepted method is the **"embedded" solution**. The 5th order Runge Kutta (RK5) is

$$\begin{aligned} k_1 &= hf(x_n, y_n) \\ k_2 &= hf(x_n + a_2 h, y_n + b_{21} k_1) \\ &... \\ k_6 &= hf(x_n + a_6 h, y_n + b_{61} k_1 + ... + b_{65} k_5)) \\ y_{n+1} &= y_n + c_1 k_1 + c_2 k_2 + ... + c_6 k_6 \end{aligned}$$

This is typically known as the **Runge-Kutta-Fehlberg methods**. The $a_2, a_3, ..., a_6$, $b_{21}, ..., b_{61}, ..., b_{65}$, and $c_1, ..., c_6$ are constants. There are tabular values of the coefficients, such as "cash-karp".

5.7 The Lorentz Model

The Lorentz model is

$$\begin{aligned}\frac{\mathrm{d}x}{\mathrm{d}t} &= \sigma(y-x)\\ \frac{\mathrm{d}y}{\mathrm{d}t} &= rx-y-xz\\ \frac{\mathrm{d}z}{\mathrm{d}t} &= xy-bz\end{aligned}$$

where r, σ, and b are constants.

This model demonstrates an example of chaos. That is to say, it is highly sensitive to a small change in the initial conditions if you iterate far enough in the future.

6 Solving Systems of Equations

Consider a general system of equations

$$\begin{gathered}a_{11}x_1 + a_{12}x_2 + \ldots + a_{1N}x_N = b_1,\\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2N}x_N = b_2,\\ \vdots\\ a_{M1}x_1 + a_{M2}x_2 + \ldots + a_{MN}x_N = b_N\end{gathered}$$

There are N unknowns: x_j for $j = 1, 2, ..., N$, related by M equations.

The coefficients a_{ij} with $i = 1, 2, ..., M$ and $j = 1, 2, ..., N$
The right had side quantities b_i for $i = 1, 2, ...M$.

If $M = N$,

- There is a good change of solving for x_j's.
- Unless one of the equaiton is linear, combination of the others
- Singular matrix

There are numerical considerations,

- Round off error may make the system singular
- Accumulated round-off error
  - important for large systems
  - the closer the matrix is to singular, the more problematic

For double precision, simple inversion with $N \leq 100$ is likely safe with any method.

There are many sophisticated packages that will detect and correct for singular issues (e.g. LSODE from ODEPACK). Some common software packages for this are

- LINPACK
  - Analyzes and solves linear least squares
  - Designed for supercomputers in the 70s and 80s
  - Largely superseded by LAPACK
- LAPACK
  - Solves systems of equations, eigenvalues problems and singular value problems
  - Also does LU decomposition, Cholesky, QR, SUD, etc.

Many compilers and software packages will include optimized versions of ODEPACK or LAPACK. These are all free and open-source.

When solving these equations, we should consider:

- Is the matrix only composed of positive numbers?
- Is the matrix equal to its own conjugate transpose?
- Is the matrix sparse (lots of zeros)?
- Is the matrix close to singular?

In these cases, there cna be significant increases in performance. Picking the right routine could make a algorithm that runs as $O(n^3)$ to $O(n \log n)$.

Our linear system can be written as

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1N} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2N} \\ & \dots\dots\dots\dots & & & \\ a_{M1} & a_{M2} & a_{M3} & \dots & a_{MN} \end{bmatrix} \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ . \\ b_N \end{bmatrix}$$

This gives

$$A\boldsymbol{x} = \boldsymbol{b}$$

where we would like to solve for $\boldsymbol{x}$.

We have M rows and N columns.

If $M < N$, there is no solution.
If $M > N$, the system is over determined. This happens frequently. The typical case is wanting a solution to satisfying all equations (e.g. data fitting).

6.1 Gauss Elimination

Consider the system

$$\begin{aligned} x_1 + x_2 \ + x_3 &= 6 \\ -x_1 + 2x_2 \quad &= 3 \\ 2x_1 \quad + x_3 &= 5 \end{aligned}$$

Adding the first equation to the second and subtracting the first multiplied by 2 from the third gives

$$\begin{aligned} x_1 + x_2 \ + x_3 &= 6 \\ 3x_2 \quad + x_3 &= 9 \\ -\ 2x_2 + x_3 &= -7 \end{aligned}$$

Now, multiplying the second equation by $-\frac{2}{3}$ and subtracting the third gives

$$\begin{aligned} x_1 + x_2 \ + x_3 &= 6 \\ 3x_2 \quad + x_3 &= 9 \\ -\frac{1}{3}x_3 &= -1 \end{aligned}$$

This is known as forward elimination and then using backward substitution to solve for $x_1, x_2, ...$ This is a $O(n^3)$ routine.

Note that you should always use **pivoting**. To illustrate the need for this, consider the set of equations

$$\begin{aligned} \epsilon x_1 + x_2 + x_3 &= 5 \\ x_1 + x_2 \quad &= 3 \\ x_1 \quad + x_3 &= 4 \end{aligned}$$

In the limit as $\epsilon \to 0$, the solution is $x_1 = 1, x_2 = 2, x_3 = 3$.

Using the forward elimination,

$$\begin{aligned}
\epsilon x_1 + x_2 + x_3 &= 5 \\
(1 - \frac{1}{\epsilon})x_2 + \frac{1}{\epsilon}x_3 &= 3 - \frac{5}{\epsilon} \\
-\frac{1}{\epsilon}x_2 + (1 - \frac{1}{\epsilon})x_3 &= (4 - \frac{5}{\epsilon})
\end{aligned}$$

In the limit as $\epsilon \to 0$, we have a problem as the term $\frac{1}{\epsilon}$ blows up.

For example, $C - \frac{1}{\epsilon} \approx \frac{1}{\epsilon}$ for small ϵ. So our system of equations becomes

$$\begin{aligned}
\epsilon x_1 + x_2 \quad + x_3 &= 5 \\
-\frac{1}{\epsilon}x_2 - \frac{1}{\epsilon}x_3 &= \frac{5}{\epsilon} \\
-\frac{1}{\epsilon}x_2 - \frac{1}{\epsilon}x_3 &= \frac{5}{\epsilon}
\end{aligned}$$

The last two equations cause the singular condition to arise, even though the original matrix is not singular.

The solution is to interchange the order of the equations for forward elimination. This is called **pivoting**. For example, in the case before, simply changing the order of the equations gives

$$\begin{aligned}
x_1 + x_2 \quad &= 3 \\
\epsilon x_1 + x_2 + x_3 &= 5 \\
x_1 \quad + x_3 &= 4
\end{aligned}$$

Which yields

$$\begin{aligned}
x_1 + x_2 \quad &= 3 \\
+ (1 - \epsilon)x_2 + x_3 &= 5 - 3\epsilon \\
- x_2 \quad + x_3 &= 4 - 3\epsilon
\end{aligned}$$

Simply picking the largest element as the **pivot** is a good chance.

6.2 LU decomposition

Every matrix A can be decomposed into a lower and upper diagonal form.

$$A = L \cdot U$$

where L is the lower diagonal and U is the upper diagonal. For example,

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} = \begin{bmatrix} \alpha_{11} & 0 & 0 & 0 \\ \alpha_{21} & \alpha_{22} & 0 & 0 \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & 0 \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} \end{bmatrix} \cdot \begin{bmatrix} \beta_{11} & \beta_{12} & \beta_{13} & \beta_{14} \\ 0 & \beta_{22} & \beta_{23} & \beta_{24} \\ 0 & 0 & \beta_{33} & \beta_{34} \\ 0 & 0 & 0 & \beta_{44} \end{bmatrix}$$

If we can get this upper and lower form, then

$$A \cdot \boldsymbol{x} = (L \cdot U) \cdot \boldsymbol{x} = L \cdot (U \cdot \boldsymbol{x}) = \boldsymbol{b}$$

The we can solve the linear set

$$L \cdot \boldsymbol{y} = \boldsymbol{b}$$

solving for $\boldsymbol{y}$ through forward substitution. And then

$$U \cdot \boldsymbol{x} = \boldsymbol{y}$$

solving through back substitution.

Note that this can also be used to find the determinant of a matrix faster also.