Newton's Method for Systems and Interpolation

Due: Monday, March 27, 11:59pm.

Push your assignment solutions to the appropriate GitHub Classroom repository. Submit the following for this assignment:

- A PDF file, typeset in LaTeX, with answers to questions 1(c), 2(a),(c) and (d), and 3(a) and (b), along with the corresponding .tex file. Also, incorporate the figures from 1(b), 2(b) and 3(e) into your LaTeX document.
- A file called `NonLinEqnsA4.py` containing the Python function `NonLinEqns` and a file called `JacobianA4.py` containing the Python function `Jacobian` for question 1(a), and a file called `NewtonPolyBuildA4.py` containing the Python function `NewtonPolyBuild` and a file called `NewtonPolyEvalA4.py` containing the Python function `NewtonPolyEval` for questions 3(c) and (d), respectively. Use the templates provided in the assignment repository.
- Python scripts called `Quest1.py` for question 1(b), and `Quest3.py` for question 3(d). Use the templates provided in the assignment repository.
- Three plots. One for each of questions 1(b), 2(b) and 3(d).

Question 1 — 30 marks

Consider the following system of equations:

$$\begin{aligned} x_1x_2^2 + x_1^2x_2 + x_1^3 &= 1 \\ x_1^3x_2^2 - 2x_1^5x_2 - x_1^2 &= -1 \end{aligned} \tag{1}$$

Your task is to find a solution of this system of equations using Newton's method for systems and using a starting guess of $\mathbf{x} = (1, 1)^t$.

(a) In the file called `NonLinEqnsA4.py`, write a Python function called `NonLinEqns` that defines the system of equations (1) as required for using Newton's method for systems. In a file called `JacobianA4.py`, write another function called `Jacobian`, that defines the Jacobian for the system of equations (1).

(b) Write a Python script called `Quest1.py`, which calls the function `NewtSysSolve` for performing Newton iteration for systems, to find the solution of the system of equations (1) using an initial guess of $\mathbf{x} = (1, 1)^t$. The function `NewtSysSolve` is defined in the file `NewtonSysSolve.py`, and is available in the assignment repository. Also, in the script, include code for plotting the approximate error versus the number of iterations on a semilogarithmic scale (log on the y axis). Save the figure and include it in your LaTeX document.

(c) Typeset in LaTeX, write out the solution you found in part (b), and answer the following question. Judging by the sequence of residuals, would you say that Newton iteration for systems has the same rate of convergence as Newton iteration for a single equation? Explain.

Question 2 **30 marks**

Consider the function

$$f(x) = e^{x/2} + \cos(x)$$

(**a**) Compute the interpolating polynomial $P_2(x)$ of degree at most 2, which is equal to $f(x)$ at the points $x_0 = 0.0$, $x_1 = 0.5$ and $x_2 = 1.0$. Write your answer, typeset in LaTeX, in the form:

$$P_2(x) = a_0 + a_1 x + a_2 x^2$$

You can use any appropriate method or code to find the coefficients. Compute the coefficients to 5 significant digits.

(**b**) On the same figure, plot the data points, the function $f(x)$, and the interpolating polynomial $P_2(x)$. Include a legend that labels each. Include the figure in your LaTeX document.

(**c**) Use the theorem in Lecture 16 to find an upper bound for the interpolation error $|E(x)| = |P_2(x) - f(x)|$ on the interval $x \in (0, 1)$. Remember that *the extrema of a differentiable function of a single variable on a finite interval occur where its derivative is zero and/or at the end points of the interval.*

(**d**) Based on the maximal value of $|f^{(k)}(x)|$ on the interval $[0, 1]$, do you expect the error of interpolation to decrease if we increase the order of interpolation (by adding more equally spaced nodes in the interval)? Explain.

Question 3 **40 marks**

In this question, you will implement and analyse polynomial interpolation using the Newton Polynomial basis instead of the standard monomial basis. In Lecture 16 Slides, we saw that we could use the Newton Polynomial basis to write the polynomial interpolant as:

$$\Pi_n(x) = \sum_{k=0}^{n} a_k \phi_k(x) = a_0\phi_0(x) + a_1\phi_1(x) + \cdots + a_n\phi_n(x),$$

where the Newton polynomial basis is given by $\phi_0 = 1, \phi_j = \prod_{k=0}^{j-1}(x - x_k), \quad 1 \le j \le n$, i.e.

$$\phi_0(x) = 1, \qquad \phi_1(x) = x - x_0, \qquad \phi_2(x) = (x - x_0)(x - x_1), \quad \ldots$$

$$\phi_n(x) = (x - x_0)(x - x_1)\ldots(x - x_{n-1})$$

If we write out the interpolation conditions $\Pi_n(x_k) = y_k$ for this basis (see Lecture 16 Slides), we see that the coefficients a_k of the interpolant $\Pi_n(x)$ can be found from the system of equations $M\mathbf{a} = \mathbf{y}$, where more specifically, we have:

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & \cdots \\ 1 & (x_1 - x_0) & 0 & 0 & \cdots \\ 1 & (x_2 - x_0) & (x_2 - x_0)(x_2 - x_1) & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}}_{M} \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}}_{\mathbf{a}} = \underbrace{\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}}_{\mathbf{y}}$$

Your task is to build the matrix M and solve the corresponding system given $n + 1$ data points (x_k, y_k).

(**a**) Write a pseudo-code for building the matrix M, for a given n and a given set of interpolating nodes (i.e. the x_k), Submit your answer type-set using LaTeX.

Your pseudo-code should have the following form:

Input: integer n and an $n+1 \times 1$ array of floats containing the values for the $n+1$ interpolating nodes

$\vdots$

Insert pseudocode here

$\vdots$

Output: M, an $n+1 \times n+1$ array of floats

(**b**) Compute the computational complexity of your algorithm from part (a), i.e. for a given n, determine the number of FLOPS it takes to build the matrix M. Your algorithm should be $O(n^2)$.

(**c**) In a file called `NewtonPolyBuildA4.py`, implement your pseudo-code in a function called `NewtonPolyBuild` which inputs an integer n and an array containing the interpolating nodes (x_k), and returns the $n+1 \times n+1$ numpy array containing the matrix M.

(**d**) Now that we have the matrix M, we can solve for the coefficients $\mathbf{a}$ by solving the corresponding system of linear equations. Once we have the coefficients $\mathbf{a}$, we need to use them to evaluate the interpolating polynomial at any desired value of x. In a file called `NewtonPolyEvalA4.py`, write a function called `NewtonPolyEval` which inputs an array of values of x, and returns an array containing the value of the polynomial interpolant at these values of x, i.e. $\Pi_n(x)$.

(**e**) Now, implement the interpolation using the Newton polynomial basis. Write a script called `Quest3.py` that calls the function `NewtonPolyBuild` and solves the corresponding system of linear equations to find the polynomial that interpolates the following data:

k	0	1	2	3	4	5
x_k	0	1.0	2.0	3.0	4.0	5.0
y_k	1.00	0.90	-0.21	-0.65	0.68	0.11

In the `Quest3.py` script, also include code for plotting your results. In particular, on the same figure, plot the data points and the interpolating polynomial, using the function `NewtonPolyEval` to evaluate the interpolant. Include the figure in your LaTex document.