# INF5620 - Numerical methods for partial differential equations

Project 2

Even Marius Nordhagen

November 6, 2017

- For the Github repository containing programs and results, follow this link: `https://github.com/UiO-INF5620/INF5620-evenmn/tree/master/project_2`

## 1 Introduction

The aim of this project is to solve the non-linear diffusion equation

$$\varrho\frac{\partial u}{\partial t} = \nabla\cdot\Big(\alpha(u)\nabla u\Big) + f(\vec{x},t) \tag{1}$$

with an implicit scheme using FEniCS. Compared with explicit schemes, implicit schemes are supposed to be more stable, but the downside is that the implementation is messier. More specific we are applying Crank-Nicolson scheme with Neumann condition on the boundaries, and it turns out that this condition simplifies the variation formulation. Furthermore the solution is approximated by Picard iterations, which is detailed in the theory part, and we will study how many iteration is needed to get a satisfying error. Both constant, linear and non-linear solutions i 1, 2 and 3 dimensions are taken into account, and the errors and solution plots are presented in results.

# 2 Theory and methods

The theory part is mainly divided into two parts; initially we find the variational formulation, and finally we are approaching the solution by using Picard iterations.

## 2.1 Variational formulation

We first need to transform the PDE to a computer-friendly form, i.e. discretize. For approximating the time differential, we use Crank-Nicolson,

$$\frac{u^{n+1} - u^n}{\Delta t} = f_i^{n+1/2}, \tag{2}$$

which is an implicit method. This gives the discrete equation

$$\varrho \frac{u^{n+1} - u^n}{\Delta t} = \nabla \cdot (\alpha(u^{n+1/2})\nabla u^{n+1/2}) + f(u^{n+1/2}) \tag{3}$$

where we can use an arithmetic mean to approximate the RHS terms

$$\alpha(u^{n+1/2})\nabla u^{n+1/2} = \frac{1}{2}(\alpha(u^n)\nabla u^n + \alpha(u^{n+1})\nabla u^{n+1}) \tag{4}$$

$$f(u^{n+1/2}) = \frac{1}{2}(f(u^{n+1}) + f(u^n)). \tag{5}$$

We can now push all the known terms to the right-hand side (RHS) (or pull all the unknowns to the left-hand side (LHS) if you like):

$$u^{n+1} - \frac{\Delta t}{2\varrho}\nabla(\alpha(u^{n+1})\nabla u^{n+1}) - \frac{\Delta t}{2\varrho} f(u^{n+1}) = u^n + \frac{\Delta t}{2\varrho}\nabla(\alpha(u^n)\nabla u^n) + \frac{\Delta t}{2\varrho} f(u^n).$$

From now on we will use the most recently computed solution in the coefficient $\alpha$, and it will no longer be specified. Additionally we will omit the $n + 1$ superscript to make the calculations neater (but remember $u$ is still what we want to find). When we multiply $v$ and integrate on both sides we therefore obtain

$$\int \left(u - \frac{\Delta t}{2\varrho}\nabla(\alpha\nabla u) - \frac{\Delta t}{2\varrho} f(u)\right) v dx = \int \left(u^n + \frac{\Delta t}{2\varrho}\nabla(\alpha\nabla u^n) + \frac{\Delta t}{2\varrho} f(u^n)\right) v dx$$

where $v$ lies in the function space $V$. When applying integration by parts on the second integral on LHS, we get

$$-\frac{\Delta t}{2\varrho}\int_\Omega \nabla(\alpha\nabla u) v dx = \frac{\Delta t}{2\varrho}\int_\Omega \alpha\nabla u\nabla v dx - \frac{\Delta t}{2\varrho}\int_{\partial\Omega} \alpha\frac{\partial u}{\partial n} v dx \tag{6}$$

where the rightmost term vanishes due to Neumann boundary conditions, $du/dn = 0$. Similarly we get rid of the gradient of $\alpha$ in the second RHS integral when applying integration by parts on it. Finally we got a scheme that the computer is satisfied with

$$\begin{aligned} &\int uvdx + \frac{\Delta t}{2\varrho}\int \alpha\nabla u\nabla vdx - \frac{\Delta t}{2\varrho}\int f(u)vdx \\ = &\int u^n vdx - \frac{\Delta t}{2\varrho}\int \alpha\nabla u^n\nabla vdx + \frac{\Delta t}{2\varrho}\int f(u^n)vdx \end{aligned} \tag{7}$$

with a LHS dependent on $u$ and $v$, and a RHS dependent of $v$ only. We define the LHS integrand as $a(u, v)$ and the RHS integrand as $L(v)$:

$$a(u,v) = uvdx + \frac{\Delta t}{2\varrho}\alpha\nabla u\nabla vdx - \frac{\Delta t}{2\varrho}f(u)vdx \tag{8}$$

$$L(v) = u^n vdx - \frac{\Delta t}{2\varrho}\alpha\nabla u^n\nabla vdx + \frac{\Delta t}{2\varrho}f(u^n)vdx \tag{9}$$

which is known as the variational formulation. To solve this equation, we could use Picard iterations.

## 2.2 Picard Iterations

Consider a polynomial of 3. order that is set equal to zero: $P(x) = ax^2 + bx + c = 0$. Assume we want to solve the system with respect to $x$, but we only have a bad approximation, namely $x^-$. Picard tells us that we can modify our problem to

$$\tilde{P}(x) = ax^-x + bx + c = 0 \tag{10}$$

with

$$x = -\frac{c}{ax^- + b} \tag{11}$$

as a better solution than the initial approximation $x^-$. We could now set $x^- = x$ and repeat several times. For each *iteration* the approximation will be better, and we can stop when the error is satisfying small.

For our problem the Picard iterations are slightly more complicated since we are dealing with time as well. The clue is to use the solution calculated in the previous time step as a approximation of the current time step solution. Of course this requires that an initial solution is given, but this is reasonable because we usually know how the system behaves at time zero.

## 2.3 Linearity and solutions

We started this report stating that the diffusion equation is non-linear, with a non-constant $\alpha$. On the other hand if $\alpha$ is constant, we simply got the heat equation, which in linear. With this in mind, we can construct some convenient test functions to verify our implementations.

### 2.3.1 Constant solution

Firstly we will study a constant solution $u(x,t) = 1.0$. A constant solution will require $f(x,t) = 0$, while $\varrho$ and $\alpha$ are arbitrary (finite) constants.

### 2.3.2 Linear solution

With a linear solution we mean that $\alpha$ is constant. Secondly we will study the linear solution $u(x,t) = \exp(-\pi^2 t)\cos(\pi x)$ and the diffusion equation in 1D with a constant $\alpha$;

$$\varrho\frac{\partial u}{\partial t} = \alpha\frac{\partial^2 u}{\partial x^2} + f(x,t). \tag{12}$$

One can show that $\partial u/\partial t = \partial^2 u/\partial x^2$, i.e $\varrho = \alpha$ and $f(x,t) = 0$. At time $t = 0$ we have $u(x,0) = I(x) = cos(\pi x)$, so this problem is quite easy to implement and makes a good test. Since this solution is linear, we expect the error to be linear as well. Later we will see that the Crank-Nicolson scheme gives a error $E = Kh$ where $h = \Delta x^2 = \Delta y^2 = \Delta z^2$, so a revealing test would be to calculate $E/h$ and see if it remains constant when reducing $h$.

### 2.3.3 Non-linear solution

Finally we will examine a non-linear solution, define by the integral

$$u(x,t) = t\int_0^x q(1-q)dq = tx^2\left(\frac{1}{2} - \frac{x}{3}\right). \tag{13}$$

$\alpha(u)$ is no longer constant, but given by $\alpha(u) = 1 + u^2$. If we insert this into the diffusion equation, we will find that the equation is satisfied if

$$f(x,t) = -\frac{\varrho x^3}{3} + \frac{\varrho x^2}{2} + \frac{8t^3x^7}{9} - \frac{28t^3x^6}{9} + \frac{7t^3x^5}{2} + \frac{5t^3x^4}{4} + 2tx - t \tag{14}$$

where $\varrho$ is arbitrary, but usually set to 1.

## 2.4 FEniCS implementation

Also in the implementation we restrict the Picard iteration to a single iteration, i.e we define both the $\alpha$'s as a function of the previous solution $u_n$. After finding the formulas to be solved in each Picard iteration, the implementation is quite straightforward since FEniCS is solving $a(u, v) = L(v)$ without any further processing. We need to implement the Picard iterations, which is simply done by a time loop. Given the system size $Nx$, $Ny$ and $Nz$, time length and time step $T$ and $Nt$, the element type *elements*, a boundary function $I$, a constant *rho* and the function *alpha*, we could do the implementation as

```
dt = float(T)/Nt
C = dt/(2*rho)

mesh = [UnitIntervalMesh(Nx), UnitSquareMesh(Nx, Ny),
    ↪ UnitCubeMesh(Nx, Ny, Nz)]
V = FunctionSpace(mesh[dimension - 1], 'P', elements)
u_n = interpolate(I, V)

u = TrialFunction(V)
v = TestFunction(V)
a = u*v*dx + C*alpha*dot(grad(u), grad(v))*dx
L = u_n*v*dx - C*alpha*dot(grad(u_n), grad(v))*dx
    ↪ + 2*C*f*v*dx

u = Function(V)
t_list = np.linspace(dt, T, Nt)

for t in t_list:
    f.t = t
    alpha.u = u_n

    solve(a == L, u)
    u_n.assign(u)
```

Notice that the constant $C = dt/(2 * \varrho)$ is taken out to decrease the number of floating point operations. You may also observe that the FEniCS expressions need to be given in c-code, where the cmath package is supported. The following is standard FEniCS implementation, but in the variational formulation implementation I have modified the equation using $(f_n + f) = 2f$ since we only are dealing with a constant function $f$. For exact implementation, see github repository.

### 2.5 Errors

Usually when solving mathematics numerically, we get errors from different sources. Often we need to approximate expressions to transform the equation to a better form, and we always have truncation errors.

In this project we get some errors when we use Crank-Nicolson to discretize the working equation, which goes as

$$E = K_t \Delta t^2 + K_x \Delta x^2 + K_y \Delta y^2 = Kh \tag{15}$$

where $K \equiv K_t + K_x + K_y$ and $h = \Delta x^2 = \Delta y^2 = \Delta z^2$. This is only true for the Crank-Nicolson scheme, for other schemes the time error could be different.

Furthermore we also have numerical errors in Fenics, which become visible at low time steps, but is affecting the final result all the time. This error is mainly truncation error, which occurs because a computer is not able to handle infinity number of digits. Most computers today got 64-bits systems, and can handle maximum 16 digits.

## 3 Results

We have been solving the diffusion equation for the functions introduced in section 2.3 with various number of spatial and time steps and P elements, and the numerical solution is compared with the exact solution is all cases. The absolute error is presented below.

### 3.1 Constant solution

This solution is pretty simple, so it is easy to compute the error. The absolute maximal errors for P1 and P2 element in 1, 2 and 3 dimensions are found in table (1).

Table 1: This table represents the error when solving the system for a constant solution.

| Elements | 1D | 2D | 3D |
|---|---|---|---|
| P1 | 2.77555756e-15 | 3.55271367e-15 | 2.60902410e-14 |
| P2 | 1.26343380e-13 | 1.39666056e-13 | 8.69304628e-14 |

We observe that the error is larger for P2 element compared with P1 element, which is interesting. The plots are very similar, so in my opinion it is sufficient to present plots only for P2 element. Plots showing the numerical solution in 2 and 3 dimensions are found in figure (1) and (2).

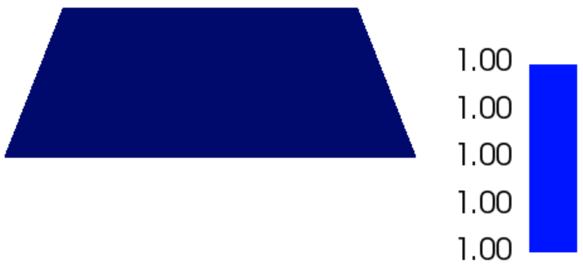


Figure 1: This figure shows the numerical solution of the constant function in 2 dimensions

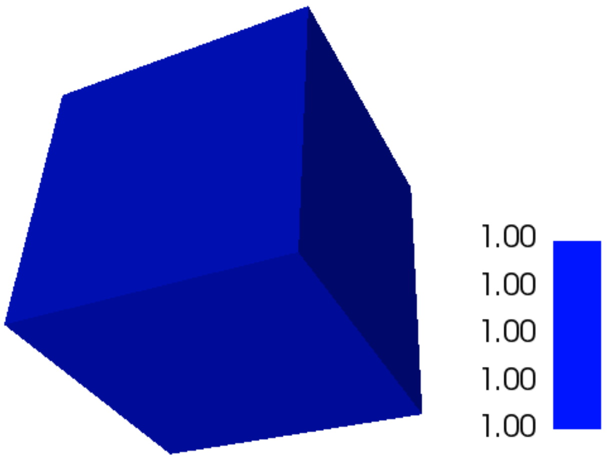


Figure 2: This figure shows the numerical solution of the constant function in 3 dimensions

We can see that both the numerical solution of the constant function in 2D (figure (1)) and 3D (figure (2)) are constant 1.0 in all directions.

## 3.2 Linear solution

For the linear solution we are studying how the error decreases when we increase the number of spatial and time steps, and moreover we see how the ratio $E/h$ develops. The absolute error is presented in table (2), and the ratio can be found in table (3).

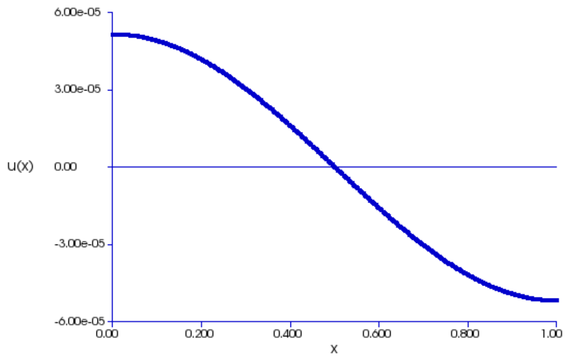
Table 2: This table presents the error when finding the linear solution numerically for 1, 2 and 3 dimensions and $h \in [10e-3, 10e-5]$ where $E$ is the error and $h$ is the step parameter $h = \Delta x^2 = \Delta y^2 = \Delta t^2$. Used $T = 0.05$ and $Nz = 10$

| $h$ | 1D | 2D | 3D |
|---|---|---|---|
| 0.00100 | 3.11016728e-06 | 2.37216402e-04 | 6.16466008e-05 |
| 0.00050 | 1.58817190e-06 | 1.174367628-04 | 2.55254641e-05 |
| 0.00010 | 3.23369943e-07 | 2.30428176e-05 | 3.31934332e-06 |
| 0.00005 | 1.62653483e-07 | 1.15036041e-05 | – |
| 0.00001 | 3.23584499e-08 | 2.26695404e-06 | – |

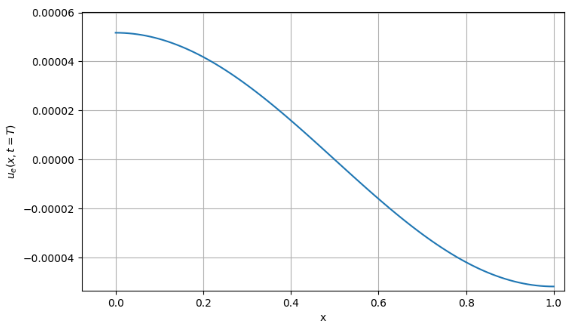
Table 3: This table presents the ratio $E/h$ when finding the linear solution numerically. $E$ is the error and $h$ is the step parameter $h = \Delta x^2 = \Delta y^2 = \Delta t^2$, and we measure with $T = 0.05$ and $Nz = 10$

| $h$ | 1D | 2D | 3D |
|---|---|---|---|
| 0.00100 | 0.00311016728 | 0.237216402 | 0.0616466008 |
| 0.00050 | 0.00317634380 | 0.234873525 | 0.0510509283 |
| 0.00010 | 0.00323369943 | 0.230428176 | 0.0331934332 |
| 0.00005 | 0.00325306966 | 0.230072082 | – |
| 0.00001 | 0.00323584499 | 0.226695404 | – |

For the 1 and 2 dimensional cases we can see that the ratio is approximately constant, but for 3 dimensions we do not have a clear constant. I ran out of memory when calculating the error for $h = 0.00005$. The 1 dimensional numerical solution is plotted side-by-side with the exact solution in figure (3a) and (3b).



(a) Numerical solution



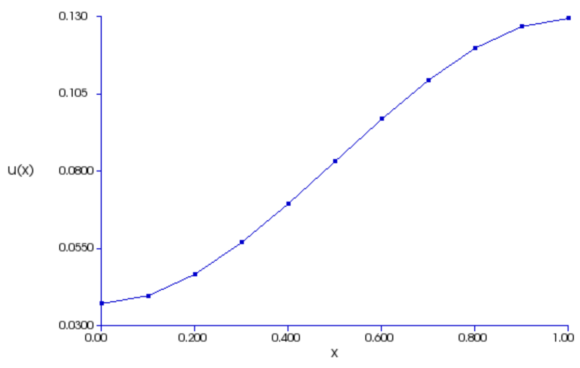
(b) Exact solution

Figure 3: On the left-hand side the 1 dimensional numerical linear solution is plotted, and on the right-hand side one can find the exact solution, which is $u_e(x,t) = \exp(-\pi^2 t)\cos(\pi x)$. The plots represent the final function at time $T$, where $T = 1.0$.

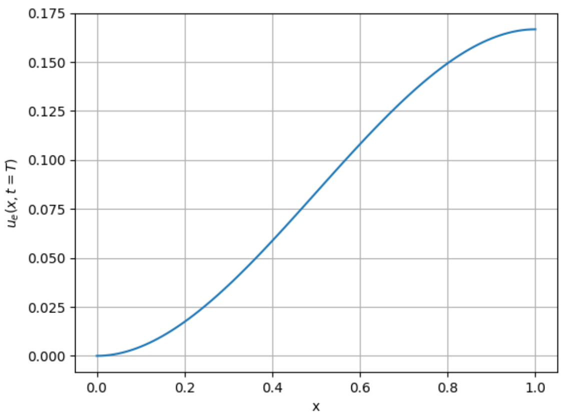
We can see that the exact and numerical solution is quite similar, without any obvious deviations.

## 3.3 Non-linear solution

For the non-linear solution we only study P1 element in one dimension, and get the absolute error $E = 0.167165830348$, which is much larger than the error for the constant and the linear function. The 1 dimensional numerical solution is plotted side-by-side with the exact solution in figure (4a) and (4b).



(a) Numerical solution



(b) Exact solution

Figure 4: On the left-hand side the 1 dimensional numerical non-linear solution is plotted, and on the right-hand side one can find the exact solution, which is $u_e(x,t) = tx^2(0.5 - x/3)$. The plots represent the final function at time $T$, where $T = 1.0$.

Also for this solution we can see that the behavior of the numerical solution is similar with the behavior of the exact solution, but we can see that while the exact solution is zero at $x = 0$, $u_e(0,T) = 0$, the numerical solution differs. Evidently the end points are the same for both, $u_e(1,T) = 1/6$.

## 4 Discussion

For the constant solution we saw that P1 element gave a better numerical solution than P2 element, which seems strange. This is perhaps because we have a constant solution, and one element will therefore fit almost perfectly to all the domain. I will therefore expect a P2 element solution to be more accurate for a non-constant solution, but lets leave that study for a later project.

We were also studying how the error developed for the linear solution as we reduced $h$, and we saw that the error decreased, as expected. I was not able to run the program for $h = 0.00005$ and $h = 0.00001$ in 3 dimensions due to memory error. As mentioned before, we expect the ratio $E/h$ to be approximately constant, which is true for 1 and 2 dimensions. For 3 dimensions the factor $E/h$ is not constant, and I suspect it is because we need even smaller time step (or shorter time $T$) to get the expected result. Anyway this was not a part of the exercise, and I will therefore not focus too much on it. As showed in figure(3) the numerical solution is evidently similar to the exact solution, at least in 1 dimension.

This is not true for the non-linear solution, we have obviously a significant error for lower $x$ values. I have no good explanation, but it is probably caused by the non-linearity. On the other hand the solution is not that bad, the form is similar and the solution at higher $x$ values seems reasonable.