Project report presented for
M4N10

Computational PDEs: Project 1

Yadu *Bhageria*

Imperial College London
Mathematics Department
United Kingdom
February 6, 2017

1. In this question it is assumed that

$$S(r,t) = 0$$

The equation that is being solved is then

$$u_t = V(r)u_r + D(u_{rr} + \frac{u_r}{r}) \quad \textbf{in } a < r < b \text{ and } t > 0 \tag{1}$$

subject to the given boundary conditions. This equations is very similar to the one solved in the code provided in `advdiff.m`. It can even be transformed to have the same form by rewriting it in the form

$$\begin{aligned} u_t &= (V(r) + \frac{D}{r})u_r + Du_{rr} \\ &= \frac{Q+D}{r}u_r + Du_{rr} \end{aligned} \tag{2}$$

The 1D grid, of size $N+1$, for this annulus is split in even step sizes of Δr for r. So the grid looks like $\{a, a+\Delta r, a+2\Delta r, \ldots, b-\Delta r, b\}$. Now centered-difference schemes can be used to define u_r and u_{rr} as follows:

$$\frac{\partial u_n}{\partial r} = \frac{u_{n+1} - u_{n-1}}{2\Delta r} \tag{3}$$

and

$$\frac{\partial^2 u_n}{\partial r^2} = \frac{u_{n+1} - 2u_n + u_{n-1}}{\Delta r^2} \tag{4}$$

This gives that for the j^{th} iteration in timestep, Δt, the PDE can be solved using an iterative formula

So the complete algorithm after setting up the problem, for Nt time steps, is

Algorithm 1: Solving the PDE

```
for $j = 1 : Nt$ do
    $u_0^{(j+1)} = 1$ for $n$ = 2:N do
        $u_n^{(j+1)} = u_n^{(j)} + \Delta t[\frac{Q}{r_n}\frac{u_{n+1}^{(j)} - u_{n-1}^{(j)}}{2\Delta r} + D(\frac{u_{n+1}^{(j)} - 2u_n^{(j)} + u_{n-1}^{(j)}}{\Delta r^2} + \frac{u_{n+1}^{(j)} - u_{n-1}^{(j)}}{2\Delta r}\frac{1}{r_n})]$
    end
    $u_{N+1}^{(j+1)} = 0$
end
```

The code for this can be found in the `q1.m` file. I have used a simple test case with a step function where

$$u_0(r) = \begin{cases} 0 & \text{if } r \le \frac{b+a}{2} \\ 1 & \text{if } r > \frac{b+a}{2} \end{cases}$$

This gives me the following resulting figure for the plot of r against $u(r,t)$.

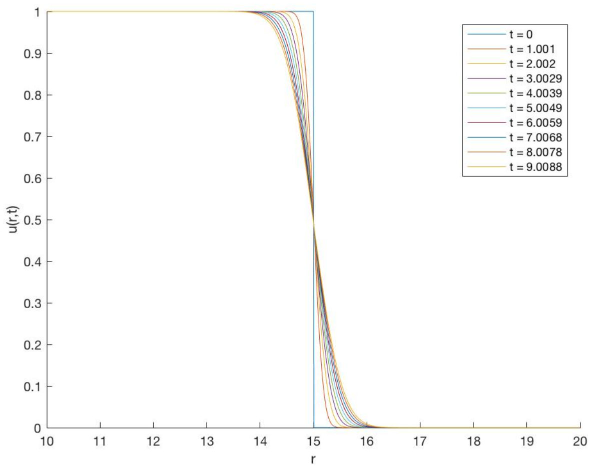

Figure 1: r vs $u(r,t)$
for t $= 0$ to 10 with $a = 10, b = 20, Q = 0.01$, $D = 0.01$, $N = 2^{10}$, $Nt = 2^{12}$

And the contour plot for this initial condition with the specified boundary condition gives

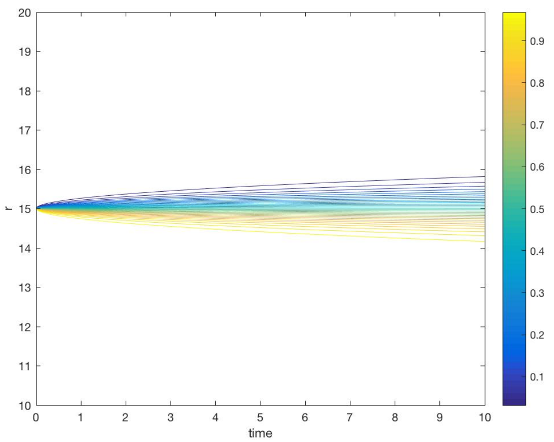

Figure 2: Contour plot of t vs r
with $a = 10, b = 20, Q = 0.01$, $D = 0.01$, $N = 2^{10}$, $Nt = 2^{12}$

2. For this section I have chosen the function

$$u(r,t) = \frac{b-r}{b-a} + (r-a)(r-b)e^{-t} \tag{5}$$

that satisfies the boundary conditions specified. Namely $u(a,t) = 1$ and $u(b,t) = 0$.

The given equation $(\star)$ can be rearranged to for $S(r,t)$ to give

$$S(r,t) = u_t - (\frac{Q}{r}u_r - D(u_r r + \frac{u_r}{r}) \tag{6}$$

For my chosen function the unknown terms are

$$\begin{aligned} u_t &= -(r-b)(r-a)e^{-t} \\ u_r &= -\frac{1}{b-a} + (2r-a-b)e^{-t} \\ u_r r &= 2e^{-t} \end{aligned} \tag{7}$$

which can be combined to explicitly give $S(r,t)$. $u_0(r)$ is

$$u_0(r) = \frac{b-r}{b-a} + (r-a)(r-b) \tag{8}$$

Furthermore my chosen constant values are

$$\begin{aligned} a &= 1 \\ b &= 2 \\ D &= 0.01 \\ Q &= 0.01 \end{aligned} \tag{9}$$

This function can be solved using the code in `solveq2.m`.

Now to check whether the error behaves according to $\mathcal{O}(k, h^2)$ I consider the point $(r_0, t_0) = (1.5, 4)$.

Using the code in `q2error_k.m` I find that the error does indeed grow linearly with k.

Using the code in `q2error_h.m` I find that the error grows quadratically with h.

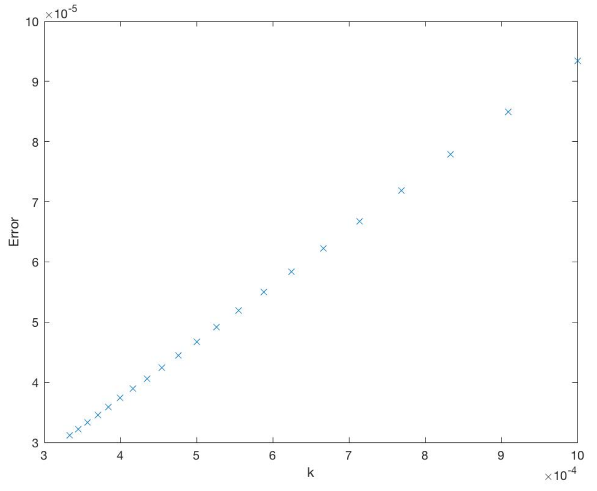

Figure 3: k vs Error

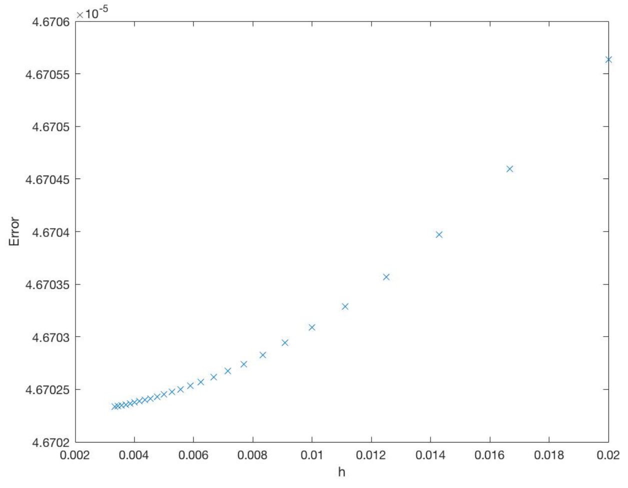

Figure 4: h vs Error

3. Since the equation above is similar to the advection equation dealt with in lecture 3-4, the stability analysis can be carried out in the same way as before.

   So

$$\mathbf{\Phi}(r,t,u,u_r,u_r r) = \frac{Q+D}{r}u_r + Du_{rr} + S(r,t) \tag{10}$$

which gives

$$\frac{\partial \boldsymbol{\Phi}}{\partial u_{rr}} = D \quad , \quad \frac{\partial \boldsymbol{\Phi}}{\partial u_r} = \frac{Q+D}{r} \quad \text{and} \quad \frac{\partial \boldsymbol{\Phi}}{\partial u} = 0 \tag{11}$$

and so the stability conditions are

$$\frac{k}{h^2} \leq \frac{1}{2D} \quad \text{and} \quad \frac{h}{2} \leq D|\frac{a}{D+Q}| \tag{12}$$

The same function as in the previous part has been used to analyse the stability conditions of the code.

Note that $\frac{1}{2D} = 50$ and that $2D|\frac{a}{D+Q}| = 1$. So by changing h and k the upper bounds can be achieved.

Setting $h = 0.01$ gives $k \leq \frac{h^2}{2D} = 0.005$. So for $k = \frac{4}{800} = 0.005$ the code works but for $k = \frac{4}{780} > 0.005$ the solution starts to become unstable. See the code in `q3.m` for a implemented example.

The second condition is overly cautious and going over it doesn't make the solution unstable.

4. $u_\infty(r)$ is found by iterating in time until the solution converges to some pre-specified tolerance. This is done in the code in `solveq4.m` where the tolerance is set to 10^{-10} although this can be altered very easily. Convergence of the solution is measured by comparing it to the norm of the difference between the solution at each iteration. That is until

$$||u^{(j+1)} - u^{(j)}|| < \text{Tolerance} \tag{13}$$

The values chosen for this part of the question are

$$\begin{aligned} a &= 1 \\ b &= 17 \\ k &= 0.01 \\ h &= 16/512 = 0.03125 \\ D &= 0.01 \end{aligned} \tag{14}$$

This gives a ratio of $\frac{b}{a} = 17$. The following plot shows some values of $\frac{Q}{D}$.

It can be seen that as Q/D increases the curve gets steeper which is reasonable as the effect of advection is much stronger than that of diffusion meaning the concentration of the drug is directed towards the centre of the annulus.

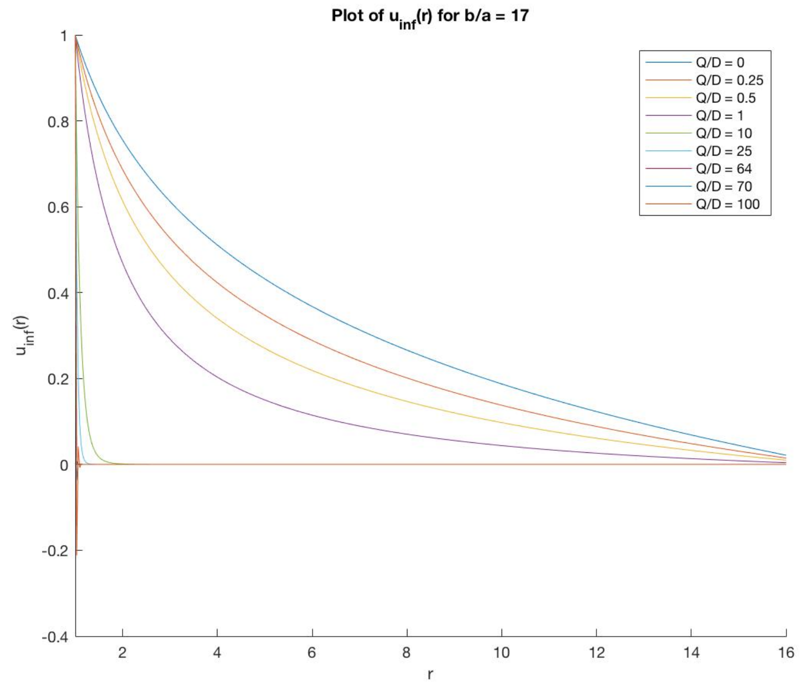

Figure 5: Various values of Q/D for $u_\infty(r)$ with a tolerance of 10^{-10}. From `q4.m`

As Q/D gets much larger as the slope becomes steeper this method becomes numerically unstable. This is to be expected as one of the stability conditions is $h < 2D|\frac{2a}{Q+D}|$. As Q/D increases this can be estimated as

$$2D|\frac{2a}{Q+D}| \approx \frac{2Da}{Q} \to 0$$

and so will eventually be smaller than the chosen value for h.

In this case $h = 0.01$ and so the scheme stops working when $\frac{D}{Q} < \frac{h}{2}$, i.e. when $Q/D > 64$. The figure above agrees with this result as the code works for $Q/D = 64$ but becomes unstable for $Q/D = 70$ and $Q/D = 100$.

Note that the first stability condition is always satisfied here as

$$10 \approx \frac{k}{h^2} < \frac{1}{2D} = 50$$

Find all the code attached in the zip file along with the submission