APPM 4600 — HOMEWORK # 4

1. Suppose we want to find a solution located near $(x, y) = (1, 1)$ to the nonlinear set of equations

$$\begin{aligned} f(x, y) &= 3x^2 - y^2 = 0, \\ g(x, y) &= 3xy^2 - x^3 - 1 = 0 \end{aligned} \tag{0.1}$$

   (a) Iterate on this system numerically (in Python), using the iteration scheme

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \end{bmatrix} - \begin{bmatrix} 1/6 & 1/18 \\ 0 & 1/6 \end{bmatrix} \begin{bmatrix} f(x_n, y_n) \\ g(x_n, y_n) \end{bmatrix}, \quad n = 0, 1, 2, \ldots$$

   starting with $x_0 = y_0 = 1$, and check how well it converges.

   (b) Provide some motivation for the particular choice of the numerical 2×2 matrix in the equation above.

   (c) Iterate on (0.1) using Newton's method, using the same starting approximation $x_0 = y_0 = 1$, and check how well this converges.

   (d) Spot from your numerical result what the exact solution is, and then verify that analytically.

2. Consider the nonlinear system

$$\begin{aligned} x + \cos(xyz) - 1 &= 0, \\ (1 - x)^{1/4} + y + 0.05z^2 - 0.15z - 1 &= 0, \\ -x^2 - 0.1y^2 + 0.01y + z - 1 &= 0. \end{aligned}$$

   Using your own codes test the following three techniques for approximating the solution to the nonlinear system to within 10^{-6}:

   - Newton's method
   - Steepest descent method
   - First Steepest descent method with a stopping tolerance of 5×10^{-2}. Use the result of this as the initial guess for Newton's method.

   Using the same initial guess, which technique converges the fastest? Try to explain the performance.

3. In this problem we find the the polynomial $p(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_n x^n$ that interpolates the data $(x_j, y_j) = (x_j, f(x_j)), j = 0, \ldots, n$.

   (a) Assume $(x_j, y_j), j = 1, \ldots, n$ are given. Derive the system $V\mathbf{c} = \mathbf{y}$ that determines the coefficients $\mathbf{c} = [c_1, \ldots, c_n]^T$ (here $\mathbf{y} = [y_1, y_2, \ldots, y_n]^T$), that is, find how the matrix V looks like.

   To solve the system of equations you can simply use inversion from Numpy's linear algebra package. You will write your polynomial evaluator.

(b) Find the polynomial (i.e. the coefficients $\mathbf{c}$) that interpolates

$$f(x) = \frac{1}{1+(10x)^2},$$

in the points $x_i = -1 + (i-1)h, h = \frac{2}{N-1}, i = 1, \ldots, N$. Plot data points as circles (`plot(x,f,'o')`) and, in the same plot, plot the polynomial and $f(x)$ on a finer grid (still on $x \in [-1, 1]$), say with 1001 points. Observe what happens when you increase N. Try $N = 2, 3, 4, \ldots$ and continue until the maximum value of $p(x)$ is about 100 (should be for $N \sim 17 - 20$). As you can see the polynomial behaves badly near the endpoints of the interval due to Runge's phenomena.

4. Solving the interpolation problem using the monomial basis (as above) is notoriously ill-conditioned, in fact it is possible to show `cond(V)`$\sim \pi^{-1} e^{\pi/4} (3.1)^n$. A better way of interpolating is to use either of the barycentric Lagrange interpolation formulas:

$$p(x) = \Phi_n(x) \sum_{j=0}^{n} \frac{w_j}{x - x_j} f(x_j).$$

$$p(x) = \frac{\sum_{j=0}^{n} \frac{w_j}{x-x_j} f(x_j)}{\sum_{j=0}^{n} \frac{w_j}{x-x_j}}, \quad x \neq x_j.$$

Where

$$\Phi_n(x) = \prod_{i=0}^{n} (x - x_i), \quad w_j = \frac{1}{\prod_{i=0, i\neq j}^{n} (x_j - x_i)}.$$

Using either of the above formulas try again to interpolate $f(x)$. Show with some pictures that you still get the same bad behavior close to the endpoints (this is a property of the function $f(x)$ and the distribution of the grid points not of the form of interpolation) but that the approximation is well behaved for small x for very large n.

5. It is much better to interpolate on a grid made up of points that are clustered towards the endpoints. Try to interpolate $f(x)$ in the Chebyshev points

$$x_j = \cos \frac{(2j-1)\pi}{2N}, \quad i = 1, \ldots, N,$$

using either of the methods above. Can you get the interpolation to fail now?