Numerical Analysis Final Exam

Name: Nate Stemen (20906566)
Email: nate.stemen@uwaterloo.ca

Due: Mon, Dec 14, 2020 4:30 PM
Course: AMATH 740

Problem 1

(a) Consider the absolute values of the eigenvalues of A. Approximately, for large n, what is the largest eigenvalue (in absolute value) of A? What is, approximately, the smallest eigenvalue (in absolute value) of A, for large n?

(b) Approximately, for large n, what is the 2-norm condition number of A? What is its order as a function of h, and as a function of n? (Remember that A is symmetric, so make sure to use the appropriate formula for the 2-norm condition number.)

Solution. (a) We start with the fact that the $\sin^2(x)$ function attains it's maximum (for positive x) at $x \in \{\frac{\pi}{2}, \frac{3\pi}{2}, \dots\}$. Now unfortunately because $k, l \in \{1, \dots, N\}$, we can never actually attain any of these values because $\frac{k\pi}{2(N+1)} < \frac{\pi}{2}$. That said for large N we have $N \approx N+1$, so if we take k to be N then $\frac{k\pi}{2(N+1)} \approx \frac{\pi}{2}$. Hence, for the largest eigenvalue we take $k = N = l$.

$$|\lambda_{\max}| = |\lambda_{N,N}| = 4\left[\sin^2\left(\frac{N\pi}{2(N+1)}\right) + \sin^2\left(\frac{N\pi}{2(N+1)}\right)\right] = 8\sin^2\left(\frac{N\pi}{2(N+1)}\right)$$

For the smallest eigenvalue, we know $\sin^2(x)$ attains it's minimum (again for positive x) at $x \in \{0, \pi, \dots\}$, but as we saw above, the argument to $\sin^2(x)$ is bounded above by $\frac{\pi}{2}$, so we opt to get as close to the $x = 0$ minimum. To do this we take $k = 1 = l$.

$$|\lambda_{\min}| = |\lambda_{1,1}| = 4\left[\sin^2\left(\frac{\pi}{2(N+1)}\right) + \sin^2\left(\frac{\pi}{2(N+1)}\right)\right] = 8\sin^2\left(\frac{\pi}{2(N+1)}\right)$$

Now for more approximations. In $|\lambda_{\max}|$, when N is *very* large, we should always be very close to the maximum of $\sin^2(x)$ which is 1, so we can say $|\lambda_{\max}| \approx 8$. For $|\lambda_{\min}|$ we know we are working with a $\sin(x)$ function near 0, and hence we use the classic $\sin(x) \approx x + \mathcal{O}(x^3)$ Taylor approximation, but squaring both sides to say $\sin^2(x) \approx x^2 + \mathcal{O}(x^4)$. Thus we have $|\lambda_{\min}| \approx \frac{2\pi^2}{(N+1)^2}$.

(b) Now to calculate the 2-norm condition number we use the fact that it can be written as the ratio of the largest eigenvalue to the smallest.

$$\kappa_2(A) = \frac{|\lambda_{\max}|}{|\lambda_{\min}|} = \frac{8}{\frac{2\pi^2}{(N+1)^2}} = \frac{4(N+1)^2}{\pi^2} = \underbrace{\frac{4}{\pi^2}\left(n + 2\sqrt{n} + 1\right)}_{\mathcal{O}(n)} = \underbrace{\frac{4}{\pi^2 h^2}}_{\mathcal{O}(h^{-2})}$$

Problem 2

(a) Write the update formula in vector form (using the decomposition $A = A_D - A_L - A_U$).

(b) Rewrite the vector form in the standard form of the update formula for a one-step, stationary iterative method,

$$\mathbf{x}_{k+1} = \mathbf{x}_k + P\mathbf{r}_k,$$

to show that the preconditioning matrix P for SOR is given by

$$P = \omega(A_D - \omega A_L)^{-1}.$$

Solution. (a),(b) Here we'll take a crack at both parts in one go. If you want, the first line of the sequence can be taken to be the solution to part (a), and the rest part (b).

$$\begin{aligned}
\mathbf{x}_{k+1} &= (1-\omega)\mathbf{x}_k + \omega A_D^{-1}(\mathbf{b} + A_L\mathbf{x}_{k+1} + A_U\mathbf{x}_k) \\
A_D\mathbf{x}_{k+1} &= (1-\omega)A_D\mathbf{x}_k + \omega(\mathbf{b} + A_L\mathbf{x}_{k+1} + A_U\mathbf{x}_k) \\
A_D\mathbf{x}_{k+1} - \omega A_L\mathbf{x}_{k+1} &= (1-\omega)A_D\mathbf{x}_k + \omega\mathbf{b} + \omega A_U\mathbf{x}_k \\
(A_D - \omega A_L)\mathbf{x}_{k+1} &= [(1-\omega)A_D + \omega(A_D - A_L - A)]\mathbf{x}_k + \omega\mathbf{b} \\
&= A_D\mathbf{x}_k - \omega A_D\mathbf{x}_k + \omega A_D\mathbf{x}_k - \omega A_L\mathbf{x}_k - \omega A\mathbf{x}_k + \omega\mathbf{b} \\
&= (A_D - \omega A_L)\mathbf{x}_k + \omega\mathbf{r}_k \\
\mathbf{x}_{k+1} &= \mathbf{x}_k + \underbrace{\omega(A_D - \omega A_L)^{-1}}_{P}\mathbf{r}_k
\end{aligned}$$

Problem 3

When solving a linear system, in which circumstances would LU decomposition be the best choice, or when is CG or GMRES useful? What are some of the key pros and cons of those methods? Consider aspects like size of the matrix, sparsity, condition number, symmetry, etc.

Solution. Let's start with LU decomposition on a matrix A. First A must be non-singular in order to guarantee a unique LU decomposition. In addition this method is great when you want to probe the matrix A more so than just solving a system $A\mathbf{x} = \mathbf{b}$. If we do have the LU decomposition of A, then we can solve a system like this with forward and backward substitution, but unfortunately it is somewhat slow costing $\mathcal{O}(n^3)$ for an $n \times n$ matrix, and $\mathcal{O}(n^2)$ if we're lucky to have a matrix A which is banded. So it is quite computationally expensive to solve systems of linear equations with this method, but first of all it is exact (in exact arithmetic), and second of all it can help us understand A further by allowing efficient computation of quantities like the determinant. If we decompose $A = LU$ where L is a unit lower triangular matrix, and U is an upper triangular matrix, then the determinant of A can be efficiently calculate as $\det(A) = \prod_i u_{ii}$. Because the determinant encodes *so* much useful information about A in it, having this efficient method of computing the determinant is wonderful.

Moving on to CG and GMRES method, we have that they are both iterative methods. This is great because as the LU decomposition required us to find exact solutions, often we do not need exact solutions and rather only accurate to a given precision or error. This means that we can often find approximate solutions with these methods much faster than we can with an exact method like LU decomposition. This is only exacerbated if one has some intuition for the problem and is able to make an educated guess for the starting point of the iteration. In this sense if CG and GMRES are running as a computer program, this allows the programmer to provide insight to the computer that methods like LU do not allow.

The above properties can be taken advantage of even further when A is sparse and even fewer flops can be done each iteration. Lastly, it's worth noting that the conjugate gradient method works with symmetric positive-definite matrices, and the generalized minimal residuals method works with sparse, but not necessarily symmetric matrices. As far as I understand both CG and GMRES both need an invertible matrix as well.

Problem 4

Let A be the $n \times n$ matrix with entries $a_{i+1,i} = 1$ for $i = 1, \dots, n-1$, $a_{1,n} = 1$ and all other entries zero. Let $\mathbf{b}$ have entries $b_1 = 1$, $b_i = 0$ for $i = 2, \dots, n$, and let $\mathbf{x}_0$ be the zero vector.
Show that GMRES applied to the system $A\mathbf{x} = \mathbf{b}$ with initial guess $\mathbf{x}_0$ takes n steps to find the true solution, and that $\|\mathbf{b} - A\mathbf{x}_k\| = 1$ for $1 \leq k \leq n-1$.

Solution. We start by writing down our matrices and vectors at play.

$$A = \begin{bmatrix} 0 & 0 & \cdots & & 1 \\ 1 & 0 & & & \\ & 1 & 0 & & \\ & & \ddots & \ddots & \\ & & & 1 & 0 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad \mathbf{x}_0 = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Now for the GMRES method we take each iteration $\mathbf{x}_k$ from the Krylov space $\mathcal{K}_{i+1}(A) = \text{span}\{\mathbf{r}_0, A\mathbf{r}_0, A^2\mathbf{r}_0, \dots, A^i\mathbf{r}_0\}$, so it'll be useful to understand the action of A on the residuals.

$$\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0 = \mathbf{b} =: \mathbf{e}_1$$

$$A\mathbf{r}_0 = A\mathbf{b} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} =: \mathbf{e}_2$$

Now, it's straightforward to see the solution to this problem is $\mathbf{e}_n$ where we're using $\mathbf{e}_i$ to denote the standard basis of $\mathbb{R}^n$. Now because we pull each iteration from $\mathcal{K}_{k+1}(A)$, we know each iteration $\mathbf{x}_k$ will be a linear combination of the sort $\sum_{n=1}^{k} \alpha_k \mathbf{e}_k$, which, will never attain our solution until $k = n$. Thus for this system we will need a total of n steps.

To show $\|\mathbf{b} - A\mathbf{x}_k\| = 1$ we need to take a look at how we choose each $\mathbf{x}_k$. In particular we take them so that we minimize the length of $\mathbf{r}_{i+1} = \mathbf{r}_0 - A\mathbf{z}_i = \mathbf{b} - A\mathbf{z}_i$, and because A acting on any vector $\mathcal{K}_{i+1}(A)$ with $i < n$ can never return anything in the $\mathbf{e}_1$ spot we know $\|\mathbf{b} - A\mathbf{x}_i\| \geq 1$. But in particular we can always take $\mathbf{x}_i$ to be the zero vector which means $\|\mathbf{b} - A\mathbf{x}_i\| = 1$ can be attained for all i until we hit the last iteration.

Problem 5

Define empirical risk, expected risk, and generalization error. Describe in your own words what the main difference is between the first two, and how do they influence generalization error? Describe in your own words what is overfitting. You can sketch some graphs to illustrate your answer.

Solution. The **empirical risk** is the average loss over the training set. This is used because while our training data may be somewhat representative of the data we will feed into our network running in the wild, it won't be perfectly representative. The empirical risk helps us capture that idea. Mathematically we may write something like $F(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{w})$ where each f_i is the loss of each sample.

The **expected risk** is the loss when taken over the entire population. This is different from the empirical risk because when training networks we don't often train them on all the data we have available so that we can get an idea how the network/algorithm performs on data it's never seen (without going out and getting more). This measure should hopefully give us an idea how the model will perform in the wild. Symbolically one might write $\mathbb{E}[l(h(\mathbf{x};\mathbf{w}),\mathbf{y})]$.

Lastly, the **generalization error** is defined as the absolute value of the difference between these two types of errors.

As is often the case, we want our models to be as accurate as possible, and so we train them until they perform very well on the training data. That said, if taken too far, the model will learn all sorts of features that are not actually helpful and the expected risk will rise. As a result we often are interested in minimizing the expected risk. To do this we train our model in batches, checking the expected risk after each batch. Normally we will see a decrease in both the empirical and expected risk initially, but once we start to learn features that we don't care about the expected risk rises and we've started over fitting. This is the reason we break out data into a training group and a validation/testing group.

Problem 6

(a) Compute

$$\frac{\partial f}{\partial \hat{y}_j}$$

the sensitivity of loss function (1) with respect to the jth component of the output $\hat{\mathbf{y}}$ of the neural network.

(b) Assume that the output $\hat{\mathbf{y}}$ of the neural network is produced by a Softmax output layer,

$$\hat{\mathbf{y}} = g_{\text{softmax}}(\mathbf{z}).$$

Compute the sensitivity of the loss function with respect to the activation $\mathbf{z}$ of the Softmax layer, and simplify to obtain

$$\nabla_{\mathbf{z}} f = -\mathbf{y} + \hat{\mathbf{y}}.$$

Solution. (a)

$$\frac{\partial f}{\partial \hat{y}_j} = \frac{y_j}{\hat{y}_j}$$

(b) We'll do this component-wise since taking gradients with respect to transposes always confuses me.

$$\begin{aligned}
\frac{\partial f}{\partial z_i} &= -\sum_{j=1}^{q} \frac{\partial}{\partial z_i} y_j \log(\hat{y}_j) \\
&= -\sum_{j=1}^{q} \frac{y_j}{\hat{y}_j} \frac{\partial \hat{y}_j}{\partial z_i} \\
&= -\frac{y_i}{\hat{y}_i} \hat{y}_i (1 - \hat{y}_i) + \sum_{j \neq i}^{q} \frac{y_j}{\hat{y}_j} \hat{y}_j \hat{y}_i \\
&= -y_i (1 - \hat{y}_i) + \sum_{j \neq i}^{q} y_j \hat{y}_i \\
&= -y_i + \sum_{j=1}^{q} y_j \hat{y}_i \\
&= -y_i + \hat{y}_i \sum_{j=1}^{q} y_j \\
&= -y_i + \hat{y}_i
\end{aligned}$$

Where I've used the equations we derived in class for $\frac{\partial \hat{y}_i}{\partial z_j}$ for $i = j$ and $i \neq j$. Thus using standard arguments to go from one partial to all of them we have that $\nabla_{\mathbf{z}} f = -\mathbf{y} + \hat{\mathbf{y}}$.

Problem 7

Write the following third-order ODE as a first-order ODE system:

$$y'''(x) + 3y''(x) - 4y'(x) + 7y(x) = x^2 + 7,$$

and give the system in matrix form.

Solution. Let's first define the following functions:

$$y_1(x) = y(x) \qquad y_2(x) = y'(x) \qquad y_3(x) = y''(x).$$

We then have the following relations between them.

$$y_1'(x) - y_2(x) = 0 \qquad y_2'(x) - y_3(x) = 0.$$

This allows us to construct the following matrix system

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}' = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -7 & 4 & -3 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ x^2 + 7 \end{bmatrix}$$

This is equivalent to the above third order ODE.

Problem 8

Consider the θ method for approximating the ODE $y' = f(x, y)$:

$$y_{n+1} = y_n + h(\theta f(x_n, y_n) + (1-\theta) f(x_{n+1}, y_{n+1})),$$

with $\theta \in [0, 1]$. Derive the order of the local truncation error. Find the value of θ for which the order is maximal.

Solution. Starting with the definition of the local truncation error $\ell_{n+1} \coloneqq \hat{y}(x_{n+1}) - y_{n+1}$, we'll expand the second term using the above relation. Here we will denote $f(x_n, y_n)$ by f_n for brevity.

$$\begin{aligned} y_{n+1} &= y_n + h\Big[\theta f_n + (1-\theta)\Big(f_n + h f_x(x_n, y_n) + h f_y(x_n, y_n) y'(x_n) + \mathcal{O}(h^2)\Big)\Big] \\ &= y_n + h\Big[\theta f_n + (1-\theta)\Big(f_n + h f_x(x_n, y_n) + h f_y(x_n, y_n) f_n + \mathcal{O}(h^2)\Big)\Big] \end{aligned}$$

Now similarly let's expand $\hat{y}(x_{n+1})$

$$\begin{aligned} \hat{y}(x_{n+1}) &= \hat{y}(x_n) + h\hat{y}'(x_n) + \frac{h^2}{2}\hat{y}''(x_n) + \mathcal{O}(h^3) \\ &= y_n + h f_n + \frac{h^2}{2}\frac{\mathrm{d}}{\mathrm{d}x} f(x, y(x))\Big|_{x=x_n} + \mathcal{O}(h^3) \\ &= y_n + h f_n + \frac{h^2}{2}\left(f_x(x_n, y_n) + f_y(x_n, y_n) y'(x_n)\right) + \mathcal{O}(h^3) \\ &= y_n + h f_n + \frac{h^2}{2}\left(f_x(x_n, y_n) + f_y(x_n, y_n) f_n\right) + \mathcal{O}(h^3) \end{aligned}$$

Now we can take the difference.

$$\ell_{n+1} \coloneqq \hat{y}(x_{n+1}) - y_{n+1} = h^2\Big(\theta - \tfrac{1}{2}\Big)\left(f_x(x_n, y_n) + f_y(x_n, y_n) f_n\right)$$

Thus we see this method is $\mathcal{O}(h)$, unless $\theta = \frac{1}{2}$ in which case it is $\mathcal{O}(h^2)$. Notice in the case of $\theta = \frac{1}{2}$ we have the trapezoid rule.

Problem 9

Consider Heun's method for a scalar ODE,

$$y_{n+1} = y_n + \frac{h}{2}(f(x_n, y_n) + f(x_{n+1}, y_n + hf(x_n, y_n))).$$

Show that this method is numerically stable for the test equation $y' = \lambda y$ (with $\lambda \in \mathbb{C}, \text{Re}(\lambda) < 0$) *if and only if* $\left|1 + h\lambda + h^2\lambda^2/2\right| < 1$.

Solution. First thing we'll do is use the fact that $f(x_n, y_n) = \lambda y_n$ and plug this into our above iteration scheme.

$$\begin{aligned} y_{n+1} &= y_n + \frac{h}{2}[\lambda y_n + \lambda(y_n + h\lambda y_n)] \\ &= \left(1 + \frac{h}{2}\left[\lambda + \lambda + h\lambda^2\right]\right) y_n \\ &= \left(1 + h\lambda + \frac{h^2\lambda^2}{2}\right) y_n \end{aligned}$$

At this point we use the fact that $y_n \to 0$ as $n \to \infty$ because of the nature of the test ODE solution $y(x) = e^{\lambda x}$ with $\text{Re}(\lambda) < 0$. This type of solution must decay to 0 at $x \to \infty$. Luckily we can rewrite the above to

$$y_{n+1} = \left(1 + h\lambda + \frac{h^2\lambda^2}{2}\right)^n y_0,$$

and hence in order to the RHS to go to 0, we must have the constant term going to 0 for large n. This means

$$\left|1 + h\lambda + \frac{h^2\lambda^2}{2}\right| < 1.$$

None of this logic was unidirectional, so we have in fact proved the if and only if statement.

Problem 10

(a) Write down the secant iteration formula specifically for $f(x) = x^2 - 4$ (note: simplify the formula).
(b) For initial guess $x_0 = 0$, $x_1 = 1$, compute the next two iterates x_2 and x_3. Compute also $f(x_0), f(x_1), f(x_2)$ and $f(x_3)$.
(c) Sketch a graph of the function $f(x)$, indicate the x-points and function values of the iterates x_0, x_1, x_2, and x_3, and draw the secant lines of $f(x)$ that illustrate graphically how x_2 and x_3 are obtained, starting from x_0 and x_1.

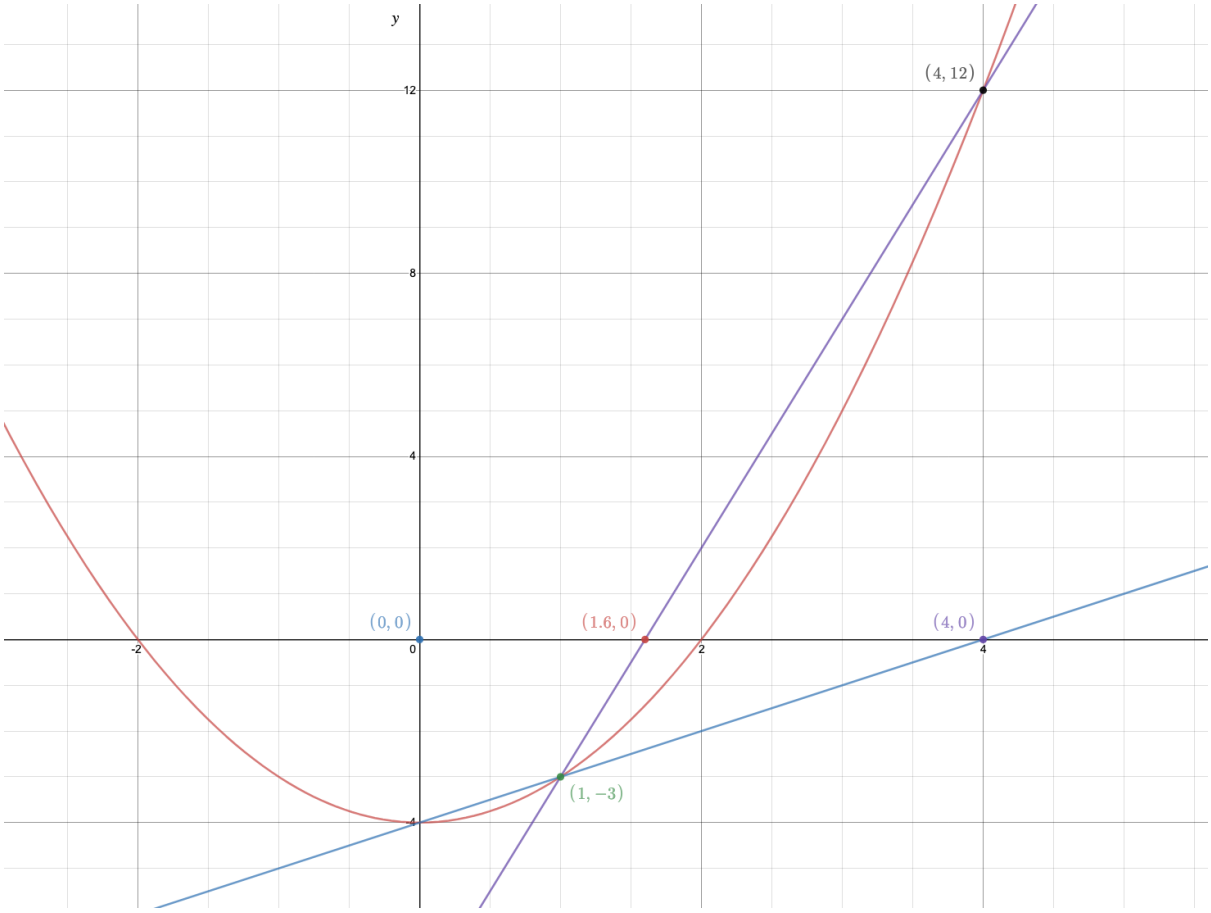
Solution. (a)

$$\begin{aligned}
x_{k+1} &= x_k - \frac{x_k^2 - 4}{(x_k^2 - 4) - (x_{k-1}^2 - 4)}(x_k - x_{k-1}) \\
&= x_k - \frac{x_k^2 - 4}{x_k^2 - x_{k-1}^2}(x_k - x_{k-1}) \\
&= x_k - \frac{x_k^2 - 4}{(x_k - x_{k-1})(x_k + x_{k-1})}(x_k - x_{k-1}) \\
&= x_k - \frac{x_k^2 - 4}{x_k + x_{k-1}} \\
&= \frac{1}{x_k + x_{k-1}}\left[x_k^2 + x_k x_{k-1} - x_k^2 + 4\right] \\
&= \frac{x_k x_{k-1} + 4}{x_k + x_{k-1}}
\end{aligned}$$

(b)

$$\begin{aligned}
& & f(x_0) &= 4 \\
& & f(x_1) &= 5 \\
x_2 &= \frac{x_1 x_0 + 4}{x_1 + x_0} = 4 & f(x_2) &= 4^2 - 4 = 12 \\
x_3 &= \frac{x_2 x_1 + 4}{x_2 + x_1} = \frac{8}{5} & f(x_3) &= \left(\frac{8}{5}\right)^2 - 4 = -\frac{36}{25}
\end{aligned}$$

(c)


y
12
8
4
-2
0
2
4
(4, 12)
(0, 0)
(1.6, 0)
(4, 0)
(1, −3)

Problem 11

(a) Show that the secant error ϵ_k for finding a simple root x^* of $f(x)$ ($\epsilon_k = x_k - x^*$), to leading order, decays like

$$\epsilon_{k+1} = \epsilon_k \epsilon_{k-1} \frac{f''(x^*)}{2f'(x^*)}.$$

(b) Consider the sequence $\{|\epsilon_0|, |\epsilon_1|, |\epsilon_2|, \ldots\}$ generated by Equation (2), and find a solution of the form

$$|\epsilon_{k+1}| = c|\epsilon_k|^d$$

for certain constants c and d that you are asked to compute (choose the relevant value of d that must occur near convergence). Conclude about the order of convergence of the secant method.

Solution. (a) First, we will take $x^* = 0$. We then have the correspondence $\epsilon_k = x_k$ which allows us to write the secant method iteration scheme as

$$\epsilon_{k+1} = \epsilon_k - f(\epsilon_k) \frac{\epsilon_k - \epsilon_{k-1}}{f(\epsilon_k) - f(\epsilon_{k-1})}.$$

Now we need to bring in the first a second derivatives of f so we'll start by Taylor expanding.

$$\begin{aligned} f(\epsilon_k) &= f(0) + f'(0)\epsilon_k + f''(0)\frac{\epsilon_k^2}{2} + \mathcal{O}\left(\epsilon_k^3\right) \\ &\approx f'(0)\epsilon_k + f''(0)\frac{\epsilon_k^2}{2} \\ f(\epsilon_k) - f(\epsilon_{k-1}) &= (\epsilon_k - \epsilon_{k-1})\left[f'(0) + (\epsilon_k + \epsilon_{k-1})\frac{f''(0)}{2}\right] \end{aligned}$$

Now let's plug these into the above iteration scheme and do algebra…

$$\begin{aligned} \epsilon_{k+1} &= \epsilon_k - \frac{f'(0)\epsilon_k + f''(0)\frac{\epsilon_k^2}{2}}{f'(0) + (\epsilon_k + \epsilon_{k-1})\frac{f''(0)}{2}} \\ &= \frac{1}{f'(0) + (\epsilon_k + \epsilon_{k-1})\frac{f''(0)}{2}} \left[\frac{\epsilon_k^2}{2} f''(0) + \frac{\epsilon_k \epsilon_{k-1}}{2} f''(0) - f''(0)\frac{\epsilon_k^2}{2}\right] \\ &= \frac{\frac{\epsilon_k \epsilon_{k-1}}{2} f''(0)}{f'(0) + (\epsilon_k + \epsilon_{k-1})\frac{f''(0)}{2}} \\ &= \frac{\epsilon_k \epsilon_{k-1} \frac{f''(0)}{2f'(0)}}{1 + (\epsilon_k + \epsilon_{k-1})\frac{f''(0)}{2f'(0)}} \\ &\approx \epsilon_k \epsilon_{k-1} \frac{f''(0)}{2f'(0)} = \epsilon_k \epsilon_{k-1} \frac{f''(x^*)}{2f'(x^*)} \end{aligned}$$

(b) Now we want $|\epsilon_{k+1}| = c|\epsilon_k|^d$ and we know $\epsilon_{k+1} = \epsilon_k \epsilon_{k-1} C$ where $C = \frac{f''(x^*)}{2f'(x^*)}$.

$$\begin{aligned} |\epsilon_{k+1}| = c|\epsilon_k|^d &= |\epsilon_k||\epsilon_{k-1}||C| \\ |\epsilon_k|^{d-1} &= |\epsilon_{k-1}|\left|\frac{C}{c}\right| \\ |\epsilon_k| &= |\epsilon_{k-1}|^{\frac{1}{d-1}}\left|\frac{C}{c}\right|^{\frac{1}{d-1}} \end{aligned}$$

Thus $d = \frac{1}{d-1}$ which has solutions $d = \frac{1\pm\sqrt{5}}{2}$, but we know d must be greater than 0 in order for convergence to happen, so $d = \frac{1+\sqrt{5}}{2}$ also known as the golden ratio φ! Cool. Similarly for the constant c we have

$$\begin{aligned} c &= \left|\frac{C}{c}\right|^{\varphi} \\ c^{\varphi+1} &= \left|\frac{f''(x^*)}{2f'(x^*)}\right| \\ c &= \left|\frac{f''(x^*)}{2f'(x^*)}\right|^{\frac{1}{\varphi+1}} \end{aligned}$$

Lastly we have

$$|\epsilon_k| = \left|\frac{f''(x^*)}{2f'(x^*)}\right|^{\frac{1}{\varphi+1}} |\epsilon|^{\varphi}$$

so we conclude that this method converges faster than linear, but not quite quadratically.

Problem 12

Consider the root finding method

$$x_{k+1} = x_k - \frac{2f(x_k)f'(x_k)}{2f'(x_k)^2 - f(x_k)f''(x_k)}.$$

Rewrite this method as a fixed point method $x_{k+1} = \phi(x_k)$. Find the order of convergence for this method (for a simple root). Derive an expression for the asymptotic error constant. (You can assume that $f(x)$ has continuous derivatives of any required order.)

Solution. First define the following function:

$$\phi(x) = x - \frac{2f(x)f'(x)}{2f'(x)^2 - f(x)f''(x)}.$$

Clearly we have $\phi(x_k) = x_{k+1}$ by using our root finding method. To find the order of convergence I'm pretty stuck. I've tried writing $f(x)$ as a Taylor series and expanding out the entire formula to the lowest degree term, but it becomes extremely messy, and I'm not sure that's the right approach to begin with. I do feel like I can make an educated guess about he order of convergence, but it's really not that strong: because the method is much more complicated than Newton's method (and takes into account more information than just the first derivative at a point), it's order of convergence must be greater than that of Newton's method (otherwise what's the point of throwing in all that extra information). So I might guess the order is qubic quartic, or maybe any $p \in (2, 3)$, where I doubt it could be higher than 4 since there are no third derivatives.