PX3143 Assignment #3 — Non-linear Elliptic Equations

Produce a code to perform the following exercises. Include sufficient documentation in the code. Hand in the code, plus a text that includes figures and explanations that address the questions below, **everything in a single, readable, document**. It should be clear from the write-up what you have done, and what you have concluded. The maximum marks for each question is given in square brackets.

This assignment is worth 30% of the total mark for the module. You may work individually, or in groups or two or three – but no more! If you are working in a group, there should be one code and write-up for the group, with the student numbers of all group members on both the code and the write-up, and each member should submit it to Learning Central.

Each group must write its own code. Copied codes (even with small changes) will be considered as plagiarism.

The assignment has to be submitted by 5pm, Wednesday the 9th of December.

The aim is to solve the following non-linear elliptic differential equation

$$u'' + \alpha e^{-\sigma x^2} - u^3 = 0, \tag{1}$$

on the range $-1 \leq x \leq 1$ with $\alpha = 500$ and $\sigma = 100$.

1. Write a function to create the finite-difference approximation of the 2^{nd} derivative operator matrix for a staggered grid. Given the inputs N (the size of the matrix) and δx (the grid spacing), the function should return the tridiagonal matrix in the form of three arrays (a,b,c). Include the Neumann boundary conditions $u'(-1) = u'(1) = 0$ by modifiying specific elements of b as was done in week 9. [3]

2. Write a function to evaluate the matrix-vector product when the matrix is tridiagonal and given in (a,b,c) form. For an input vector $\mathbf{u}$, the function should return the vector $\mathbf{v}$ where

$$\begin{aligned} v_1 &= b_1 u_1 + c_1 u_2, \\ v_i &= a_i u_{i-1} + b_i u_i + c_i u_{i+1} \quad (\text{for } i = 2, 3, ..., N-1), \\ v_N &= a_N u_{N-1} + b_N u_N. \end{aligned} \tag{2}$$

[3]

3. Make a function to evaluate the LHS of the elliptic differential equation. Given an array of spatial points, $\mathbf{x}$, and an approximation of the solution, $\mathbf{u}$, the function should return the array $\mathbf{F}$, where

$$\mathrm{F}_i = (D^{(2)}\mathbf{u})_i + \alpha e^{-\sigma x_i^2} - u_i^3. \tag{3}$$

   Here, $D^{(2)}$ is the 2$^{\text{nd}}$ derivative operator from part 1. The matrix-vector product $(D^{(2)}\mathbf{u})$ can be evaluated using the function from step 2. [4]

4. Make a function to evaluate the derivative of the LHS of the elliptic differential equation with respect to u. Given an array of spatial points, $\mathbf{x}$, and an approximation of the solution, $\mathbf{u}$, the function should compute the matrix A in (a,b,c) form where $A_{i,j} = \partial F_i / \partial u_j$. See the lecture notes if you are unsure how to do this. [4]

5. Make a function to implement the tridiagonal matrix algorithm to solve the linear system $A\mathbf{x} = \mathbf{u}$ for $\mathbf{x}$ where the input matrix A is given in (a,b,c) form. This can be done in the same way as was done in weeks 8 and 9 for linear elliptic equations. [4]

6. With the above steps in place, iteratively solve the linearized version of the non-linear elliptic equation as follows. This should be done inside a function that takes the positions of left and right boundaries, the number of spatial points and the tolerance for the required accuracy, ε as inputs.

   Initialize the solver by making a staggered array ($\mathbf{x}$) of N points evenly spaced between the boundaries. Make an initial guess for the solution. This can simply be $\mathbf{u}_{\text{guess}} = \{1, 1, 1, ..., 1\}$. Next, perform the following steps within a *for*-loop:

   (a) Evaluate the array $\mathbf{F}$ using the function from step 3 with $\mathbf{x}$ and $\mathbf{u}_{\text{guess}}$ as inputs.

   (b) Evaluate the matrix $\mathbf{A}$ in (a, b, c) form using the function from step 4 with $\mathbf{x}$ and $\mathbf{u}_{\text{guess}}$ as inputs.

   (c) Using the tridiagonal matrix algorithm from step 5, solve $A\mathbf{d} = \mathbf{F}$ for $\mathbf{d}$.

   (d) Update the solution: $\mathbf{u}_{\text{guess}} \to \mathbf{u}_{\text{guess}} - \mathbf{d}$.

   (e) Calculate the difference $\delta = \sqrt{d_1^2 + d_2^2 + ... + d_N^2}$.

   (f) If $\delta < \varepsilon$ then return $\mathbf{x}$, $\mathbf{u}_{\text{guess}}$ and the values of δ from each iteration. Otherwise, repeat steps (a)-(f). You need to set a maximum number of iterations (20 should be sufficient) in case of the event that the algorithm diverges. [6]

7. Run your code using $\varepsilon = 10^{-10}$ and boundaries at $x_a = -1$ and $x_b = 1$. Generate three solutions using $N = 40$, 80 and 160 and verify 2$^{\text{nd}}$ order convergence. Plot each solution against x. Also, plot the values of δ against iteration number on a log-scale for each solution. [4]

8. Generate another solution with $N = 160$, $\varepsilon = 10^{-20}$ and use 20 as the maximum number of iterations. Again, plot the values of δ on a log-scale. Describe and **explain** your findings. [2]