Nonlinear diffusion equation

Marte Fossum

November 2019

Introduction

We are given the nonlinear diffusion equation

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

with initial condition $u(\boldsymbol{x}, 0) = I(\boldsymbol{x})$, boundary condition $\frac{\partial u}{\partial n} = 0$ and where ϱ is a constant and $\alpha(u)$ is a known function. We want to look at various numerical aspects of this model.

a)

Using Backward Euler for discretizing time, we get

$$u_t(\boldsymbol{x}, t_n) = \frac{u^n - u^{n-1}}{\Delta t}$$

where $u^n = u(\boldsymbol{x}, t_n)$. Inserted into (1) gives us

$$\begin{aligned} \varrho\frac{u^n - u^{n-1}}{\Delta t} &= \nabla \cdot (\alpha(u^n)\nabla u^n) + f(\boldsymbol{x}, t_n) \\ u^n - u^{n-1} &= \frac{\Delta t}{\varrho}(\nabla \cdot (\alpha(u^n)\nabla u^n) + f(\boldsymbol{x}, t_n)) \end{aligned}$$

Knowing that $u^0 = I$, we can find u^i for $i = 1, 2, ...$

Assuming that we know u^{n-1} from the previous timestep, we move everything known to one side and everything unknown to the other side.

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

where $f^n = f(\boldsymbol{x}, t_n)$. In order to derive a variational formulation, we multipy with a test function $v \in \hat{V}$ and integrate.

$$\int_\Omega \left(u^n v - \frac{\Delta t}{\varrho}\nabla \cdot (\alpha(u^n)\nabla u^n)v \right) dx = \int_\Omega \left(u^{n-1}v + \frac{\Delta t}{\varrho}f^n v \right) dx \tag{2}$$

And this again we can rewrite to

$$a(u, v) = L_n(v)$$

where $a(u, v)$ and $L_n(v)$ are left and right side of (2) respectively.

b)

To find the Picard method, we substitute $\alpha(u^n)$ in (2) with $\alpha(u^{n-1})$, assuming that u^{n-1} is the most recently computed value. Then this value will be updated for every iteration. Giving us

$$\int_\Omega \left(u^n v - \frac{\Delta t}{\varrho}\nabla \cdot (\alpha(u^{n-1})\nabla u^n)v \right) dx = \int_\Omega \left(u^{n-1}v + \frac{\Delta t}{\varrho}f^n v \right) dx \tag{3}$$

$$\tag{4}$$

c)

Implemented a) and b) and have written a code that should solve (1).

d)

For the values $\varrho = 1, \alpha(u) = 1, f = 0$ and $I = 1$, we get the constant solution $u(\boldsymbol{x}, t) = 1$. The error between the exact solution and the numerical solution is then in the $\sim 10^{-14}$ range, which is almost machine precision.

```
1 For the interval mesh with N = 10, the error: 3.258166265798654e-14
2 For the square mesh with N = 10, the error: 3.6036395822482216e-14
3 For the box mesh with N = 10, the error: 8.493735394549023e-14
```

e)

We want to show that the ratio of E/h is about constant when you change the mesh in both space and time using $h = \Delta x^2 = \Delta y^2 = \Delta t$.

These are the given values we get for E/h:

```
1 h = 0.05000, E/h = 13.446361, mesh = [20, 20]
2 h = 0.02500, E/h = 27.582856, mesh = [40, 40]
3 h = 0.01250, E/h = 55.862414, mesh = [80, 80]
4 h = 0.00625, E/h = 112.426199, mesh = [160, 160]
5 h = 0.00313, E/h = 225.556464, mesh = [320, 320]
```

Looking at the result, we can see that the ratio doesn't seem very constant, meaning something in my code probably is wrong. It seems however that E is constant and very high, which adds to the idea of something wrong in my code.

I did try to change (3) by taking the second term of the left hand side and do integration by parts (that's the a that is written as a comment in the program).

$$-\int_\Omega \frac{\Delta t}{\varrho}\nabla\cdot(\alpha(u^{n-1})\nabla u^n)v dx = -\frac{\Delta t}{\varrho}\left[\alpha(u^{n-1})\frac{\partial u^n}{\partial n}v\right]_{\partial\Omega} + \frac{\Delta t}{\varrho}\int_\Omega \left(\alpha(u^{n-1})\nabla u\cdot\nabla v\right) dx$$

$$= \frac{\Delta t}{\varrho}\int_\Omega \left(\alpha(u^{n-1})\nabla u\cdot\nabla v\right) dx$$

as $\frac{\partial u}{\partial n} = 0$ on the boundary.

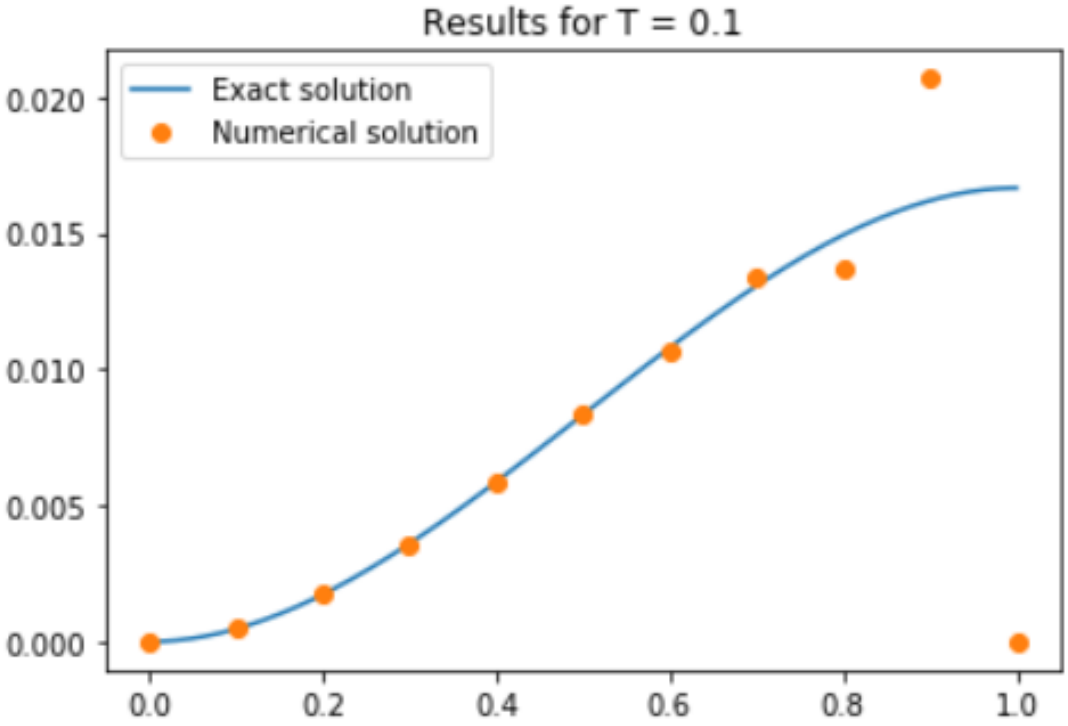
With this other a I get the following results:

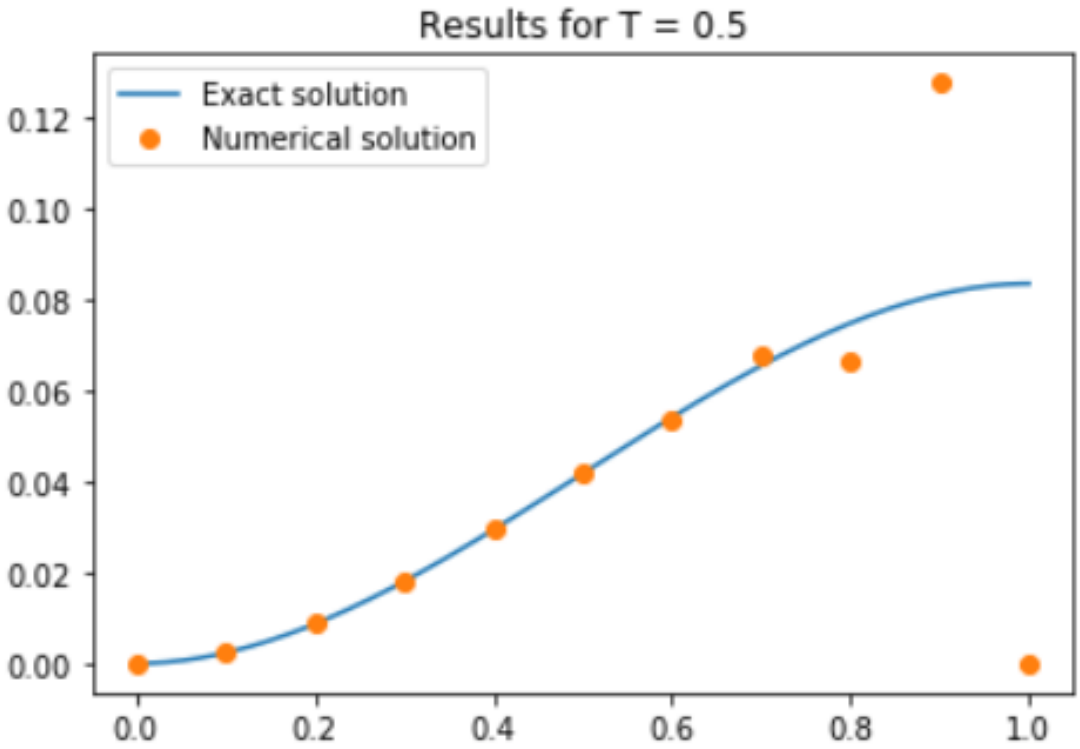
```
h = 0.05000, E/h = 0.003937, mesh = [20, 20]
h = 0.02500, E/h = 0.002726, mesh = [40, 40]
h = 0.01250, E/h = 0.002219, mesh = [80, 80]
h = 0.00625, E/h = 0.001992, mesh = [160, 160]
h = 0.00313, E/h = 0.001884, mesh = [320, 320]
```

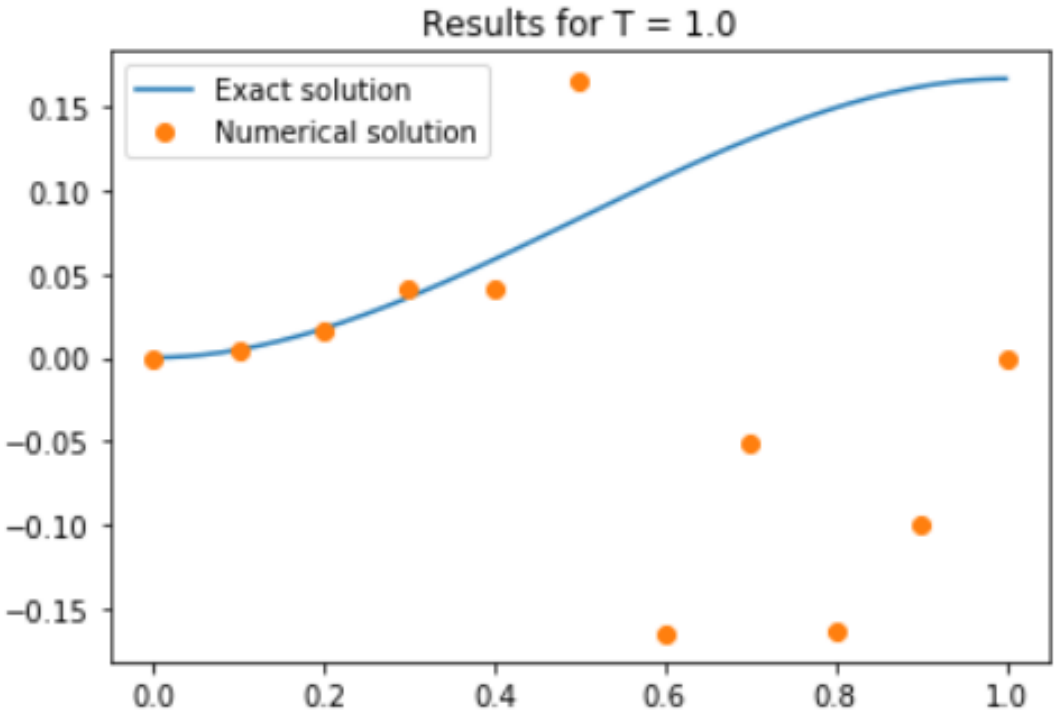
Which seems like a more probable result, though it still is not very constant.

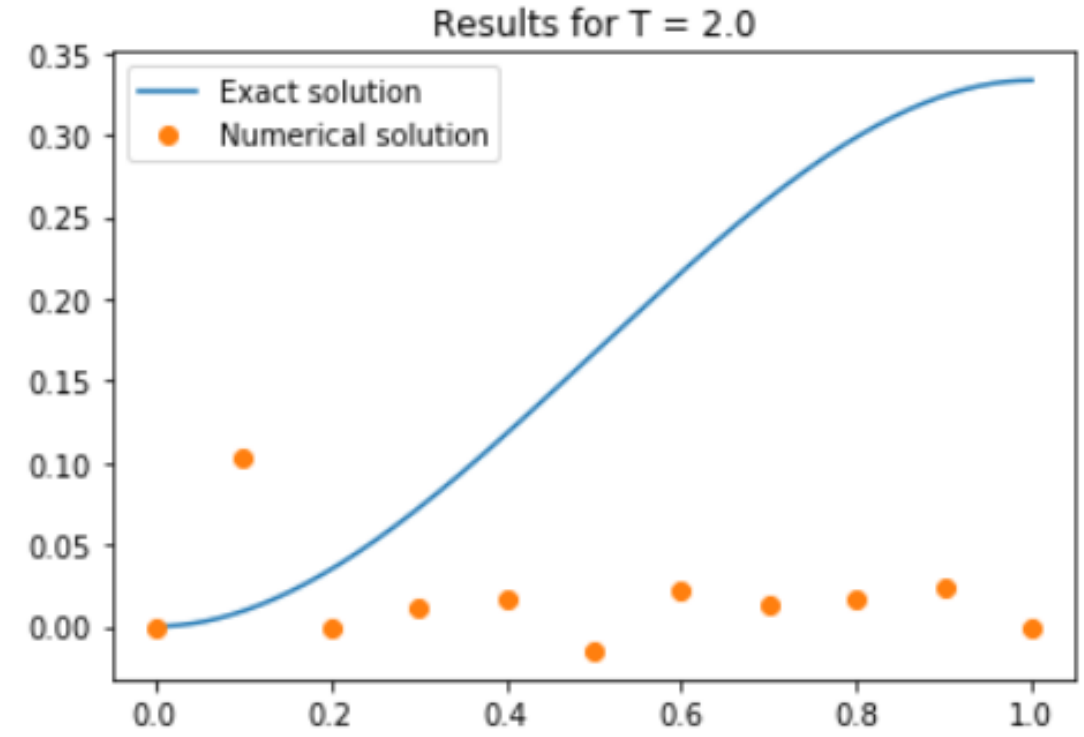
f)

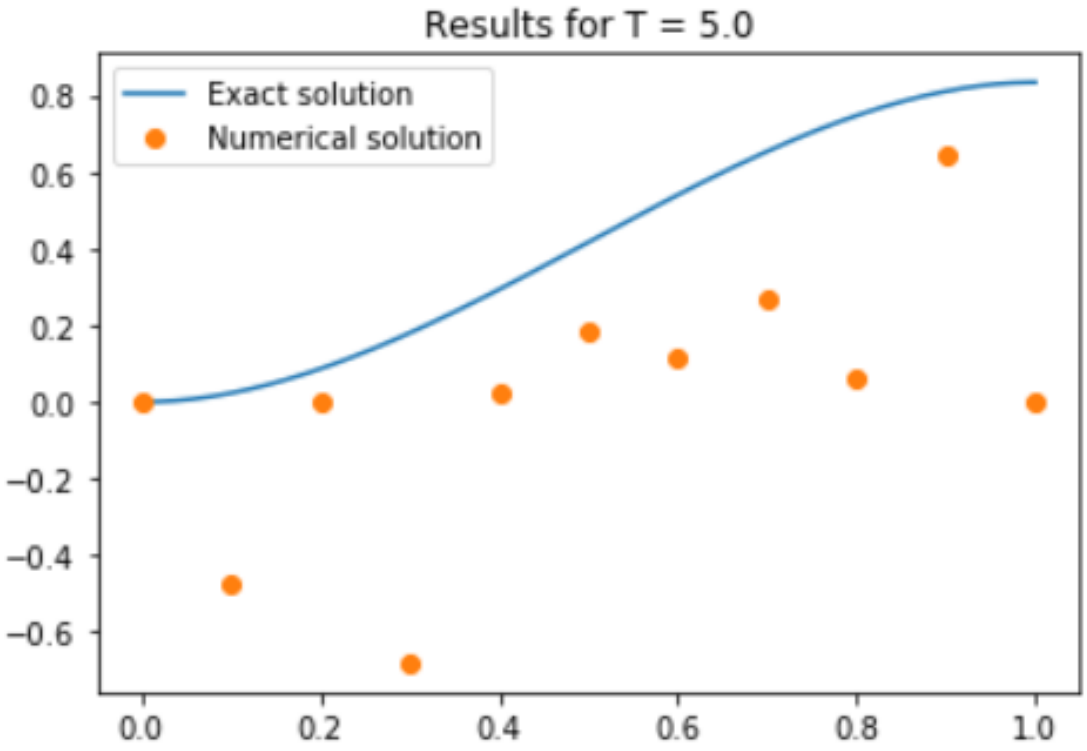
I get weird results here. I also forgot to label the axes.




Results for T = 0.5
Exact solution
Numerical solution
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0.0
0.2
0.4
0.6
0.8
1.0


Results for T = 1.0
Exact solution
Numerical solution
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
0.0
0.2
0.4
0.6
0.8
1.0



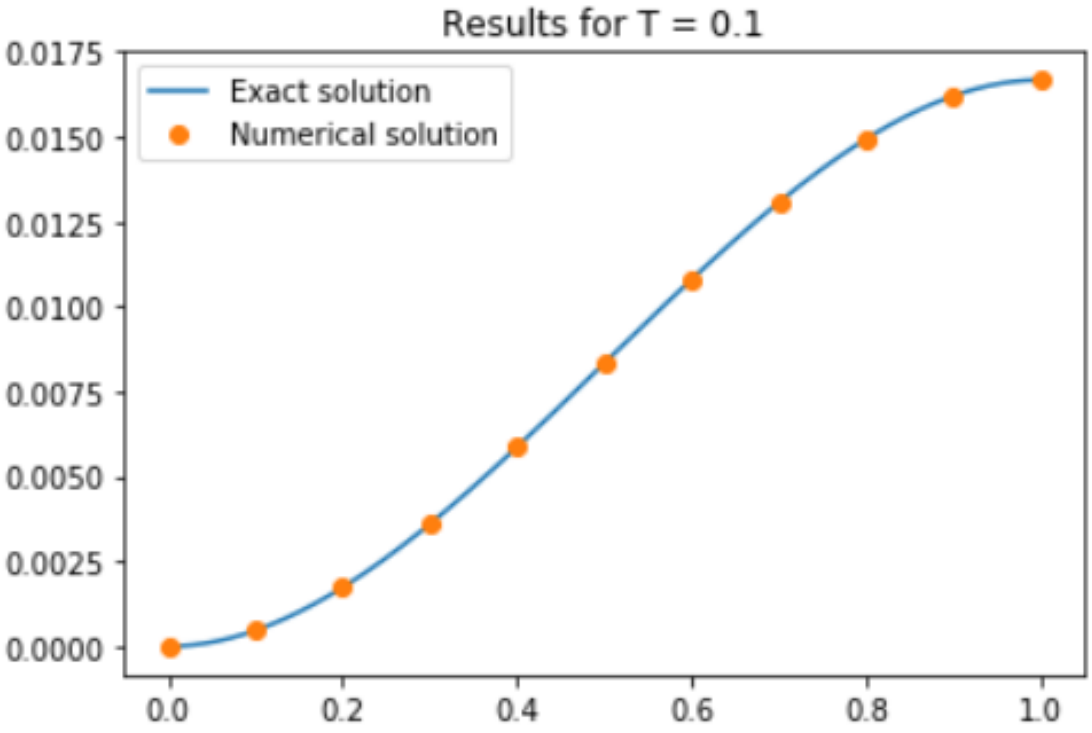


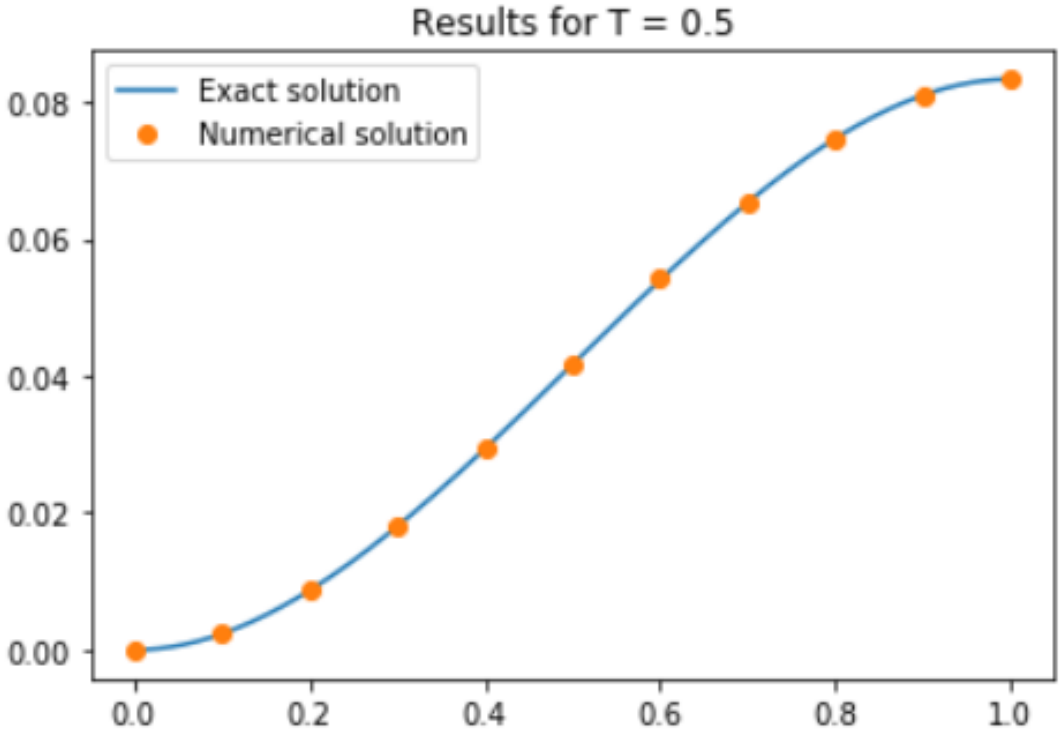
We see that something weird is happening at the $x = 1$ boundary, which is probably something in my code.

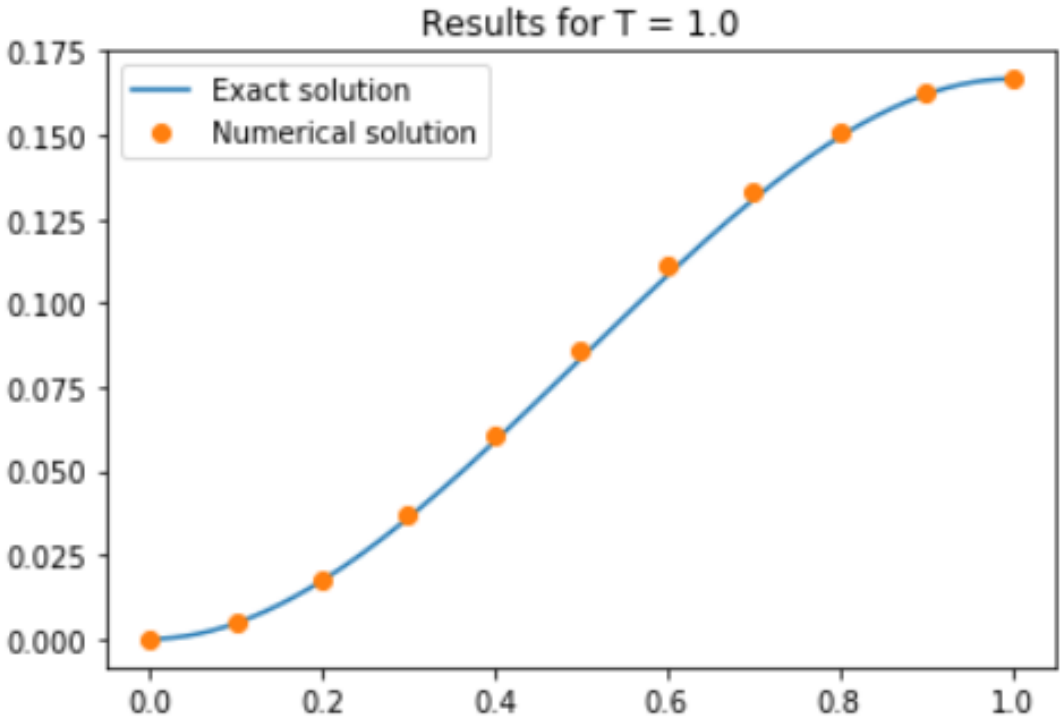
However, by removing the boundary like this.

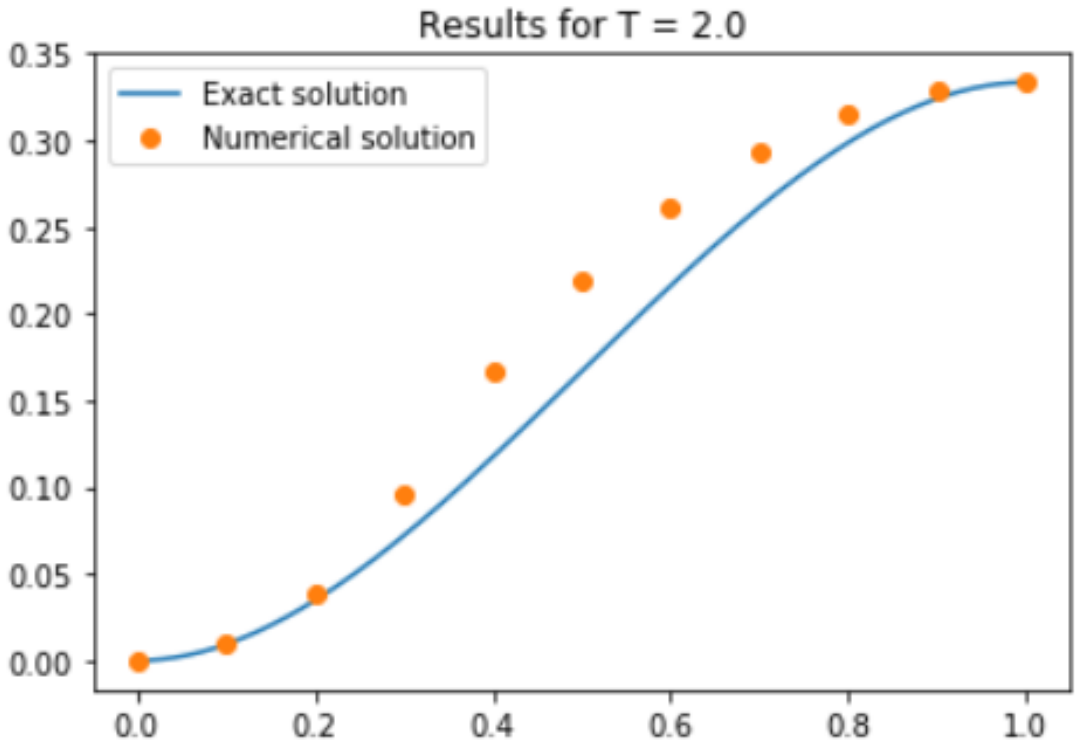
```
1 u, E = solver('interval', N, f, alpha, I, rho, dt, time, u_e)
```

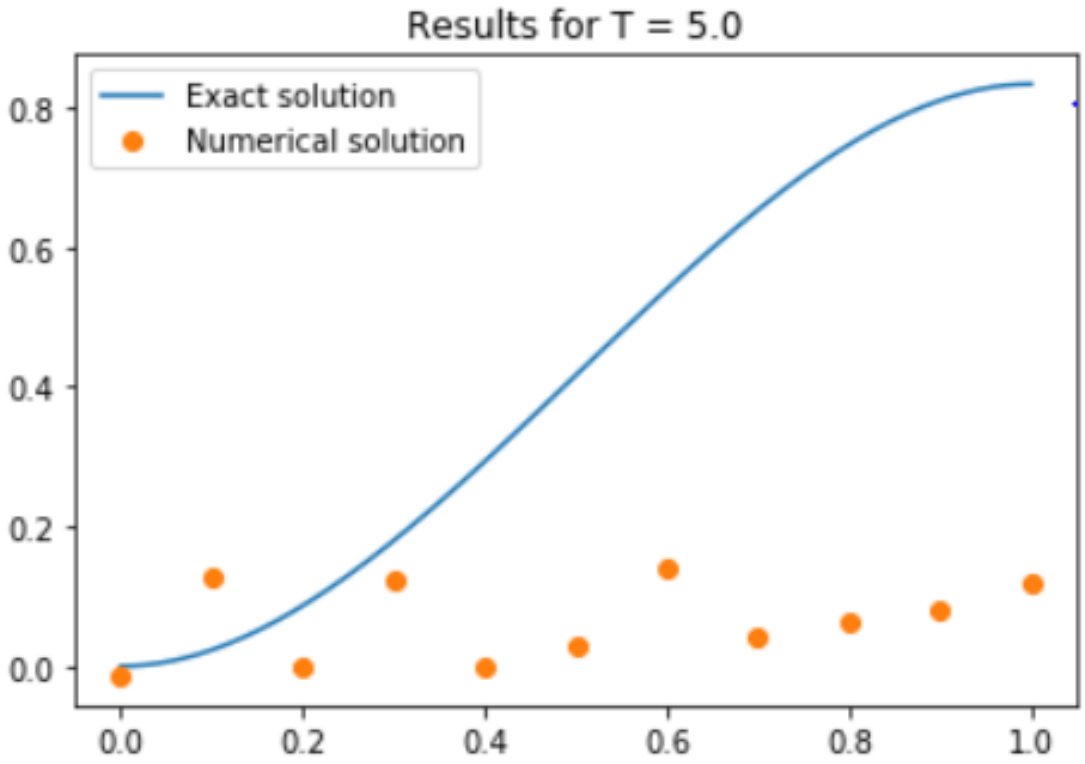
Then the result is a lot better.






Results for T = 1.0
Exact solution
Numerical solution
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0.0
0.2
0.4
0.6
0.8
1.0


Results for T = 2.0
Exact solution
Numerical solution
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0.0
0.2
0.4
0.6
0.8
1.0



I don't see why this should give a better result as the boundary condition is not implemented. What we see from both of the implementations however, is that the numerical solution is more accurate for small times.

f)

Things that might be wrong in the program:

- The scheme that is used (here we have used Beckward Euler)
- Too big of a time (as we saw from excersise e)
- The mesh (probably, it's always something that can go wrong)