02-712 Week 3

Biological Modeling and Simulation

Aidan Jan

September 11, 2025

Linear Algebra Stuff

Consider we are modeling how an epidemic spreads. Let S represent the number of susceptible, I represent the number infected, and R represent the number recovered. We can maybe make the model like:

$$\begin{aligned}\frac{\mathrm{d}S}{\mathrm{d}t} &= -\beta I - d_S S \\ \frac{\mathrm{d}I}{\mathrm{d}t} &= \beta SI - d_I I \\ \frac{\mathrm{d}R}{\mathrm{d}t} &= d_R R\end{aligned}$$

In this case, we have a linear model, since none of the variables are exponential. This is a tradeoff of realism for simplicity, but if we say, made the model nonlinear to be more realistic, it may be too complicated to be useful. In this case, we can write this model as a matrix.

- Put S, I, and T in a vector.
- Basically,
$$\begin{bmatrix} S_{t+1} \\ I_{t+1} \\ T_{t+1} \end{bmatrix} = \begin{bmatrix} \\ A \\ \\ \end{bmatrix} \begin{bmatrix} S_t \\ I_t \\ T_t \end{bmatrix}$$
- Or, the number of people susceptible, infected, and recovered in the next time step is based on the current number of people susceptible, infected, and recovered, times some linear function, A.

The matrix A can be thought of as a transformation that is applied to one vector to turn it into another. Understanding the properties of A, even if it cannot be solved for, it may give insight on how the systems change. For example, it may give some insight about the speed diseases spread.

Properties of Matrices

- Determinant: Describes the area (in 2D) or volume (in 3D+) the vectors of the matrix encompasses. (This is why a determinant doesn't exist if one of the vectors is zero, or if the vectors aren't linearly independent.)
- Eigenvectors/Eigenvalues:
  - Eigenvectors are special (linearly independent) vectors that aren't affected by certain effects of A. For example, it will be immune to shears in certain directions.
  - The eigenvalues are the amount that the vector is scaled.

– A can be decomposed into a product with the eigenvectors and eigenvalues. If eigenvectors are $x_0, \ldots, x_n$ and eigenvalues are $\lambda_0, \ldots, \lambda_n$, then

$$A = X\text{diag}(\lambda)X^{-1}$$

where $X = \begin{bmatrix} x_1 \ldots x_n \end{bmatrix}$ and $\text{diag}(\lambda) = \begin{bmatrix} \lambda_1 & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & \lambda_n \end{bmatrix}$.

– The eigenvalues may show things like whether a disease is spreading (for example, $\lambda_I > 1$, where λ_I is the eigenvalue corresponding to the vector describing the number of people infected.), or shrinking ($0 < \lambda_I < 1$).

– The eigenvalues also have the property that if the $\text{diag}(\lambda)$ is raised to a power of t, it corresponds to repeated application of the entire matrix, allowing for easy simulation over longer periods of time.

Continuous Optimizations

Continuous optimization is optimization over real numbers, unlike integers we have been seeing so far.

- $F(x_1, \ldots, x_n) = F(\vec{x}), x \in \mathbb{R}$

An example of a continuous optimization problem is, consider a protein structure we want to create. We know its function and how it should fold, but how do we decide which animo acids to use? We can frame this as an optimization problem to pick amino acids to force the folding into a certain way.

- These problems started popping up a lot more often recently because deep learning tends to work very well for them.
- Functions in this domain of problems tend to be C∞ functions - that is, the function itself is continuous, and every derivative of the function, up to infinity, are all continuous.
- Typically, derivatives are bounded. e.g., $\frac{\mathrm{d}^t c}{\mathrm{d}x^t} < B$, for some number B.

Continuous optimizations tend to be very hard to solve; they are intractible. We make some assumptions to make it easier.

Lipshitz Continuity

One assumption we use is the Lipshitz Continuity. This basically means that the function in question has the property:

$$|F(\vec{x_1}) - F(\vec{x_2})| < K|\vec{x_1} - \vec{x_2}| \forall \vec{x_1}\vec{x_2}$$

Or basically, the function does not vary in the order of an exponential. They are well-behaved.

Local Optimization

Another assumption we use is that a lot of times, we assume the local minimum is good enough. The minimum we find may not necessarily be the global minimum, but locally it is the minimum.

1-D Optimization

A lot of times, we only consider one dimension of optimization at a time. This means that the minimum (or maximum) would be when the derivative is equal to zero. Optimizing a problem then becomes a problem of finding zeroes of a function.

Methods of Optimization

Bisection Method

Suppose we want to find a point where the function is zero. How do we do that?

- Since functions we work with are continuous, we first pick two points - one where the value is negative and one where the value is positive.
- In this case, it must be that at some point between the two points we picked is zero.
- We then bisect the interval (divide it in half), and check the two sides; one of the sides would contain the zero.
- We repeat the bisection until we get whatever precision we want for the zero.

The following is a pseudocode for the bisection method:

```
def Bisection(x_min, x_max, thres):
    f_min = f(x_min)
    f_max = f(x_max)
    while (f_max - f_min > thres):
        x_mid = (x_min + x_max) / 2
        f_mid = f(x_mid)
        if (f_mid == 0):
            return x_mid
        elif (f_mid > 0):
            x_max = x_mid
            f_max = f_mid
        else:
            x_min = x_mid
            f_min = f_mid
    return (x_min + x_max) / 2
```

We could add a feature to terminate early if `f_mid` is close enough to zero, but not quite. We can set a threshold that way too. But this is the general idea.

Example:

Consider $f(x) = x^2 - 2$, and interval $[0, 2]$.

iteration	x_{min}	f_{min}	x_{max}	f_{max}	x_{mid}	f_{mid}
0	0	-2	2	2	1	-1
1	1	-1	2	2	1.5	2.5
2	1	-1	1.5	0.25	1.25	-0.44
⋮			⋮			

With enough iterations, we would approach the true answer (which in this case we can calculate analytically, but usually can't), $\sqrt{2}$.

Secant Method

What if our zero is very close to one of our bounds? In this case, the bisection method is quite inefficient. Instead, we can use the secant method.

- Instead of taking the midpoint between x_{min} and x_{max}, we can draw a straight line from (x_{min}, f_{min}) to (x_{max}, f_{max}), and use the point where the straight line intersects with the x-axis as our guess instead.

- With simple high-school algebra, we can write our guess as:

$$x_{guess} = x_{min} - \frac{x_{max} - x_{min}}{f_{max} - f_{min}} \cdot f_{min}$$

- To convert the bisection method to the secant method, we just replace x_{mid} with x_{guess}.
- Note that this is a heuristic that probably sometimes works better than bisection. However, it is possible for it to perform worse.

Newton's Method (Newton-Raphson)

This is a method that tends to work way better than both the bisection and secant method. For this, we need to know the value of one point on the line (instead of an interval), as well as the slope of the line at that point (the derivative.)

- From here, use the slope of the derivative and draw the tangent line.
- Find where the tangent line intersects the x-axis, and that is the next guess.
- Repeat until convergence.

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

For our previous example with equation $f(x) = x^2 - 2$, we can find $f'(x) = 2x$. Therefore, our guess equation would be:

$$x_{i+1} = x_i - \frac{x_i^2 - 2}{2x_i}$$

We can choose to start $x_0 = 2$, and calculate iterations with this initial value.

- Notice that the derivative cannot be at a zero! (Otherwise the fraction blows up.)
- With a good initial guess, this will converge very quickly.
- With a bad initial guess, it is possible that this method diverges instead and fails to find a solution.

Proof of Newton's Method

If we can get the Taylor series of a function, like:

$$f(x) = f(x_1) + f'(x_1)(x - x_i) + \frac{1}{2}f''(\xi)(x - x_i)^2$$

Assume $f(x) = 0$, and $f'(x) \neq 0$. ξ is the greek character Xi, and it represents some constant that can approximate the rest of the taylor series.

Simplifying,

$$\begin{aligned}
\frac{f(x)}{f'(x)} + (x - x_i) + \frac{f''(\xi)}{2f'(x_i)}(x - x_i)^2 &= 0 \\
x - \left(x_i + \frac{f(x_i)}{f'(x_i)}\right) &= -\frac{f''(\xi)}{2f'(x_i)}(x - x_i)^2 \\
x - x_{i+1} &= -\frac{f''(\xi)}{2f'(x_i)}(x - x_i)^2 \\
x - x_{i+1} &= C(x - x_i)^2
\end{aligned}$$

We replaced the second derivative with the ξ term with a constant in the last step, and this shows why this method works well. Every iteration we take, our error from the exact value $(x - x_i)$, decreases quadratically every iteration, as long as none of the derivatives blow up.

Going back to Proteins

Consider the protein folding problem from before. We can imagine the function as a black box that takes in a series of amino acids, and outputs the error to how closely it folds to the target. We can then optimize this error.

- However, with a function like this, we don't have access to a derivative!
- Instead, we use a **numerical derivative**, or an estimate of a derivative.
  - Suppose we let the function be $f(x)$.
  - In this case, we can do a taylor expansion and write $f(x+\Delta x) = f(x) + \Delta x f'(x) + \frac{1}{2}(\Delta x)^2 f''(\xi)$, where Δx is a small number.
  - If we simplify, we eventually get to
    $$\frac{f(x+\Delta x) - f(x)}{\Delta x} = f'(x) + \frac{\Delta x}{2} f"(\xi)$$
    In this case, the left side is a good approximation of the derivative, we just want to minimize the $\frac{\Delta x}{2}$ term.
- Unfortunately, we can't make Δx too small since our computers cannot represent floating points well enough.
- Instead, if we need more accurate derivatives, we increase the order of our taylor series.
  $$f(x+\Delta x) = f(x) + \Delta x f'(x) + \frac{(\Delta x)^2}{2} f''(x) + \frac{(\Delta x)^3}{6} f'''(\xi)$$
  $$f(x-\Delta x) = f(x) - \Delta x f'(x) + \frac{(\Delta x)^2}{2} f''(x) - \frac{(\Delta x)^3}{6} f'''(\xi)$$
- Here is an example with expansion to order 3. We would perturb the two expansions in opposite directions, and subtract one from the other.
  $$f(x+\Delta x) - f(x-\Delta x) = 2\Delta x f'(x) + 2\frac{(\Delta x)^3}{6} f'''(\xi)$$
  Simplifying:
  $$\frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x} = f'(x) + \frac{(\Delta x)^2}{6} f'''(\xi)$$
  Now, our error goes down in the order of 2, which would shrink way faster than before.

Multidimensional Optimization

If we have two dimensions, instead of doing this with the derivative, we would do this with the gradient.

- Instances of $f'(x)$ are replaced with $\nabla f(x)$, and we want to solve when this is equal to zero, or $\nabla f(x) = \vec{0}$.
- Instances of $f''(x)$ are replaced with $H(f)$, where H is the Hessian, or the matrix of second derivatives. The Hessian is the Jacobian of the gradient.
- The multi-dimensional Newton-Raphson optimization is:
  $$\vec{x_{i+1}} = \vec{x_i} - H(\vec{x_i})^{-1} \nabla F(\vec{x_1})$$
- We are forced to do a matrix inversion since matrix division is not defined.
- However, matrix inversion sucks typically, and causes all sorts of problems. Instead, we would assume that
  $$H(\vec{x_i})\vec{y} = \nabla F(\vec{x_1})$$
  and solve for $\vec{y}$. $\vec{y}$ would be used in place of the inverse matrix.

Gradient Descent

Instead of calculating analytically, what the next point should be, we can estimate which direction to move by only using the gradient.

- Pick a starting point on the gradient.
- Move in the opposite direction of the gradient.
- Repeat until moving doesn't decrease loss anymore.

The benefit of gradient descent is that it is very robust. You can start off with a bad guess and still eventually make it to a local minimum. However, Newton-Raphson works faster.

Levenburg-Marquardt Method

This method takes inspiration from both Newton-Raphson and gradient descent:

$$\vec{(}x_{i+1}) = \vec{(}x_i) - (H(x_i) = \lambda I)^{-1} \nabla F(\vec{x_1})$$

Basically, we set λ based on some heuristic that estimates how close we are to the local minimum. When far away, this method works more like gradient descent, and when close, this method works like Newton-Raphson and speeds up.