Topic: ODEs

Nick Battista

Date Created: 5/21/2014
Date Modified: 7/20/16

1 BACKGROUND

It's no surprise that most ODEs cannot be solved analytically, even with the patience of a saint and a love of wasting paper. Luckily computers enjoying doing a lot of fast computations and arithmetic and we'll discuss how to work out a deal with them to find our desired solution to the ODE we're studying.

For the remainder of this section, we'll use the following notation,

$$\frac{dy}{dt} = f(t, y(t)),$$

with initial value, $y(0) = y_0$.

In general, $f(t, y(t))$ is any function, linear or nonlinear and the like, of both the independent variable, t, and dependent variable $y(t)$. Before we talk about numerical analysis of ODEs, we'll review a few quick tidbits from analysis, namely *existence* and *uniqueness* (Don't worry, they're useful and cute.)

1.1 EXISTENCE AND UNIQUENESS

Consider a region $R = [t_0-\alpha, t_0+\alpha]\times[y_0-\beta, y_0+\beta]$, for some $\alpha, \beta > 0$, and $M = \max_R(|f(t, y(t))|)$.

1.1.1 ANALYSIS: EXISTENCE

If $f(t, y(t))$ is continuous, $|f| \leq M$ in R, then $y(t)$ exists in a neighborhood of t_0, $|t - t_0| \leq \alpha$.

1.1.2 Analysis: Uniqueness

We have two different ways to look at uniqueness,

- If $f(t, y(t))$ is continuous, $|f| \leq M$ in R, and $\frac{df}{dt}$ and $\frac{df}{dy}$ are continuous, then the solution is unique.
- If $f(t, y(t))$ is *Lipschitz continuous*, i.e., if in $R \exists L$ such that $|f(t, y_1) - f(t, y_2)| \leq L|y_1 - y_2|$, then $f(t, y(t))$ is *Lipschitz continuous* and the solution exists and is unique to the ODE.

Note: If $\frac{df}{dy}$ exists and L is connected to $\frac{df}{dy}$, and for some $\xi \in (y_1, y_2)$ where $f(y_1) - f(y_2) = f'(\xi)(y_1 - y_2)$, then $L = \max_{\xi \in (y_1, y_2)} \left| \frac{df}{dy}(t, \xi) \right|$.

2 Introduction of Numeric Methods

Just like doing various Riemann sum schemes for approximating integrals in Calculus, a similar (fun) experience is had by students in introductory ODE class, where students approximate solutions to (usually unncessarily simplistic) differential equations by hand, using a scheme called the *Euler Method*.

Unfortunately, most students don't realize the tremendous power they were just handed in that (little bit of heaven) in class since they are just applying it to toy problems. Let's take a look. The Euler Method is defined by

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

where h is called the step-size. We find successive approximations to the solution of a differential equation by *time-stepping* forward, using each successive value to find the next. Essentially we would like h to be as small as possible. There's natural logic - the smaller h is, the closer all of numerical solution values are to the previous value, so less error may compound over time. Although this is true and feels right, making h too small results in computations that take orders of magnitude longer to solve.

Luckily for us, we'll discuss ways to get around this problem of making h ridiculously minuscule to find a highly accurate solution. Like root finding methods having different orders of convergence, we find different ODE numerical solvers have different orders of accuracy. It turns out that Euler's Method is only 1^{st} order. In a few minutes, we'll explore this idea further, but now I'll just list a few other popular ODE numeric methods.

- **Implicit Euler** $y_{n+1} = y_n + hf(t_{n+1}, y_{n+1})$

We see that the above looks strikingly similar to Euler's Method, except with future values of t_{n+1} and y_{n+1} inside the function f. Implicit methods are used for *stability* reasons for *stiff* differential equations. In order to step forward in time for this method, you will need use your favorite root-finding method as well. Like its explicit brother, the Euler Method, it is only 1^{st} order but has more useful stability properties.

- **Trapezoid** $y_{n+1} = y_n + \frac{1}{2}\left[f(t_n, y_n) + f(t_{n+1}, y_{n+1})\right]$

  It is apparent this seems to be a hybrid of both the Euler method and the Implicit Euler method. Hence it is still implicit and a root-finding algorithm must be used as well. However, unlike the explicit and implicit Euler Methods, Trapezoid is a 2^{nd} order method.

- **Taylor Methods** $y_{n+1} = y_n + hf(t_n, y_n) + \frac{1}{2}h^2\left(\frac{d}{dt}\right)^2 f(t_n, y_n) + ...$

  This method flat out resembles a sort of Taylor Series. It is. However, is exhibits k^{th} order accuracy, which is splendid, but unfortunately it falls into a mess of stability problems in many problems. Don't worry, we'll get to more of this stability business in a bit.

- **RK2** (Runge-Kutta-2) $y_{n+1} = y_n + \frac{h}{2}\left[f(t_n, y_n) + f(t_n, y_n + hf(y_n))\right]$

  The Runge-Kutta methods have a different flavor than the other examples above. In that they are not examples of *multi-step* methods but in a class of their own. This particular Runge-Kutta method has 2^{nd} order accuracy.

- **RK4** (Runge-Kutta-4) $y_{n+1} = y_n + \frac{1}{6}[k_1 + 2k_2 + 2k_3 + k_4]$, where

$$
\begin{aligned}
k_1 &= hf(t_n, y_n) \\
k_2 &= hf(t_n + \frac{h}{2}, y_n + \frac{1}{2}k_1) \\
k_3 &= hf(t_n + \frac{h}{2}, y_n + \frac{1}{2}k_2) \\
k_4 &= hf(t_n + h, y_n + k_3)
\end{aligned}
$$

  The RK4 method is arguably the most popular ODE numerical scheme that is used. It is usually employed as an *adaptive* scheme, meaning the step-size h changes appropriately as the solver computes more time-steps. The idea is to preserve the desired amount of accuracy as the solution moves forward in time, without having to use the worst case scenario h in that region, i.e., for regions where the solution is more singular than others you'd expect h to be very tiny, while regions where the solution is extremely smooth you could expect to use a larger h and get desired accuracy. Oh, and also this cute little method has 4^{th} accuracy.

There are many other popular methods that go by name, but memorizing all these methods isn't a good use of brain resources at this junction, especially when there is so much more mathematical magic to be studied!

2.1 MULTI-STEP METHODS!

The general multi-step method has the (seemingly) complicated form,

$$y_{n+1} = \sum_{j=0}^{p} a_j y_{n-j} + h \sum_{j=-1}^{p} b_j f(t_{n-j}, y_{n-j}),$$

where $\{a_j\}$ and $\{b_j\}$ are coefficients we'll discuss how to explicitly find later. Note if $b_{-1} = 0$ then method is *explicit*, but it is *implicit* if $b_{-1} \neq 0$. Also, we can think about multi-step methods as being derived through interpolating polynomials, i.e.,

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f(t, y(t))\, dt,$$

where you would interpolate or integrate $f(t, y(t))$ using some stencil and thereby deriving a specific multi-step scheme. These methods offer both advantages and disadvantages.

One advantage is that they only require 1 function call per time-step because everything from previous was already computed. Compared to $RK4$, which requires 4 function calls per time-step. Hence in theory a multi-step method should be about 4 times faster than $RK4$! One disadvantage is that multi-step methods *must* use uniform step sizes, unlike Runge-Kutta methods, which can used in an adaptive time-stepping algorithm.
Now it's time to discuss how to tell a good multi-step method from a dud. As alluded to above, we need to choose the coefficients $\{a_j\}$ and $\{b_j\}$ in some way - why not choose a way that will guarantee convergence! Splendid! It is now time to go down the rabbit hole of the *Dahlquist Equivalence Theorem*, which unites consistency and stability to convergence for a numerical ODE method. We'll begin by stating the theorem

Theorem 1. *A multi-step method is convergent if and only if it is consistent and stable.*

Well, that's concise. Let's chat about what *consistency* and *stability* mean in this context.

CONSISTENCY A numerical ODE method is said to be consistent if the multi-step discretization satisfies the governing ODE in the limit as $h \to 0$.

Okay, done. Nah, let's look at a few examples to solidify this concept.

- **Example 1:** Euler Method $y_{n+1} = y_n + hf(t_n, y_n)$

  How do we show the discretization satisfies the governing equation in the limit as $h \to 0$? Well, when in doubt call on your old friend - Taylor Series. Since we assume h is *sufficiently small*, meaning h is fine for numerical analysis in a hand-wavy sense, we can write y_{n+1} in a Taylor series from the previous value, y_n.

  $$y_{n+1} = y_n + h\frac{d}{dt}y_n + \frac{1}{2!}h^2\frac{d^2}{dt^2}y_n + \ldots$$

  Substituting the above into the Euler Method and rearranging, we find

$$\left[y_n + h\frac{d}{dt}y_n + \frac{1}{2!}h^2\frac{d^2}{dt^2}y_n + ... \right] - y_n = hf(t_n, y_n),$$

and hence we see

$$\frac{dy_n}{dt} = f(t_n, y_n) - \frac{1}{2!}h\frac{d^2y_n}{dt^2} + ...$$

$$\rightarrow \frac{dy_n}{dt} = f(t_n, y_n) - O(h)$$

Taking the limit as $h \rightarrow 0$, it is clear the governing ODE is satisfied, namely $\frac{dy}{dt} = f(t, y(t))$. Moreover, note that the governing ODE is satisfied linearly as $h \rightarrow 0$.

- **Example 2:** Two-Step Adams-Bashforth Method $y_{n+1} = y_n + \frac{3}{2}h\, f(t_n, y_n) - \frac{1}{2}h\, f(t_{n-1}, y_{n-1})$

  Although, this looks a bit more complicated, the analysis will proceed exactly the same, except now we *get* to play with even more Taylor Series! Check it out.

$$y_{n+1} = y_n + h\frac{d}{dt}y_n + \frac{1}{2!}h^2\frac{d^2}{dt^2}y_n + ...$$

$$f(t_{n-1}, y_{n-1}) = f(t_n, y_n) - h\frac{d}{dt}f(t_n, y_n) + \frac{1}{2}\, h^2 f(t_n, y_n) + ...$$

Substituting the above series into the numerical method and rearranging, we find

$$y_{n+1} - y_n = \frac{3}{2}h\, f(t_n, y_n) - \frac{1}{2}h\, f(t_{n-1}, y_{n-1})$$

$$\left[y_n + h\frac{d}{dt}y_n + \frac{1}{2!}h^2\frac{d^2}{dt^2}y_n + ... \right] - y_n = \frac{3}{2}h\, f(t_n, y_n) - \frac{1}{2}\, h\left[f(t_n, y_n) - h\frac{d}{dt}f(t_n, y_n) + \frac{1}{2}\, h^2 f(t_n, y_n) + ... \right]$$

$$\frac{dy_n}{dt} + \frac{1}{2!}\, h\frac{d^2y_n}{dt^2} + ... = f(t_n, y_n) + \frac{1}{2}\, h\frac{d}{dt}f(t_n, y_n) - \frac{1}{4}\, h^2\frac{d^2}{dt^2}f(t_n, y_n) + ...$$

$$\frac{dy_n}{dt} = f_n - \frac{1}{2}\, h\left[\frac{d^2y_n}{dt^2} - \frac{d}{dt}f_n \right] - h^2\left[\frac{1}{3!}\frac{d^3y_n}{dt^3} + \frac{1}{4}\frac{d^2}{dt^2}f_n \right] + ...$$

where $f_n = f(t_n, y_n)$. Recall that the original ODE was,

$$\frac{dy}{dt} = f(t, y(t)).$$

Differentiating both sides with respect to t, we find

$$\frac{d^2 y_n}{dt^2} = \frac{d}{dt} f(t_n, y_n).$$

Hence we see that,

$$\frac{dy_n}{dt} = f(t_n, y_n) + O(h^2).$$

Therefore as $h \to 0$, we see the discretization satisfies the governing equation to second order accuracy.

STABILITY *(Zero-Stability)* A numerical method is *zero-stable* if the solution remains bounded as $h \to 0$ for finite overall time.
For a general multi-step method, as $h \to 0$,

$$y_{n+1} = \sum_{j=0}^{p} a_j y_{n-j}.$$

The above equation is a linear difference equation, which can be solved explicitly using standard recursive methods, i.e., let

$$\begin{aligned} y_{n+1} &= \lambda y_n \\ y_n &= \lambda y_{n-1} \\ &\vdots \\ y_{n-p+1} &= \lambda y_{n-p} \end{aligned}$$

and therefore we get:

$$\lambda^p - \sum_{j=0}^{p} a_j \lambda^{p-1-j} = 0.$$

Now doing every high schoolers dream is to do nothing but solve these types of algebraic equations. Assuming you're able to accomplish that feat, if

$$|\lambda| \leq 1,$$

the method is *zero-stable*.

2.2 RUNGE-KUTTA METHODS

At first glance, Runge-Kutta methods look overly complicated compared to their multi-step comrades.The complicated function evaluations come into play from the beautiful mechanics behind Runge-Kutta methods, where trial steps are used in intermediate points of a single time-step interval to cancel out lower-order error terms.

In general, Runge-Kutta methods match Taylor expansions as high as possible. The general form of a Runge-Kutta method is

$$y_{n+1} = y_n + \sum_{j=1}^{N} a_j K_j,$$

where

$$K_j = hf\left(t_n + c_j h, y_n + \sum_{m=1}^{j-1} d_{jm} K_m\right)$$

$$c_j = \sum_{m=1}^{j-1} d_{jm}.$$

By matching coefficients with those of a Taylor series of desired order, say N, we can find all necessary coefficients above for a method with desired truncation error, $O(h^N)$.

Let's do an example to illustrate this.
Note, compared to multi-step methods, the Runge-Kutta method has an easy test for *consistency*. It's actually given by the bottom equation above, if $c_j = \sum_{m=1}^{j-1} d_{jm}$ the method is consistent.

2.2.1 DERIVATION OF RK2

In this section we wish to derive the standard $RK2$ scheme mentioned prior, e.g.,

$$y_{n+1} = y_n + \frac{h}{2} f(t, y) + \frac{h}{2} f(t + h, y + hf(t, y)), \tag{2.1}$$

to solve the problem of $y' = f(t, y)$ with $y(0) = y_0$.
To do this, we must remember our multivariate Taylor Series old friend. Consider $f(t, y)$ to be a C^∞ function, with h, k being sufficiently small parameters defined by an open interval around $\mathbf{x} = (t, y)$, i.e., $|t - a| < h$ and $|y - b| < k$, where the Taylor Series is centered around some point $\mathbf{c} = (a, b)$, then we can write

$$f(\mathbf{x}) = f(\mathbf{c}) + \left[(\mathbf{x} - \mathbf{c}) \cdot \nabla f(\mathbf{c})\right] + \left[(\mathbf{x} - \mathbf{c}) \cdot H(\mathbf{x}) \cdot (\mathbf{x} - \mathbf{c})\right] + \ldots,$$

or

$$f(t+h,y+k) = f(t,y) + hf_t(t,y) + kf_y(t,y) + \frac{h^2}{2!}f_{tt} + khf_{ty} + \frac{k^2}{2!}f_y y(t,y) + \ldots.$$

Now we look at the Taylor Series form of $y(t+h)$,

$$y(t+h) = y(t) + hy'(t) + \frac{h^2}{2!}y''(t) + \mathcal{O}(h^3) \quad \text{now substitute } y'(t) = f(t,y) \text{ and } y''(t) = \frac{d}{dt}f(t,y)$$

$$= y(t) + hf(t,y) + \frac{h^2}{2}\frac{d}{dt}f(t,y) + \mathcal{O}(h^3)$$

$$= y(t) + hf(t,y) + \frac{h^2}{2}\big[f_t(t,y) + f_y(t,y)f(t,y)\big] + \mathcal{O}(h^3)$$

$$= y(t) + \frac{h}{2}f(t,y) + \frac{h}{2}\big[f(t,y) + hf_t(t,y) + hf_y(t,y)f(t,y)\big] + \mathcal{O}(h^3)$$

$$= y(t) + \frac{h}{2}f(t,y) + \frac{h}{2}f(t+h,y+hf(t,y)) + +\mathcal{O}(h^3)$$

$$= y(t) + \frac{h}{2}K_1 + \frac{h}{2}K_2.$$

And therefore upon the usual discretization shenanigans,

$$y_{n+1} = y_n + \frac{h}{2}\big(K_1 + K_2\big),$$

where $K_1 = f(t,y)$ and $K_2 = f(t+h, y+hf(t,y))$. Thus we have derived the $RK2$ numerical scheme. Deriving the RK methods typically will have this sort of flavor, where you expand a Taylor Series, and continually remove terms by the adding additional terms to cancel higher and higher order terms from the series. Here we only concerned ourselves with removing terms up to $\mathcal{O}(h^2)$, but in principle could aim our sights higher.

2.2.2 RK AND STABILITY

It's time to briefly chat about stability. For Runge-Kutta methods, we'll consider the concept of *B-stability*. Consider 3 matrices, A, M and Q. Let

$$\begin{aligned}
\mathbf{a} &= (a_1, a_2, ..., a_p)^T \\
D &= \left[d_{jm}\right] \\
A &= diag(a_1, a_2, ..., a_p). \\
M &= AD + D^T A - \mathbf{aa^T} \\
Q &= AD^{-1} + D^{-T} A - D^{-T}\mathbf{aa^T}D^{-1}.
\end{aligned}$$

If matrices A and Q are non-negative definite, then the Runge-Kutta scheme is said to be *B-stable*. Note If matrices A and M are both non-negative definite, then the method is said to be *algebraically stable*.

3 BVPs!

The other type of ordinary differential equation that we have not yet considered, are *boundary value problems*, or BVPs. A boundary value problem has the same look as an initial value problem; however, coming in a higher derivative form and with boundary conditions at some left-point, $x = a$, and right-point, $x = b$, in an interval of interest, $[a, b]$. Consider now the following problem,

$$\frac{d^2y}{dx^2} = f\left(x, y, \frac{dy}{dx}\right),$$

with $y(a) = y_a$ and $y(b) = y_b$ on $[a, b]$. As in the initial value case, $f\left(x, y, \frac{dy}{dx}\right)$ can be a non-linear function of y or $\frac{dy}{dx}$. This flavor of equation come in very useful for studying many physical processes, like heat transfer, fluid flows, and electromagnetism for problems in $1D$...however, for problems in reality, these equations take on a more *PDE* identity.

Unlike initial value problems, we will not be able to use schemes like the time-stepping ones we described before. Rather we will have to develop methods that solve the equation on all the points in the interval $[a, b]$ simultaneously. Luckily for us, a lot of the machinery we can use is fairly straight-forward when we use *finite difference* methods and employ differentiation stencils to do most of the work.

We will proceed by introducing how to setup a finite difference method for a general 2^{nd} order BVP in both a linear and non-linear case.

3.1 FINITE DIFFERENCE METHODS FOR BVPS

Using Finite Difference methods for BVPs has the journey of the approximating a bunch of derivatives using differentiation stencils and then arriving at a big linear (or non-linear) sys-

tem of equations to solve. We will illustrate this using two different cases- one linear and one non-linear, and discuss the differences in each method. Spoiler, for linear problems we will arrive at a linear system to solve, of which we can "easily" invert a matrix to get our numerical solution, and for non-linear problems, we arrive at a non-linear system, in which we need to use a desired root-finding scheme to find the numerical solution.

Either way, in both problems, our discretized solution we will be the approximate function values at specific equally spaced grid points. For a resolution of $N+1$ points in $[a,b]$, we define a step-size, $\Delta x = h = \frac{b-a}{N+1}$, and get the following grid

$$\mathbf{x} = \Big\{ x_0 = a, x_1 = a+h, x_2 = a+2h, \dots x_{N+1} = a+(N+1)h = b \Big\}.$$

Now we define the approximate solution vector, $\mathbf{y}$, to be

$$\mathbf{y} = \begin{pmatrix} y_a \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{N-1} \\ y_N \\ y_b \end{pmatrix} = \begin{pmatrix} y_a \\ y(x_1) \\ y(x_2) \\ y(x_3) \\ \vdots \\ y(x_{N-1}) \\ y(x_N) \\ y_b \end{pmatrix},$$

where y_a and y_b are the given boundary conditions of the problem. However, since we know the true values of y at $x = a$ and $x = b$, we only need to solve this equation on $[x_1, x_N]$ from our discretization, and hence only are concerned with $\mathbf{y} = \big(y_1, y_2, \dots, y_{N-1}, y_N\big)$. Now that we have defined our solution vector, $\mathbf{y}$, and numerical grid, $\mathbf{x}$, we will look at an example of solving a linear and non-linear problem.

3.2 Finite Difference Method for Linear BVPs

Consider the following general 2^{nd}-order linear BVP,

$$\frac{d^2y}{dx^2} + p(x)\frac{du}{dx} + q(x)u = g(x),$$

with $y(a) = y_a$ and $y(b) = y_b$ on $[a,b]$. To solve this problem, we have already defined the numerical grid, $\mathbf{x}$, and solution vector, $\mathbf{y}$ above, and now have to approximate the $\frac{d^2y}{dx^2}$ and $\frac{dy}{dx}$ term in terms of $\mathbf{y}$. This is what is referred to *discretizing the equation* in Finite Difference lingo.

We can now use the machinery of differentiation stencils to approximate those derivatives. One thing to be aware of is that depending of which stencil we use, will give us different orders of accuracy. For example if we choose to use a 2^{nd}-order differentiation stencil for $\frac{d^2y}{dx^2}$,

but only a 1^{st}-order stencil for $\frac{dy}{dx}$, we can only assume over overall error will be 1^{st}-order. On that note, if we have a desired order of accuracy, we have to make sure each stencil achieves such order.

For this problem, let's assume we wish our solution to be 2^{nd}-order accurate. Let's start with $\frac{dy}{dx}$. If you recall your differentiation stencils, this would be a centered difference scheme for the first derivative, i.e.,

$$y'_n = \frac{y_{n+1} - y_{n-1}}{2h} + \mathcal{O}\left(h^2\right).$$

Using the above stencil we can discretize the first derivative by

$$\frac{d\mathbf{y}}{dx} = \begin{pmatrix} y'_1 \\ y'_2 \\ \vdots \\ y'_{N-1} \\ y'_N \end{pmatrix} = \begin{pmatrix} \frac{y_2 - y_a}{2h} \\ \frac{y_3 - y_1}{2h} \\ \vdots \\ \frac{y_N - y_{N-2}}{2h} \\ \frac{y_b - y_{N-1}}{2h} \end{pmatrix} + \mathcal{O}\left(h^2\right).$$

Next we approximate $\frac{d^2y}{dx^2}$ using a 2^{nd}-order stencil, i.e.,

$$y''_n = \frac{y_{n+1} - 2y_n + y_{n-1}}{h^2} + \mathcal{O}\left(h^2\right).$$

Hence, similar to the first derivative case, we simply use this stencil at all of the grid points, e.g.,

$$\frac{d^2\mathbf{y}}{dx^2} = \begin{pmatrix} y''_1 \\ y''_2 \\ \vdots \\ y''_{N-1} \\ y''_N \end{pmatrix} = \begin{pmatrix} \frac{y_2 - 2y_1 + y_a}{h^2} \\ \frac{y_3 - 2y_2 + y_1}{h^2} \\ \vdots \\ \frac{y_N - 2y_{N-1} + y_{N-2}}{h^2} \\ \frac{y_b - 2y_N + y_{N-1}}{h^2} \end{pmatrix} + \mathcal{O}\left(h^2\right).$$

Now that we have all the discretizations, we can substitute them into the BVP, along with evaluating the coefficient functions at the associated grid points,

$$\begin{pmatrix} \frac{y_2 - 2y_1 + y_a}{h^2} \\ \frac{y_3 - 2y_2 + y_1}{h^2} \\ \vdots \\ \frac{y_N - 2y_{N-1} + y_{N-2}}{h^2} \\ \frac{y_b - 2y_N + y_{N-1}}{h^2} \end{pmatrix} + \begin{pmatrix} p(x_1)\left(\frac{y_2 - y_a}{2h}\right) \\ p(x_2)\left(\frac{y_3 - y_1}{2h}\right) \\ \vdots \\ p(x_{N-1})\left(\frac{y_N - y_{N-2}}{2h}\right) \\ p(x_N)\left(\frac{y_b - y_{N-1}}{2h}\right) \end{pmatrix} + \begin{pmatrix} q(x_1)y_1 \\ q(x_2)y_2 \\ \vdots \\ q(x_{N-1})y_{N-1} \\ q(x_N)y_N \end{pmatrix} = \begin{pmatrix} g(x_1) \\ g(x_2) \\ \vdots \\ g(x_{N-1}) \\ g(x_N) \end{pmatrix}.$$

However, since the above equation is linear we can write the optimal matrix form of $A\mathbf{y} = \mathbf{g}$. Take a look!

$$\begin{bmatrix} -\frac{2}{h^2}+q(x_1) & \frac{1}{h^2}+\frac{p(x_1)}{2h} & 0 & \cdots & 0 \\ \frac{1}{h^2}-\frac{p(x_2)}{2h} & -\frac{2}{h^2}+q(x_2) & \frac{1}{h^2}+\frac{p(x_3)}{2h} & 0 & \cdots 0 \\ 0 & \frac{1}{h^2}-\frac{p(x_3)}{2h} & -\frac{2}{h^2}+q(x_3) & \frac{1}{h^2}+\frac{p(x_4)}{2h} & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ & & \frac{1}{h^2}-\frac{p(x_{N-1})}{2h} & -\frac{2}{h^2}+q(x_{N-1}) & \frac{1}{h^2}+\frac{p(x_N)}{2h} \\ 0 & \cdots & & \frac{1}{h^2}+\frac{p(x_N)}{2h} & -\frac{2}{h^2}+q(x_N) \end{bmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{N-1} \\ y_N \end{pmatrix}$$

$$= \begin{pmatrix} g(x_1)+\frac{y_a}{h^2}-p(x_1)\frac{y_a}{2h} \\ g(x_2) \\ \vdots \\ g(x_{N-1}) \\ g(x_N)+\frac{y_b}{h^2}-p(x_N)\frac{y_b}{2h} \end{pmatrix}.$$

All we are left to do is to solve this linear system of equations. We note that the above matrix is tri-diagonal, which lends itself to be inverted using *LU* factorization, or a "Sweep" type method, which makes it $\mathcal{O}(N)$ operations. We then have our approximate solution to the linear BVP.

3.3 Finite Difference Method for Non-Linear BVPs

We now consider the following non-linear BVP,

$$\frac{d^2y}{dx^2} + F(y) = g(x),$$

, where $F(y)$ is a non-linear function of y, with $y(a) = y_a$ and $y(b) = y_b$ on $[a,b]$. To solve this problem, we have already defined the numerical grid, $\mathbf{x}$, and solution vector, $\mathbf{y}$ above. We have already discretized $\frac{d^2y}{dx^2}$ in the above linear in terms of $\mathbf{y}$. We will proceed similarly, using a 2^{nd}-order finite difference scheme, i.e.,

$$y_n'' = \frac{y_{n+1} - 2y_n + y_{n-1}}{h^2}.$$

We then have

$$\frac{d^2\mathbf{y}}{dx^2} = \begin{pmatrix} y_1'' \\ y_2'' \\ \vdots \\ y_{N-1}'' \\ y_N'' \end{pmatrix} = \begin{pmatrix} \frac{y_2 - 2y_1 + y_a}{h^2} \\ \frac{y_3 - 2y_2 + y_1}{h^2} \\ \vdots \\ \frac{y_N - 2y_{N-1} + y_{N-2}}{h^2} \\ \frac{y_b - 2y_N + y_{N-1}}{h^2} \end{pmatrix} + \mathcal{O}\left(h^2\right),$$

and hence can define

$$A = \frac{1}{h^2}\begin{bmatrix} -2 & 1 & \cdots & & 0 \\ 1 & -2 & 1 & 0 & \vdots \\ 0 & 1 & -2 & 1 & \\ 0 & \cdots & & 1 & -2 \end{bmatrix},$$

and

$$F(\mathbf{y}) = \begin{pmatrix} f(y_1) \\ f(y_2) \\ \vdots \\ f(y_{N-1}) \\ f(y_N) \end{pmatrix},$$

and

$$g(\mathbf{x}) = \begin{pmatrix} g(x_1) \\ g(x_2) \\ \vdots \\ g(x_{N-1}) \\ g(x_N) \end{pmatrix}.$$

Therefore we have the following non-linear system of equations to solve

$$A\mathbf{y} + f(\mathbf{y}) = g(\mathbf{x}).$$

To solve this non-linear system, we find use a root-finding scheme. For example, we will proceed with a standard *multivariate Newton's Method* to do so. Recall the multivariate Newton's Method to find the roots of a some multivariable function, $H(\mathbf{z})$, can be written as

$$\mathbf{z}^{m+1} = \mathbf{z}^m - \left(\nabla H\left(\mathbf{z}^m\right)\right)^{-1} H\left(\mathbf{z}^m\right),$$

where $\nabla H(\mathbf{z}^m)$ is the Jacobian matrix. To put our problem into the form for the multivariate Newton's Method, we define

$$H(\mathbf{y}) = A\mathbf{y} + f(\mathbf{y}) - g(\mathbf{x}) = 0,$$

and hence finding the roots of H will give us our numerical solution to the non-linear BVP.

The algorithm then looks like

Multivariate Newton's Method Algorithm

Initial guess: $\mathbf{y}^0$
while (*error tolerlance is not met*)
    Find: function value $H^m = H(\mathbf{y}^m)$
    Find: Jacobian Matrix: $\left(\nabla H^m\right) = \nabla H(\mathbf{y}^m)$
    Solve the system: $\left(\nabla H^m\right)\mathbf{z}^m = H^m$ for $\mathbf{z}^m$.
    Solve for next iterate: $\mathbf{y}^{m+1}$, $\mathbf{y}^{m+1} = \mathbf{z}^m - \mathbf{y}^m$.