IN5270 exam

Id 15718

December 6, 2021

1 Task 1

In this task we will solve the 1D PDE given as

$$-u_{xx} + au = f(x), \tag{1}$$

defined on the domain $x \in (0, L)$, where $f(x)$ is a given function and a is strictly positive. We have Neuman boundary conditions on both boundaries, i.e

$$\frac{du}{dx}|_{x=0} = C, \quad \frac{du}{dx}|_{x=L} = D$$

1.1 a)

The Galerkin method minimizes the residual using projection to a function space V. Our redisual is given as

$$R = u_{xx} - au + f(x) = 0.$$

Thus, using Galerkin method we get

$$(R, \psi_i) = 0, \psi_i \in V$$

where the parenthesis notation indicates the inner product of the two components R, v. We then get

$$(u_{xx} - au + f(x), \psi_i) = (u_{xx}, \psi_i) - (au, \psi_i) + (f(x), \psi_i) = 0$$

To handle the Neuman boundary conditions we use integration by parts. Thus we get

$$\begin{aligned} 0 &= \int_0^L u_{xx}\psi_i \, dx - \int_0^L au\psi_i \, dx + \int_0^L f(x)\psi_i \, dx \\ &= -\int_0^L u_x\psi_i' \, dx + [u_x\psi_i]_0^L - \int_0^L au\psi_i \, dx + \int_0^L f(x)\psi_i \, dx. \end{aligned}$$

The Neuman boundary conditions are used is $[u_x\psi_i]_0^L$, such that we get

$$[u_x\psi_i]_0^L = D\psi_i(L) - C\psi_i(0)$$

Using this as well as solving for u, we can rewrite the equation as

$$\int_0^L u_x\psi_i'\,dx + \int_0^L au\psi_i\,dx = D\psi_i(L) - C\psi_i(0) + \int_0^L f(x)\psi_i\,dx \tag{2}$$

To approximate u, we construct u using a sum of basis functions, i.e

$$u(x) = \sum_j c_j\psi_j. \tag{3}$$

The number os basis functions is equal to the total number of nodes in the domain. This gives us the complete weak variational form as

$$\sum_j c_j\left[\int_0^L \psi_j'\psi_i'\,dx + \int_0^L a\psi_j\psi_i\,dx\right] = D\psi_i(L) - C\psi_i(0) + \int_0^L f(x)\psi_i\,dx \tag{4}$$

This also gives us a set of linear algebraic equations on the form $(A\vec{x} = \vec{b})$, which can be solved using linear algebra.

1.2 b)

From equation 20 we see that we have a set of linear algebraic equations. This can be rewritten as a matrix equation, where we solve for the coefficients c_j. Here we have that

$$A_{i,j} = \int_0^L \left[\psi_j'\psi_i' + a\psi_j\psi_i\right]\,dx \tag{5}$$

and

$$b_i = D\psi_i(L) - C\psi_i(0) + \int_0^L f(x)\psi_i\,dx \tag{6}$$

Finite element divides the domain into several elements. Each element has at a minimum two nodes, used as reference points, and each node has an associated basis function. These basis functions are used to approximate the solution locally, and then assemble the solutions together to compute the entire solution. In our case we have N_e elements with size h_e. The advantage of computing the solution locally for each element is that the basis functions, when mapped to a local coordinate is equal to one is the node it belongs to, and zero on all other nodes. A mapping from the local domain $X \in [-1, 1]$ can be

$$x = x_m + \frac{h_e}{2}X, \tag{7}$$

where x_m is the midpoint of the element in global coordinate and h_e is the element size.

We will use P1 basis functions, which gives us two nodes per element. In the standardized coordinate system we have that the basis functions are defined as

$$\tilde{\phi}_0 = \frac{1}{2}(1 - X) \tag{8}$$

$$\tilde{\phi}_1 = \frac{1}{2}(1 + X) \tag{9}$$

Now, we want to go from the global $A_{i,j} \rightarrow \tilde{A}^e_{r,s}$, where r,s are local coordinates in the given element e. I.e we need to compute

$$\tilde{A}^e = \begin{bmatrix} (\tilde{\phi}_{0,x}, \tilde{\phi}_{0,x}) + a(\tilde{\phi}_0, \tilde{\phi}_0) & (\tilde{\phi}_{0,x}, \tilde{\phi}_{1,x}) + a(\tilde{\phi}_0, \tilde{\phi}_1) \\ (\tilde{\phi}_{1,x}, \tilde{\phi}_{0,x}) + a(\tilde{\phi}_1, \tilde{\phi}_0) & (\tilde{\phi}_{1,x}, \tilde{\phi}_{1,x}) + a(\tilde{\phi}_1, \tilde{\phi}_1) \end{bmatrix} \tag{10}$$

Here the subindex of x denotes the derivative with respect to x. Note here that the inner product is now defined over the local domain $X \in [-1, 1]$, so to transform $dx \rightarrow dX$ we need to multiply the integrand with the jacobi determinant, which here is $detJ = h_e/2$. We also need to convert the derivative of the basis functions from $d/dx \rightarrow d/dX$. To do this we note that

$$\frac{d\tilde{\phi}_r}{dx} = \frac{d\tilde{\phi}_r}{dX}\frac{dX}{dx} = \frac{2}{h_e}\frac{d\tilde{\phi}_r}{dX}$$

Thus, for a given element in the local matrix A^e we need to calculate

$$A^e_{r,s} = \int_{-1}^{1} \left[\frac{4}{h_e^2}\frac{d\tilde{\phi}_r}{dX}\frac{d\tilde{\phi}_s}{dX} + a\tilde{\phi}_r\tilde{\phi}_s \right] \frac{h_e}{2}\, dX \tag{11}$$

so our final element matrix $\tilde{A}^e$ is

$$\tilde{A}^e = \begin{bmatrix} \frac{1}{h_e} + \frac{ah_e}{3} & -\frac{1}{h_e} + \frac{ah_e}{6} \\ -\frac{1}{h_e} + \frac{ah_e}{6} & \frac{1}{h_e} + \frac{ah_e}{3} \end{bmatrix} \tag{12}$$

1.3 c)

Given that we have Neuman condition on both sides of the domain, the element matrix only changes the size of the element, i.e for the left most element we have

$$\tilde{A}^0 = \begin{bmatrix} \frac{1}{h_0} + \frac{ah_0}{3} & -\frac{1}{h_0} + \frac{ah_0}{6} \\ -\frac{1}{h_0} + \frac{ah_0}{6} & \frac{1}{h_0} + \frac{ah_0}{3} \end{bmatrix}, \tag{13}$$

and for the rightmost element we have

$$\tilde{A}^{N_e-1} = \begin{bmatrix} \frac{1}{h_{N_e-1}} + \frac{ah_{N_e-1}}{3} & -\frac{1}{h_{N_e-1}} + \frac{ah_e}{6} \\ -\frac{1}{h_{N_e-1}} + \frac{ah_{N_e-1}}{6} & \frac{1}{h_{N_e-1}} + \frac{ah_{N_e-1}}{3} \end{bmatrix}. \tag{14}$$

The element vectors for the leftmost and rightmost element is defined from equation 6. We note here that the Neuman boundary criteria is upheld if

$$\phi_i(L) = \begin{cases} 1 & \text{if } i = N_e - 1 \\ 0 & \text{else} \end{cases}$$

$$\phi_i(0) = \begin{cases} 1 & \text{if } i = 0 \\ 0 & \text{else} \end{cases}$$

thus the generalized b vector can be written as

$$b_r^e = \frac{h_e}{2} \int_{-1}^{1} f(x(X))\tilde{\phi}_r(X) + D\delta_{e,N_e-1}\delta_{r,1} - C\delta_{e,0}\delta_{r,0}$$

Using this we thus get that for the leftmost element we have that

$$b^0 = \begin{bmatrix} -C + \frac{h_0}{2}\int_{-1}^{1} f(x(X))\tilde{\phi}_0\, dX \\ \frac{h_0}{2}\int_{-1}^{1} f(x(X))\tilde{\phi}_1\, dX \end{bmatrix}.$$

and for the rightmost element we have

$$b^{N_e-1} = \begin{bmatrix} \frac{h_{N_e-1}}{2}\int_{-1}^{1} f(x(X))\tilde{\phi}_0\, dX \\ D + \frac{h_{N_e-1}}{2}\int_{-1}^{1} f(x(X))\tilde{\phi}_1\, dX \end{bmatrix}$$

1.4 d)

The number of rows in a matrix is determined by the number of elements and the choice of basis functions. We have N_e elements and uses P1 basis functions, thus each element has two nodes. We thus have that the element matrix A has $N_e + 1$ rows and $N_e + 1$ columns. The number of non-zero elements in the matrix is given by $N_z = 2 + (N_e - 1) * 3 + 2$. This can be shown visually in table 1

*	*	0	0	0
*	*	*	0	0
0	*	*	*	0
0	0	*	*	*
0	0	0	*	*

Table 1: Table showing the assembly of the element matrices into the global A matrix

Each row represents each node. For the first node we have 2 contributions, from the inner product of $\tilde{\phi}_0$ with it self and the inner product between $\tilde{\phi}_0$ and $\tilde{\phi}_1$. The next $N_e - 2$ rows has 3 contributions, given that they are boundary nodes between two elements. Thus each node will have two contributions from the basis functions in its own element, and one contribution from the inner product of the basis node and the basis function connected to it from the next element. The last row has two

contributions, with the same calculations as for the first row. Thus the number of non-zero rows is $N_z = 2 + (N_e - 1) * 3 + 2$.

The total A matrix can be assembled as shown in table 1, and the values for each non-zero element has been calculated in equation 12. Thus for a given row i in the interior points we have that

$$\begin{aligned} A_{i,i-1} &= \tilde{A}^{i-1}_{1,0} = \frac{ah_{i-1}}{6} - \frac{1}{h_{i-1}} \\ A_{i,i} &= \tilde{A}^{i-1}_{1,1} + \tilde{A}^{i}_{0,0} = \frac{ah_{i-1}}{3} + \frac{1}{h_{i-1}} + \frac{ah_i}{3} + \frac{1}{h_i} \\ A_{i,i+1} &= \tilde{A}^{i}_{0,1} = \frac{ah_i}{6} - \frac{1}{h_i}. \end{aligned} \tag{15}$$

The boundaries are given as

$$\begin{aligned} A_{0,0} &= \frac{ah_0}{6} - \frac{1}{h_0} \\ A_{0,1} &= \frac{ah_0}{3} + \frac{1}{h_0} \end{aligned} \tag{16}$$

and

$$\begin{aligned} A_{N_e,N_e-1} &= \frac{ah_{N_e-1}}{3} + \frac{1}{h_{N_e-1}} \\ A_{N_e,N_e} &= \frac{ah_{N_e-1}}{6} - \frac{1}{h_{N_e-1}} \end{aligned} \tag{17}$$

1.5 e)

We now change the left boundary condition to a so called Robin condition, given as

$$u_x = r(u - B), \tag{18}$$

where r, B are real scalars. The weak variational form from equation 20 with the Robin condition is thus.

$$\sum_j c_j \left[\int_0^L \psi_j' \psi_i' \, dx + \int_0^L a\psi_j \psi_i \, dx \right] = D\psi_i(L) - r(\sum_j c_j \psi_j(0) - B)\psi_i(0) + \int_0^L f(x)\psi_i \, dx. \tag{19}$$

Rewritten, the new weak variational form is given as

$$\sum_j c_j \left[\int_0^L \psi_j' \psi_i' \, dx + \int_0^L a\psi_j \psi_i \, dx + r\psi_j(0)\psi_i(0) \right] = D\psi_i(L) + rB\psi_i(0) + \int_0^L f(x)\psi_i \, dx. \tag{20}$$

From this equation we see that our global matrix A will add in a contribution of r in the upper left most matrix element, since $\psi_j(0)$ and $\psi_i(0)$ are 1 in only the first node, and zero elsewhere. The global b vector will add in a contribution of rB in the first vector element for the same reason as just mentioned.

2 Task 2

In this task we will look at a 2D diffusion equation, given as

$$\frac{\partial u}{\partial t} = \kappa \left[\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right] + f(u) \tag{21}$$

defined on the domain $(x, y) \in \Omega = (0, 1)^2$, with $t \in (0, T]$. The initial condition is given as

$$u(x, y, 0) = I(x, y)$$

and the boundary condition is given as the Neuman condition on all boundaries, i.e

$$\frac{\partial u}{\partial n} = 0, \quad x = 0, \quad x = 1, \quad y = 0, \quad y = 1.$$

2.1 a)

We will now use Backward Euler to discretize the time dependent part of the PDE. We will use the notation that $u(t_0 + \Delta t * n) = u^n$. We then have

$$\frac{u^{n+1} - u^n}{\Delta t} = \kappa \left[\frac{\partial^2 u^{n+1}}{\partial x^2} + \frac{\partial^2 u^{n+1}}{\partial y^2} \right] + f(u^{n+1}) \tag{22}$$

2.2 b)

To find the system of nonlinear algebraic equations we first need to discretize the second derivative with respect to x and y. Here we will use center difference method:

$$\frac{u^{n+1}_{i,j} - u^n_{i,j}}{\Delta t} = \kappa \left[\frac{u^{n+1}_{i+1,j} - 2u^{n+1}_{i,j} + u^{n+1}_{i-1,j}}{\Delta x^2} + \frac{u^{n+1}_{i,j+1} - 2u^{n+1}_{i,j} + u^{n+1}_{i,j-1}}{\Delta y^2} \right] + f(u^{n+1}_{i,j}) \tag{23}$$

This is thus the system of nonlinear algebraic equations for all i,j,n in their respective domains. Each equation is given as equation above, and there are $(N_x + 1) \times (N_y + 1) \times N_t$ equations. This is because we need to calculate the Neuman boundary conditions for each time step, as well as the internal points in space. We have an initial condition for the system as well, thus we only need to iterate over N_t temporal

points. Here $i \in \{0, 1, ..., N_x\}$ and $j \in \{0, 1, ..., N_y\}$. This will result in the use of ghost points at the boundary.

2.3 c)

Picard iterations introduce a new function $\tilde{F}(u) \approx F(u)$ to handle the non-linear terms of the equation. Here $F(u)$ is the residual of the function, i.e all terms moved to one side. In our case, we approximate the non-linear function $f(u)$ such that it is linear, but as without as much change as possible. If, for example, $f(u) = u^2$ we would approximate this as $\tilde{f}(u) = u^- u$. A crude approximation would instead be to simply use u^- for u. This would be necessary in the case of $f(u) = \sin(u) \approx \sin(u^-)$.

Then we solve the equation for a given time step, use the new solution as the new u^- until we reach a satisfying approximation for the non-linear term. This stopping criteria can for instance be if the current and previous solution has an absolute difference lower than a given tolerance, i.e $\mid u - u^- \mid \leq \epsilon_u$, or if the absolute value of the approximated residual is lower than a given tolerance, i.e $\mid \tilde{F}(u) \mid < \epsilon_r$. The only way to solve this as a matrix equation is in the crude case, where we approximate all of $f(u)$. Then we do not get a dependence of $u^(n+1)$ in the b vector, and thus it can be solved as a matrix equation. We have several unknowns:

$$\frac{u_{i,j}^{n+1}}{\Delta t} - \kappa \left[\frac{u_{i+1,j}^{n+1} - 2u_{i,j}^{n+1} + u_{i-1,j}^{n+1}}{\Delta x^2} + \frac{u_{i,j+1}^{n+1} - 2u_{i,j}^{n+1} + u_{i,j-1}^{n+1}}{\Delta y^2} \right] = \frac{u_{i,j}^{n}}{\Delta t} + \tilde{f}(u^-) \tag{24}$$

To save space, we define two new constants C_1, C_2, where

$$C_1 = \frac{\Delta t}{\Delta x^2}$$

$$C_2 = \frac{\Delta}{\Delta y^2}$$

so our equation is thus

$$u_{i,j}^{n+1}(1 + 2\kappa C_1 + 2\kappa C_2) - \kappa C_1(u_{i+1,j}^{n+1} + u_{i-1,j}^{n+1}) - \kappa C_2(u_{i,j+1}^{n+1} + u_{i,j-1}^{n+1}) = u_{i,j}^{n} + \tilde{f}(u^-)\Delta t \tag{25}$$

Given that we have two grids, one for x and y, we now have to solve for all combinations of $u_{i,j}$, such that

$$x = \begin{bmatrix} u_{0,0} \\ u_{1,0} \\ \vdots \\ u_{N_x,0} \\ u_{0,1} \\ u_{1,1} \\ \vdots \\ u_{N_x,1} \\ \vdots \\ u_{0,N_y} \\ u_{1,N_y} \\ \vdots \\ u_{N_x,N_y} \end{bmatrix}.$$

where x is $(N_x+1)(N_y+1)$ long, and $u_{i,j} = u_{i,j}^{n+1}$.

The total matrix is too large to write out, so here is a $3x3$ example. This placement applies for bigger grids aswell. To save space we define $(1 + 2\kappa C_2 + 2\kappa C_1) = Y$. Thus our matrix A is given as

$$A = \begin{bmatrix} Y & -\kappa C_1 & 0 & -\kappa C_2 & 0 & 0 & 0 & 0 & 0 \\ -\kappa C_1 & Y & -\kappa C_1 & 0 & -\kappa C_2 & 0 & 0 & 0 & 0 \\ 0 & -\kappa C_1 & Y & 0 & 0 & -\kappa C_2 & 0 & 0 & 0 \\ -\kappa C_2 & 0 & 0 & Y & -1 & 0 & -\kappa C_2 & 0 & 0 \\ 0 & -\kappa C_2 & 0 & -\kappa C_1 & Y & -\kappa C_1 & 0 & -\kappa C_2 & 0 \\ 0 & 0 & -\kappa C_2 & 0 & -\kappa C_1 & Y & 0 & 0 & -\kappa C_2 \\ 0 & 0 & 0 & -\kappa C_2 & 0 & 0 & Y & -\kappa C_1 & 0 \\ 0 & 0 & 0 & 0 & -\kappa C_2 & 0 & -\kappa C_1 & Y & -\kappa C_1 \\ 0 & 0 & 0 & 0 & 0 & -\kappa C_2 & 0 & -\kappa C_1 & Y \end{bmatrix}$$

This can be generalized to $(N_x+1)(N_y+1)$ rows and columns. One must however be careful with the matrix elements corresponding to where the index reverts back. In other words

$$x = \begin{bmatrix} \vdots \\ u_{N_x,0} \\ u_{0,1} \\ \vdots \end{bmatrix}.$$

Here one needs to be aware of the $i, j \pm 1$ and which constant that will contribute.

In the $3x3$ case, our x vector is given as

$$x = \begin{bmatrix} u_{0,0} \\ u_{1,0} \\ u_{2,0} \\ u_{0,1} \\ u_{1,1} \\ u_{2,1} \\ u_{0,2} \\ u_{1,2} \\ u_{2,2} \end{bmatrix}$$

and b vector is given as

$$b = \begin{bmatrix} u^p_{0,0} + f(u^p_{0,0}) \\ u^p_{1,0} + f(u^p_{1,0}) \\ u^p_{2,0} + f(u^p_{2,0}) \\ u^p_{0,1} + f(u^p_{0,1}) \\ u^p_{1,1} + f(u^p_{1,1}) \\ u^p_{2,1} + f(u^p_{1,2}) \\ u^p_{0,2} + f(u^p_{0,2}) \\ u^p_{1,2} + f(u^p_{1,2}) \\ u^p_{2,2} + f(u^p_{2,2}) \end{bmatrix}$$

where $u^p_{i,j}$ is the notation used for the previous timestep.

2.4 d)

Newton's method is originally a method to find a function's root. In our case, that function is the residual $F(u)$, and the root is the approximated value u^- that most satisfies the original problem. In this case we have that

$$F(u) = -\frac{u^{n+1}_{i,j} - u^n_{i,j}}{\Delta t} + \kappa \left[\frac{u^{n+1}_{i+1,j} - 2u^{n+1}_{i,j} + u^{n+1}_{i-1,j}}{\Delta x^2} + \frac{u^{n+1}_{i,j+1} - 2u^{n+1}_{i,j} + u^{n+1}_{i,j-1}}{\Delta y^2} \right] + f(u^{n+1}_{i,j}) \tag{26}$$

Newton's method will thus be

$$u_{i,j}^{n+1,k+1} = u^{-,k} - \frac{F(u^{-,k})}{F'(u^{-,k})}$$

$$u_{i,j}^{n+1} = u^{-} - \frac{-u^{-} + u_{i,j}^{n} + \kappa C_1(u_{i+1,j}^{n+1} - 2u^{-} + u_{i-1,j}^{n+1}) + \kappa C_2(u_{i,j+1}^{n+1} - 2u^{-} + u_{i,j-1}^{n+1}) + f(u^{-})\Delta t}{-1 - 2\kappa C_1 - 2\kappa C_2 + f'(u^{-})\Delta t}$$

For convenience I drop the k index in the second line in the equation above. This equation we be calculated K times until we reach a satisfied solution for that time step. For the next step, the first u^{-} will be set to the previous time step final solution, i.e $u^{-,k=0} = u^{n}$. This is repeated for all N_t time steps.

The system of linear algebraic equations can then be written as a matrix equation. Lets start by first defining $\xi = -1 - 2\kappa C_1 - 2\kappa C_2 + f'(u^{-})\Delta t$. Then we gather all the unknowns on the left side:

$$u_{i,j}^{n+1}\xi + \kappa C_1(u_{i+1,j}^{n+1} + u_{i-1,j}^{n+1}) + \kappa C_2(u_{i,j+1}^{n+1} + u_{i,j-1}^{n+1}) = u^{-}\xi - Yu^{-} - u_{i,j}^{n} + f(u^{-})\Delta t$$

where Y is the same as in task 2c. Note here that $\xi - D = f'(u^{-})$, so we then get

$$u_{i,j}^{n+1}\xi + \kappa C_1(u_{i+1,j}^{n+1} + u_{i-1,j}^{n+1}) + \kappa C_2(u_{i,j+1}^{n+1} + u_{i,j-1}^{n+1}) = -u^{-}f'(u^{-}) - u_{i,j}^{n} + f(u^{-})\Delta t$$

In the $3x3$ case, our A matrix and x, b vector are given as

$$A = \begin{bmatrix} \xi & \kappa C_1 & 0 & \kappa C_2 & 0 & 0 & 0 & 0 & 0 \\ \kappa C_1 & \xi & \kappa C_1 & 0 & \kappa C_2 & 0 & 0 & 0 & 0 \\ 0 & \kappa C_1 & \xi & 0 & 0 & \kappa C_2 & 0 & 0 & 0 \\ \kappa C_2 & 0 & 0 & \xi & \kappa C_1 & 0 & \kappa C_2 & 0 & 0 \\ 0 & \kappa C_2 & 0 & \kappa C_1 & \xi & \kappa C_1 & 0 & \kappa C_2 & 0 \\ 0 & 0 & \kappa C_2 & 0 & \kappa C_1 & \xi & 0 & 0 & \kappa C_2 \\ 0 & 0 & 0 & \kappa C_2 & 0 & 0 & \xi & \kappa C_1 & 0 \\ 0 & 0 & 0 & 0 & \kappa C_2 & 0 & \kappa C_1 & \xi & \kappa C_1 \\ 0 & 0 & 0 & 0 & 0 & \kappa C_2 & 0 & \kappa C_1 & \xi \end{bmatrix}$$

$$x = \begin{bmatrix} u_{0,0} \\ u_{1,0} \\ u_{2,0} \\ u_{0,1} \\ u_{1,1} \\ u_{2,1} \\ u_{0,2} \\ u_{1,2} \\ u_{2,2} \end{bmatrix}$$

Here we use the notation that $u_{i,j} = u_{i,j}^{n+1}$.

$$b = \begin{bmatrix} -f'(u_{0,0}^p)u_{0,0}^p - u_{0,0}^p + f(u_{0,0}^p) \\ -f'(u_{1,0}^p)u_{01,0}^p - u_{1,0}^p + f(u_{1,0}^p) \\ -f'(u_{2,0}^p)u_{2,0}^p - u_{2,0}^p + f(u_{2,0}^p) \\ -f'(u_{0,1}^p)u_{0,1}^p - u_{0,1}^p + f(u_{0,1}^p) \\ -f'(u_{1,1}^p)u_{1,1}^p - u_{1,1}^p + f(u_{1,1}^p) \\ -f'(u_{2,1}^p)u_{2,1}^p - u_{2,1}^p + f(u_{1,2}^p) \\ -f'(u_{0,2}^p)u_{0,2}^p - u_{0,2}^p + f(u_{0,2}^p) \\ -f'(u_{1,2}^p)u_{1,2}^p - u_{1,2}^p + f(u_{1,2}^p) \\ -f'(u_{2,2}^p)u_{2,2}^p - u_{2,2}^p + f(u_{2,2}^p) \end{bmatrix}$$

This can again be generalized to $(N_x + 1)(N_y + 1)$ rows and columns.

2.5 e)

To find the ideal values for Δy, Δx and Δt we will perform a gridsearch. There are an assumption we need to make here. Even though the task claims that the spacial grid can be of different size, we assume that they are the same. Next we need an estimate of how long the computation will take. We know that our computation should take S hours, given that our friend's computer is S times faster, and he has 1 hour of computation time. We also know that this should be equal to the time each time step computation takes. Thus we have

$$T_{tot} = T_{friend}S = T_{\Delta t}N_t = \frac{T_{\Delta t}T_{sim}}{\Delta t}$$

The time each time step takes to compute is dependent on the grid size of the matrix inversion, as well as the number of iterations from Newton's method. This relation is given as

$$T_{\Delta t} = T_{A_{inv}}(\Delta x, \Delta y)k,$$

where $T_{A_{inv}}$ is the time it takes to invert the matrix A. Now, we first need to estimate $T_{A_{inv}}$. We know that it is proportional to Δx, and thus also Δy, i.e $T_{A_{inv}} = U\Delta x^{\gamma}$. To find these two parameters we first calculate the time it takes to calculate each grid for varying grid sizes and create a log log plot. γ will then be the time convergence. Having found a term for $T_{A_{inv}}$, allows us to narrow the grid search down. We have that

$$k = \frac{T_{sim}}{T_{friend}S\Delta t}U\Delta x^{\gamma} \tag{27}$$

Our problem has now been reduced to a problem with two unknowns, Δx and Δt. Picking a combination of the two gives us via equation 27 the number of iterations for Newton's method. With these three variables we can calculate the error. Now if we have an exact solution, we could just use the L2 norm to estimate the error. If we however do not have an exact solution we will have to use mean squared error of the residual as a measure of the error. Here we get a resulting global minimum problem, where the minimum gives us the ideal Δt, k and $\Delta x = \Delta y$. Note that this would take a very long time to compute in its entirety. To reduce the computation time, we could cut off the calculation at a given time step, and assume that the error would not suddenly increase in the calculations after.