Nonlinear diffusion equation

Andreas V. Solbrå

University of Oslo

Department of Computational Physics

December 13, 2012

Abstract

The goal of this project is to discuss various numerical aspects of a nonlinear diffusion model:

$$\varrho u_t = \nabla \cdot (\alpha(u)\nabla u) + f(x,t) \tag{1}$$

With initial condition $u(x,0) = I(x)$ and boundary condition $\partial u/\partial n = 0$. The coefficient ϱ is constant and $\alpha(u^n)$ is a known function of u.

1 Implicit time scheme

We begin by inserting an implicit finite difference method in time. For simplicity we will use the Backward Euler Scheme. In operator form we may write this as

$$[\varrho D_t u = \nabla \cdot (\alpha(u)\nabla u) + f(x,t)]^n. \tag{2}$$

And written out explicitly we get

$$\varrho \frac{u^n - u^{n-1}}{\Delta t} = \nabla \cdot (\alpha(u^n)(\nabla u)^n) + f(x,t^n) \tag{3}$$

or

$$u^n - \frac{\Delta t}{\varrho} \left(\nabla \cdot (\alpha(u^n)(\nabla u)^n) + f(x,t^n) \right) = u^{n-1} \tag{4}$$

2 Variational form by Galerkin method

Now let $\{v_i\}_{i=1}^N$ be some space of functions from which we want to construct an approximate solution to the partial differential equation (PDE), and let $\langle f,g \rangle$ be the standard inner product for real integrable functions,

$$\langle f,g \rangle = \int_\Omega f(x)g(x)\,d\Omega, \tag{5}$$

1

where Ω is the integration domain. The variational formulation of the PDE is then to find a solution with respect to the space spanned by $\{v_i\}$, i.e. we want to find a function u such that

$$\left\langle u^n - \frac{\Delta t}{\varrho} \left(\nabla \cdot (\alpha(u^n)(\nabla u)^n) + f(x, t^n) \right), v_i \right\rangle = \langle u^{n-1}, v_i \rangle \tag{6}$$

for all i. Using linearity of the inner product we may write this out as

$$\int_\Omega u^n(x) v_i(x) \, d\Omega - \frac{\Delta t}{\varrho} \int_\Omega \left(\nabla \cdot (\alpha(u^n)(\nabla u(x))^n) + f(x, t^n) \right) v_i(x) \, d\Omega = \int_\Omega u^{n-1}(x) v_i(x) \, d\Omega \tag{7}$$

Where we can use partial integration of the second term to write it as

$$\int_\Omega \nabla \cdot (\alpha(u^n) \nabla u(x)^n) v(x) \, d\Omega = \int_{\partial\Omega} \alpha(u^n) \frac{\partial u}{\partial n} v(x) \, ds - \int_\Omega \alpha(u^n) \nabla u^n(x) \cdot \nabla v_i(x) \, d\Omega \tag{8}$$

$$= - \int_\Omega \alpha(u^n) \nabla u^n(x) \cdot \nabla v_i(x) \, d\Omega \tag{9}$$

where the first term disappears as we are looking at the boundary condition $\partial u / \partial n = 0$.

In order to solve this system of equations, we expand u^n in the basis $\{v_i\}$, so that

$$u^n = \sum_{i=1}^{N} c_i^n v_i. \tag{10}$$

For the initial position $u^0(x) = I(x)$, we find the closest approximation to $I(x)$ in $\{v_i\}$ by the Galerkin method as

$$\left\langle \sum_i c_j^0 v_j - I, v_i \right\rangle = 0 \tag{11}$$

for all i, which translates to the set of equations

$$\sum_{j=0}^{N} c_j^0 \langle v_j, v_i \rangle = \langle I, v_i \rangle, \quad i = 0, 1, \ldots, N, \tag{12}$$

which gives us a set of equations in order to determine c_j^0. If we introduce the matrix M where $M_{ij} = \langle v_i, v_j \rangle$, and the vector $\mathbf{I}$, where $\mathbf{I}_i = \langle I, v_i \rangle$ and the vector $\mathbf{c}^n$ where $\mathbf{c}_i^n = c_i^n$, we may write equation (12) as

$$M\mathbf{c}^0 = \mathbf{I}. \tag{13}$$

This allows us to determine $\mathbf{c}^0$, and with $\mathbf{c}^0$ known, we should be able to determine $\mathbf{c}^1$ from the PDE by requiring

$$\int_\Omega u^n(x) v_j(x) + \frac{\Delta t}{\varrho} \left(\alpha(u^n) \nabla u^n(x) \cdot \nabla v_j(x) \right) d\Omega = \int_\Omega (u^{n-1}(x) + f(x, t^n)) v_j(x) \, d\Omega \tag{14}$$

for all j. Here all the terms on the right hand side are known, while the terms on the left hand side are generally not. An obvious problem that arises is that we have so far not paid much attention to the factor $\alpha(u^n)$, which may be impossible to incorporate exactly if it has a complicated form. We will deal with this in the next section.

3   Picard iteration for estimation nonlinear coefficients

Piccard iteration is a very simple fixed-point method for solving nonlinear equations on the form

$$f(u)u = b \tag{15}$$

where $f(u)$ is some nonlinear function. The iteration is done by first coming up with some good guess for u, and then approximating the solution through solving the equation

$$f(u_n)u_{n+1} = b, \tag{16}$$

which is linear in u_{n+1}.

The way we implement this into our PDE-solver is that we initially calculate the solution of the case where we let $\alpha(u) = \alpha(u^{n-1})$. this gives us an approximated solution u_1^n. We can then go back and calculate the approximated solution again, but this time we let $\alpha(u) = \alpha(u_1^n)$, which will give us a new and hopefully better approximationg to the solution. This procedure can be repeated, and the approximations u_i^n will converge.

4   FEniCS implementation

For a FEniCS implementation, we must first recognize the terms $L(v)$ and $a(u, v)$, which are the known and unknown parts of out variational problem, respectively. That is,

$$a(u, v) = \int_\Omega u(x)v(x) + \frac{\Delta t}{\varrho}\left(\alpha(u^n)\nabla u(x) \cdot \nabla v(x)\right) d\Omega \tag{17}$$

$$L(v) = \int_\Omega (u^{n-1}(x) + \frac{\Delta t}{\varrho}f(x, t^n))v_j(x) \, d\Omega \tag{18}$$

We will in this implementation use a single Piccard iteration, that is, simply use the previous u as an approximation for the current u. An implementation then may look like this:

```
 from PDE_tools import *

def alpha(u):
    a = 1.
    return a

#Physical constants:
rho = 1.

# Time discretisation:
T = 1.
dt = 0.001

xnodes, ynodes = 6,6
```

```
mesh = UnitSquare(xnodes, ynodes)
V = FunctionSpace(mesh, 'Lagrange', 1)
u0 = Expression('exp(-pi*pi*t)*cos(pi*x[0])',pi=pi,t=0)
u0.t = 0
u_p = project(u0, V)
u = TrialFunction(V)
v = TestFunction(V)
f = Constant(0.0)
a = u*v*dx + dt/rho*alpha(u_p)*inner(nabla_grad(u), nabla_grad(v))*dx
L = (u_p + dt/rho*f)*v*dx
u = Function(V)
A = assemble(a)
u = Function(V)
b = None
c = None
t = dt
while t<= T:
    b = assemble(L,tensor=b)
    c = assemble(a,tensor=c)
    u0.t = t
    solve(a == L,u)
    t += dt
    u_p.assign(u)
    u_e = interpolate(u0, V)
```

5   Verification of the code

There are several ways to test our code, and we will go through some basic tests

5.1   Convergence rate test

Using the Backward Euler time discretisation, the error should be of order $\mathcal{O}(\Delta x^2) + \mathcal{O}(\Delta y^2) + \mathcal{O}(\Delta t)$. If we set $\Delta t = \Delta x^2 = \Delta y^2 = h$ for many different values of h, then if we denote the error by E, E/h should remain approximately constant. The error can be computed as

```
e = u_e.vector().array() - u.vector().array()
E = np.sqrt(np.sum(e**2)/u.vector().array().size)
```

and we may test our code by looping over a number of different h, and check if E/h remains relatively constant. In our implementation, we will let $\Omega = [0,1]^2$, $\alpha(u) = 1$, P1 elements, and $I(x,y) = \cos(\pi x)$. The exact solution is then $u(x,y,t) = e^{-pi^2 t}\cos(\pi x)$. An implementation of this test may look like the following:

```
def test_convergence_rate():
    h = 0.1
```

Time	Error
0.1	3.07157973e-07
0.5	1.84039444e-05
1.0	8.81135316e-05
2.0	3.39763715e-04
3.0	7.72650525e-04

Table 1: This table shows the total error for different times, using parameters as specified in the text.

```
T = 1.
errorlist = []
for i in range(7):
    dt = h
    xnodes = ynodes = int(1/sqrt(h))
    u_e, u = run_main(xnodes, ynodes, dt, T)
    e = u_e.vector().array() - u.vector().array()
    E = np.sqrt(np.sum(e**2)/u.vector().array().size)
    errorlist.append(E/h)
    h = h/2
errordiff = []
for i in range(6):
    errordiff.append(errorlist[i+1]/errorlist[i])

return errordiff
```

And this test, for our code gives the following result for the errors:

```
user@UNIX:~/../prosjekt$ python nonlinear_diffusion.py
...
[0.85586086798890837, 0.62980350073545388, 0.65710001385587757,
0.93447713096923124, 0.97493810537888737, 0.97357406967548188]
```

And this appears to be converging, as expected.

5.2 Method of manufactured solution

To get an indication whether the implementation of the nonlinear diffusion PDE is correct or not, we can use the method of manufactured solutions. Say we restrict the problem to one space dimension, $\Omega = [0, 1]$, and choose $u(x, t) = tx \int_0^x q(1q)dq = tx^2(1/2x/3)$ and $\alpha(u) = 1 + u^2$. For this choice, we can show that $u(x, t)$ is a solution if $f(x, t) = -\varrho x^3/3 + \varrho x^2/2 + 8t^3 x^7/9 - 28t^3 x^6/9 + 7t^3 x^5/2 - 5t^3 x^4/4 + 2tx - t$. We can test the program against the analytical solution for different final times, and for 31 nodes and a time step of 0.01, the results for the total error E are shown in table 1, and the local error is shown in figure 1
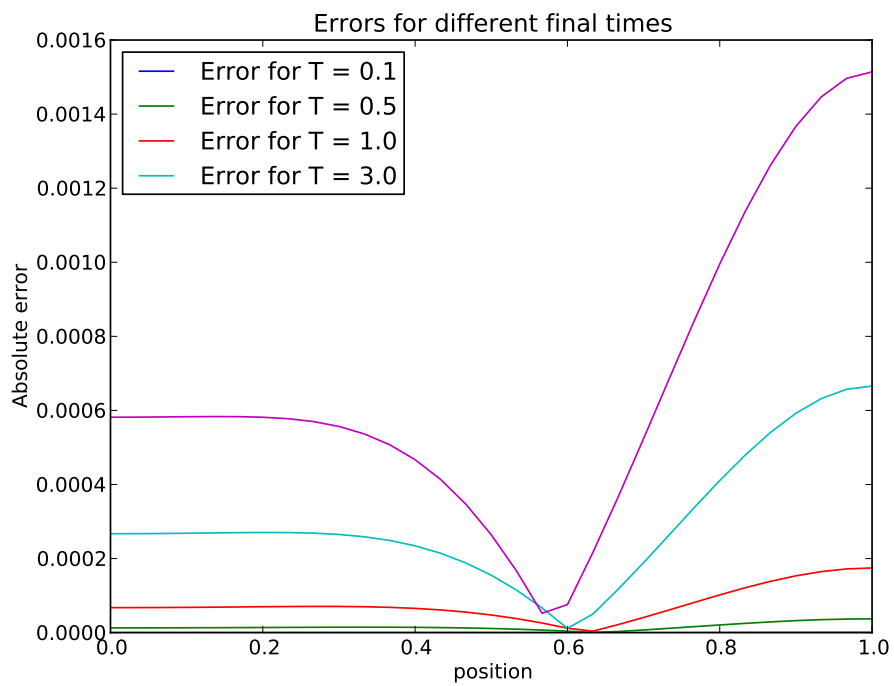
5

Figure 1: The figure shows the error for different times, using parameters as specified in the text.

Time	Error
0.1	3.20008488e-10
0.5	9.51625737e-08
1.0	1.33755315e-06
2.0	2.05114946e-05
3.0	1.04725019e-04

Table 2: The table shows the total error for different times using the alternate scheme for $f(x,t)$, as described in the text, using parameters as specified in the text. The errors are generally smaller than those in table 1.

6   Sources of errors in the solution

There are several important sources of errors, notably

- We use $\alpha(u^{n-1})$ instead of $\alpha(u^n)$ in our calculations of $a(u,v)$. This could be improved for instance by using several Piccard iterations.

- Truncation error from approximating the function with piecewise linear elements.

- Other sources such as error due to the numerical integration in FEniCS, etc, these should be considered minor.

- The time discretisation is also a source of error, however this is not explicitly due to FEniCS.

6.1   Elimination of the error due to a single Piccard iteration

In this section, we will choose f such that our choice for u fulfills equation (1) with $\alpha(u^{n-1})$ rather than $\alpha(u^n)$. This will eliminate a major source of error, and will allow us to perform a convergence rate test for the manufactured solution. For this case, one can show that $f(x,t) = \rho x^2(-2x+3)/6 - (-12tx + 3t(-2x+3))(x^4(-dt+t)^2(-2x+3)^2 + 36)/324 - (-6tx^2 + 6tx(-2x+3))(36x^4(-dt+t)^2(2x-3) + 36x^3(-dt+t)^2(-2x+3)^2)/5832$, and if we select the same parameters as we did previously, we find errors as shown in table 2, and a convergence test as performed previously gives the result

```
user@UNIX:~/../prosjekt$ python nonlinear_diffusion.py
...
[1.9583957954613822, 1.9612382435719218, 1.980102551522378,
1.981220449666488, 1.988819119920602, 1.9914942289734918]
```

which again appears to converge.

7 Nonlinear diffusion of a Gaussian function

We will now simulate the diffusion of a gaussian function in 2D. Due to symmetry, it suffices to simulate one quarter of the domain:

$$I(x, y) = \exp(\frac{1}{2^2}(x^2 + y^2)), \quad (x, y) \in \Omega = [0, 1] \times [0, 1], \tag{19}$$

where σ measures the width of the Gaussian function. We choose $\alpha(u) = 1 + \beta u^2$, where β is a constant one can play around with.

7.1 Animating the solution in MayaVI

The 2D gaussian diffusion is a good cause to explore more powerful animation tools than the standard matplotlib library. MayaVI is a very powerful tool for visualization of 2D-functions, and can be used very similarly to other well known tools such as matplotlib and MATLAB. the function `mcrtmv` can be found in the program `PDE_tools.py` at www.github.com/andreavs/INF5620/pro along with an example in the file `diffusion.mpg`.

8 Handling the nonlinear term

In this section we will look at some other methods of handling the linear term.

8.1 Group finite element method

The group finite element method is another way to handle the nonlinear term $\alpha(u)$. In the finite element method we approximate u^n in terms of a set of functions, $u^n = \sum_j c_j^n v_j$. We now assume α is linear in these, so that

$$\alpha(u^n) = \alpha \left(\sum_j c_j^n v_j \right) \approx \sum_j \alpha(c_j) v_j. \tag{20}$$

This will assure a linear equation in u^n! If we now go back to equation (14), we can insert the expression for u^n and the group finite element method to get

$$\int_\Omega \sum_j c_j^n v_j(x) v_i(x) + \frac{\Delta t}{\varrho} \left(\sum_j \alpha(c_j^n) v_j \sum_k c_k^n \nabla v_k(x) \cdot \nabla v_i(x) \right) d\Omega = \int_\Omega (\sum_j c_j^{n-1} v_j(x) + f(x, t^n)) v_i(x) \, d\Omega \tag{21}$$

for all i. This can in principle be solved for c_j^n, as we have N equations for the N unknowns.

8.2 Newtons's method

Another way to solve for the nonlinear term is to use Newtons method. With $a(u, v)$ and $L(v)$ defined as previously in equations (17)-(18), we can write the problem as

$$a(u, v) - L(v) = 0 \tag{22}$$

$$F(u, v) = 0 \quad \forall v \in V \tag{23}$$

where $F(u, v) = a(u, v) - L(v)$. If we now insert $u^n = \sum_j c_j^n v_j$. and $v = v_i$, we get the equation

$$F_i(u, v) = a\left(\sum_j c_j^n v_j, v_i\right) - L(v_i) \tag{24}$$

$$= \int_\Omega \sum_j c_j^n v_j(x)v_i(x) + \frac{\Delta t}{\varrho}\left(\alpha\left(\sum_j c_j^n v_j\right)\sum_k c_k^n \nabla v_k(x) \cdot \nabla v_i(x)\right) d\Omega \tag{25}$$

$$- \int_\Omega (\sum_j c_j^{n-1} v_j(x) + f(x, t^n))v_i(x) \, d\Omega = 0 \tag{26}$$

for all i. For such a vector equation, Newtons method can be formulated as

$$J_F(\mathbf{c}_i^n)(\mathbf{c}_{i+1}^n - \mathbf{c}_i^n) = -\mathbf{F}(\mathbf{c}_i^n) \tag{27}$$

Which can be iterated and should converge to the correct solution $\mathbf{c}^n$, where J_F is the Jacobian for F, such that $(J_F)_{ij} = \partial F_i / \partial c_j$. This is a linear system of equations, and in order to solve it, we should calculate $(J_F)_{kl}$. If we note that

$$\frac{\partial u}{\partial c_l} = \frac{\partial}{\partial c_l}\sum_j c_j v_j = v_l \tag{28}$$

We have that

$$(J_F)_{kl} = \frac{\partial F_k}{\partial c_l} \tag{29}$$

$$= \int_\Omega v_k v_l + \frac{\Delta t}{\varrho}\left(\frac{\partial \alpha}{\partial u}\bigg|_{u=u_i^n} v_l(x)\sum_j c_j^n \nabla v_j(x) \cdot \nabla v_k(x) + \alpha(u_i^n)c_l \nabla v_l(x) \cdot \nabla v_k(x)\right) d\Omega \tag{30}$$

If we now reduce the problem to one space dimension and using P1 elements, and integrating with the trapezoidal rule (using the node endpoints as integration points) we have that

$$\int_\Omega v_k v_l \, d\Omega = \frac{2h}{3}\delta_{k,l} \tag{31}$$

9

which can easily be checked. The other term requires a bit more work. We first note that $(\partial v_j/\partial x)(\partial v_k/\partial x) = 0$ unless $j = k$ or $j = k \pm 1$. In these cases we have

$$\frac{\partial v_k}{\partial x}\frac{\partial v_k}{\partial x} = \begin{cases} h^{-2} & \text{if } x_{k-1} \leq x < x_{k+1} \\ 0 & \text{otherwise} \end{cases} \tag{32}$$

and

$$\frac{\partial v_k}{\partial x}\frac{\partial v_{k\pm 1}}{\partial x} = \begin{cases} -h^{-2} & \text{if } x \in [x_k, x_{k\pm 1}) \\ 0 & \text{otherwise} \end{cases} \tag{33}$$

Using this we may write the sum over j as

$$\sum_j c_j^n \frac{\partial v_j}{\partial x}\frac{\partial v_k}{\partial x} = \begin{cases} \frac{c_k - c_{k-1}}{h^2} & \text{if } x_{k-1} \leq x < x_k \\ \frac{c_k - c_{k+1}}{h^2} & \text{if } x_k \leq x < x_{k+1} \\ 0 & \text{otherwise} \end{cases} \tag{34}$$

Using this, we note that the full product including the v_l term will have a nonzero integral only if $l = k$ or $l = k \pm 1$. In total, using the trapezoidal rule, we have

$$\int_\Omega \frac{\partial \alpha}{\partial u}\bigg|_{u=u_i^n} v_l(x) \sum_j c_j^n \nabla v_j(x) \cdot \nabla v_k(x)\, d\Omega = \begin{cases} \frac{2}{3}\alpha'(u_{k-1})(c_k - c_{k-1}) & \text{if } l = k-1 \\ \frac{1}{3}\alpha'(u_k)(2c_k - c_{k+1} - c_{k-1}) & \text{if } l = k \\ \frac{2}{3}\alpha'(u_{k-1})(c_k - c_{k+1}) & \text{if } l = k+1 \\ 0 & \text{otherwise} \end{cases} \tag{35}$$

And finally, from the $\alpha(u)c_l \nabla v_l(x) \cdot \nabla v_k(x)$ term we get

$$\int_\Omega \alpha(u)c_l \nabla v_l(x) \cdot \nabla v_k(x)\, d\Omega = \begin{cases} -\alpha(u_{k-1})c_{k-1}/h & \text{if } l = k-1 \\ 2\alpha(u_k)c_k/h & \text{if } l = k \\ -\alpha(u_{k+1})c_{k+1}/h & \text{if } l = k+1 \\ 0 & \text{otherwise} \end{cases} \tag{36}$$

Gathering all the results, we find that

$$(J_F)_{kl} = \begin{cases} \frac{\Delta t}{\varrho}\left(\frac{2}{3}\alpha'(u_{k-1})(c_k - c_{k-1}) - \alpha(u_{k-1})c_{k-1}/h\right) & \text{if } l = k-1 \\ \frac{2h}{3} + \frac{\Delta t}{\varrho}\left(\frac{1}{3}\alpha'(u_k)(2c_k - c_{k+1} - c_{k-1}) + 2\alpha(u_k)c_k/h\right) & \text{if } l = k \\ \frac{\Delta t}{\varrho}\left(\frac{2}{3}\alpha'(u_{k-1})(c_k - c_{k+1}) - \alpha(u_{k+1})c_{k+1}/h\right) & \text{if } l = k+1 \\ 0 & \text{otherwise} \end{cases} \tag{37}$$

where all the c's and u's are naturally evaluated at the time step n.

8.2.1   Comparrison with a finite difference scheme

In this final section we will compare the results obtained with Newton's method to the equations arising from using a finite difference scheme. For the time discretisation we use the same Backward Euler scheme, and for the space we use the symmetric scheme

$$D_x(\alpha(u)D_x u) + f(x,t) \to D_x\left(\alpha(u)\frac{u_{i+1/2} - u_{i-1/2}}{\Delta x}\right) + f(x_i, t) \tag{38}$$

$$\to \alpha(u_{i+1/2})\left(\frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2}\right) + \alpha(u_{i-1/2})\left(\frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2}\right) + f(x_i, t) \tag{39}$$

So that our discretized PDE can be written as

$$u_i^n - \frac{\Delta t}{\varrho \Delta x^2}\left((\alpha(u_{i+1/2}^n) + \alpha(u_{i-1/2}^n))\left(u_{i+1}^n - 2u_i^n + u_{i-1}^n\right) + f(x_i, t^n)\right) = u_i^{n-1} \tag{40}$$

And we can write this as

$$M\mathbf{u}^n - \mathbf{u}^{n-1} + \frac{\Delta t}{\varrho \Delta x^2}\mathbf{f}(\mathbf{x}, t) = 0, \tag{41}$$

Where

$$M_{ij} = \begin{cases} \frac{-\Delta t}{\varrho \Delta x^2}(\alpha(u_{i+1/2}^n) + \alpha(u_{i-1/2}^n)) & \text{if } i = j - 1 \\ 1 + \frac{2\Delta t}{\varrho \Delta x^2}(\alpha(u_{i+1/2}^n) + \alpha(u_{i-1/2}^n)) & \text{if } i = j \\ \frac{-\Delta t}{\varrho \Delta x^2}(\alpha(u_{i+1/2}^n) + \alpha(u_{i-1/2}^n)) & \text{if } i = j + 1 \\ 0 & \text{otherwise} \end{cases} \tag{42}$$

The α terms makes this once again non-linear in u^n, and so we can find an iterated solution by Newton's method,

$$J_F(\mathbf{u}_i^n)(\mathbf{u}_{i+1}^n - \mathbf{u}_i^n) = -\mathbf{F}(\mathbf{u}_i^n) \tag{43}$$

where $\mathbf{F}(\mathbf{u}_i^n)$ is precisely the left hand side of equation (41). The Jacobian will in this case be

$$(J_F)_{kl} = \frac{\partial F_k}{\partial u_l} \tag{44}$$

$$= \frac{\partial}{\partial u_l}\sum_j M_{jk}u_k \tag{45}$$

At this point it will be useful to relate $\alpha u_{i+1/2}$ to the integer indices. We make the approximation $\alpha(u_{i+1/2}) \approx (\alpha(u_i) + \alpha(u_{i+1}))/2$. In this way, the Jacobian becomes

$$(J_F)_{kl} = \sum_j \frac{\partial M_{jk}}{\partial u_l}u_k + \sum_j M_{jl}\delta_{kl} \tag{46}$$

and we have that the sum over j only contributes when $|j - k| \leq 1$. Also

$$\sum_j \frac{\partial M_{jk}}{\partial u_l} = \begin{cases} \frac{-\Delta t}{\varrho \Delta x^2} \alpha'(u_{k-1}) & \text{if } k = l - 1 \\ \frac{2\Delta t}{\varrho \Delta x^2} \alpha'(u_k) & \text{if } k = l \\ \frac{-\Delta t}{\varrho \Delta x^2} \alpha'(u_{k+1}) & \text{if } k = l + 1 \end{cases} \tag{47}$$

So that in total,

$$(J_F)_{kl} \begin{cases} \frac{-\Delta t}{\varrho \Delta x^2} \alpha'(u_{k-1}) u_k & \text{if } k = l - 1 \\ 1 + \frac{2\Delta t}{\varrho \Delta x^2} (\alpha(u_{k+1}^n) + 2\alpha(u_k) + \alpha(u_{k-1})) + \frac{2\Delta t}{\varrho \Delta x^2} \alpha'(u_k) u_k & \text{if } k = l \\ \frac{-\Delta t}{\varrho \Delta x^2} \alpha'(u_{k+1}) u_k & \text{if } k = l + 1 \end{cases} \tag{48}$$