COMP5930M - Scientific Computing

Coursework 2

September 21, 2022

Deadline

09:00, Monday 19th December

Task

The numbered sections of this document describe problems which are modelled by partial differential equations. A numerical model is specified which leads to a nonlinear system of equations. You will use one or more of the algorithms we have covered in the module to produce a numerical solution.

Matlab scripts referred to in this document can be downloaded from Minerva under Learning Resources / Coursework. Matlab implementations of some algorithms have been provided as part of the module but you can implement your solutions in any other language if you prefer.

You should submit your answers as a single PDF document via Minerva before the stated deadline. MATLAB code submitted as part of your answers should be included in the document. MATLAB functions should include appropriate `help` information that describe the purpose and use of that function.

Standard late penalties apply for work submitted after the deadline.

Disclaimer

This is intended as an individual piece of work and, while discussion of the work is encouraged, plagiarism of writing or code in any form is strictly prohibited.

1. A one-dimensional PDE: Nonlinear parabolic equation

[10 marks total]

Consider the nonlinear parabolic PDE: find $u(x,t)$ such that

$$\frac{\partial u}{\partial t} = \epsilon \frac{\partial^2 u}{\partial x^2} + \alpha \left(\frac{\partial u}{\partial x} \right)^2 \tag{1}$$

in the spatial interval $x \in (0,1)$ and time domain $t > 0$.
Here ϵ and α are known, positive, constants.

Boundary conditions are specified as $u(0) = 0$ and $u(1) = 1$.

Initial conditions are specified at $t = 0$ as $u(x,0) = x$.

We numerically approximate (1) using the method of lines on a uniform spatial grid with m nodes on the interval $[0,1]$ with grid spacing h, and a fixed time step of Δt.

Code for this problem can be downloaded from Minerva and is in the **Q1/** folder.

(a) Find the fully discrete formulation for (1) using the *central finite difference formulas*

$$u(x_i) \approx u_i, \quad \frac{\partial u}{\partial x}(x_i) \approx \frac{u_{i+1} - u_{i-1}}{2h}, \quad \frac{\partial^2 u}{\partial x^2}(x_i) \approx \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2}$$

in space and the *implicit Euler method* in time.
Define the nonlinear system $\mathbf{F}(\mathbf{U}) = \mathbf{0}$ that needs to be solved at each time step to obtain a numerical solution of the PDE (1).

(b) Derive the exact expression for each of the non-zero elements in row i of the Jacobian for this problem.

(c) Solve the problem for different problem sizes: $N = 81, 161, 321, 641$, where N is the number of unknowns to solve for (choose the grid size m appropriately to get the correct N). Use $N_T = 40$ time steps to solve from $t_0 = 0$ to $t_f = 2$. Create a table that shows the total computational time taken T, the total number of Newton iterations S, and the average number of Newton iterations per time step $T_S = T/S$. Time the execution of your code using Matlab functions `tic` and `toc`. Show for each different N:

i. the total number of unknowns, N
ii. the total computational time taken, T
iii. the total number of Newton iterations, S
iv. the average time spent per Newton iteration, $t_S = T/S$

Estimate the algorithmic cost of one Newton iteration as a function of the number of equations N from this simulation data. Does your observation match the theoretical cost of the algorithms you have used?
Hint: Take the values N and the measured $t_S(N)$ and fit a curve of the form $t_S = CN^P$ by taking logarithms in both sides of the equation to arrive at

$$\log t_S = \log C + P \log N,$$

then use `polyfit( log(N), log(tS), 1 )` to find the P that best fits your data.

(d) Repeat the timing experiment (c), but using now the Thomas algorithm `sparseThomas.m` to solve the linear system and the tridiagonal implementation of the numerical Jacobian computation `tridiagonalJacobian.m`. You will need to modify the code to call `newtonAlgorithm.m` appropriately.

Create a table that shows for each different N:

i. the total number of unknowns, N
ii. the total computational time taken, T
iii. the total number of Newton iterations, S
iv. the average time spent per Newton iteration, $t_S = T/S$

Estimate the algorithmic cost of one Newton iteration as a function of the number of equations N from this set of simulations. Does your observation match the theoretical cost of the algorithms you have used?

Hint: Take the values N and the measured $t_S(N)$ and fit a curve of the form $t_S = CN^P$ by taking logarithms in both sides of the equation to arrive at

$$\log t_S = \log C + P \log N,$$

then use `polyfit( log(N), log(tS), 1 )` to find the P that best fits your data.

2. A three-dimensional PDE: Nonlinear diffusion

[10 marks total]

Consider the following PDE for $u(x,y,z)$:

$$\frac{\partial}{\partial x}\left((1+u^2)\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left((1+u^2)\frac{\partial u}{\partial y}\right) + \frac{\partial}{\partial z}\left((1+u^2)\frac{\partial u}{\partial z}\right) = g(x,y,z), \qquad (2)$$

defined on $[x,y,z] \in [-10,10]^3$ with $u(x,y,z) = 0$ on the boundary of the domain, and some known function $g(x,y,z)$ that does not depend on u.

Using a finite difference approximation to the PDE (2) on a uniform grid, with grid size h and n nodes in each coordinate direction, the nonlinear equation $F_{ijk} = 0$ to be solved at an internal node may be written as

$$\begin{aligned}
F_{i,j,k} &= \frac{1}{h^2}\left[\left(1+\left(\frac{u_{i+1,j,k}+u_{i,j,k}}{2}\right)^2\right)(u_{i+1,j,k}-u_{i,j,k}) - \left(1+\left(\frac{u_{i,j,k}+u_{i-1,j,k}}{2}\right)^2\right)(u_{i,j,k}-u_{i-1,j,k})\right.\\
&+ \left(1+\left(\frac{u_{i,j+1,k}+u_{i,j,k}}{2}\right)^2\right)(u_{i,j+1,k}-u_{i,j,k}) - \left(1+\left(\frac{u_{i,j,k}+u_{i,j-1,k}}{2}\right)^2\right)(u_{i,j,k}-u_{i,j-1,k})\\
&+ \left.\left(1+\left(\frac{u_{i,j,k+1}+u_{i,j,k}}{2}\right)^2\right)(u_{i,j,k+1}-u_{i,j,k}) - \left(1+\left(\frac{u_{i,j,k}+u_{i,j,k-1}}{2}\right)^2\right)(u_{i,j,k}-u_{i,j,k-1})\right]\\
&- g(x_{i,j,k}, y_{i,j,k}, z_{i,j,k}) = 0 \qquad (3)
\end{aligned}$$

Code for this problem can be downloaded from Minerva and is in the **Q2/** folder. The Matlab function `runFDM.m` controls the numerical simulation of the discrete nonlinear system (3). To solve the problem call `runFDM(m)`, where m is the number of grid points in each direction so that the total number of unknowns is $N = m^3$.

(a) The Jacobian matrix is computed in the routine `newtonAlgorithm`. Extract the Jacobian matrix $J(x_0)$ from the first Newton iteration of the algorithm. Then:

   i. Visualise the sparsity pattern of the Jacobian matrix using the command `spy` in the case $m = 10$ and include the plot in your report. How many non-zero elements does it have?

   ii. Use the command `lu` to compute the LU-factorisation of `A` in the case $m = 10$. Visualise the sparsity pattern of the LU-factors using the command `spy` and include the plot in your report. What are the number of non-zero elements L and U, respectively?

   iii. Use these sparsity patterns to explain what we mean by *fill-in* of the LU-factors. Do you observe fill-in here?

(b) Solve the tangent problem by using the iterative linear solver `iterativeLinearSolve.m` based on the GMRES iteration (implemented by the `gmres` command in MATLAB). Use a fixed linear tolerance of $linTol = 10^{-7}$ and 100 maximum linear iterations. Solve the problem for $m = 10, 15, 20$ and measure in each case the number of Newton iterations and total number of linear iterations until convergence is achieved. Perform the same experiment with two different GMRES strategies:

   i. No preconditioner (default);

   ii. Inexact LU-preconditioner $M \approx LU$ computed by the MATLAB command

```
options.type = 'crout';
options.milu = 'row';
options.droptol = 0.05;
[L,U] = ilu(A,options);
M=L*U;
```

   (modify the file `iterativeLinearSolve.m` to implement the preconditioner).

Create a table that measures the total number of Newton iterations and total number of linear iterations in each case for $m = 10, 15, 20$ for both choices of preconditioning strategies. How does preconditioning affect the number of linear iterations required?

(c) Modify the file `iterativeLinearSolve.m` to implement an inexact Newton-Krylov method where the GMRES linear stopping tolerance is chosen as

$$linTol = \max\{\eta \, \|\mathbf{F}(\mathbf{x}_k)\|, 10^{-7}\},$$

where $\mathbf{F}(\mathbf{x}_k)$ is the nonlinear residual at iteration k and the parameter η is chosen from the Eisenstat-Walker rule:

$$\eta = 0.1 \frac{\|\mathbf{F}(\mathbf{x}_k)\|^2}{\|\mathbf{F}(\mathbf{x}_{k-1})\|^2}.$$

Repeat the experiment from (b) and update the table to include the number of Newton iterations and total number of linear iterations in this case. How does the inexact Newton-Krylov influence the total number of iterations required in this case?

Learning objectives

- Formulation of sparse nonlinear systems of equations from discretised PDE models.
- Measuring efficiency of algorithms for large nonlinear systems.
- Efficient implementation for sparse nonlinear systems.

Mark scheme

This piece of work is worth 20% of the final module grade.

There are 20 marks in total.

1. One-dimensional PDE [10 marks total]
   (a) Formulation of the discrete problem [3 marks]
   (b) Jacobian structure [3 marks]
   (c) Timings and analysis of complexity, general case [2 marks]
   (d) Timings and analysis of complexity, tridiagonal case [2 marks]
2. Three-dimensional PDE [10 marks total]
   (a) Analysis of the sparsity of the Jacobian matrix and the the LU factors [4 marks]
   (b) Experiments with the iterative GMRES linear solver [2 marks]
   (c) Experiments with the inexact Newton-Krylov method [4 marks]