Assignment 5: Results/Description

Problem 1

For this problem I have written two programs (`myFTCS` and `myCN`) to solve the following problem. These may be compiled all together with a `make all` as shown below.

```
aburrow@schooner1:~/NumericalMethods/hw5
[aburrow@schooner1 hw5]$ [aburrow@schooner1 hw5]$
[aburrow@schooner1 hw5]$ pwd
/home/aburrow/NumericalMethods/hw5
[aburrow@schooner1 hw5]$ ls
config  doc  Makefile  plot.py  README.md  run_full  src
[aburrow@schooner1 hw5]$ make all
mkdir -p bin
mkdir -p output
mkdir -p doc/figs
icpc -mkl -c src/myFTCS.cpp -o bin/myFTCS.o
icpc -mkl -c src/io.cpp -o bin/io.o
icpc -mkl -c src/ftcs.cpp -o bin/ftcs.o
icpc -mkl bin/myFTCS.o bin/io.o bin/ftcs.o -o bin/myFTCS
icpc -mkl -c src/myCN.cpp -o bin/myCN.o
icpc -mkl -c src/crankNicholson.cpp -o bin/crankNicholson.o
icpc -mkl -c src/tridiag.cpp -o bin/tridiag.o
icpc -mkl bin/myCN.o bin/io.o bin/crankNicholson.o bin/tridiag.o -o bin/myCN
[aburrow@schooner1 hw5]$
```

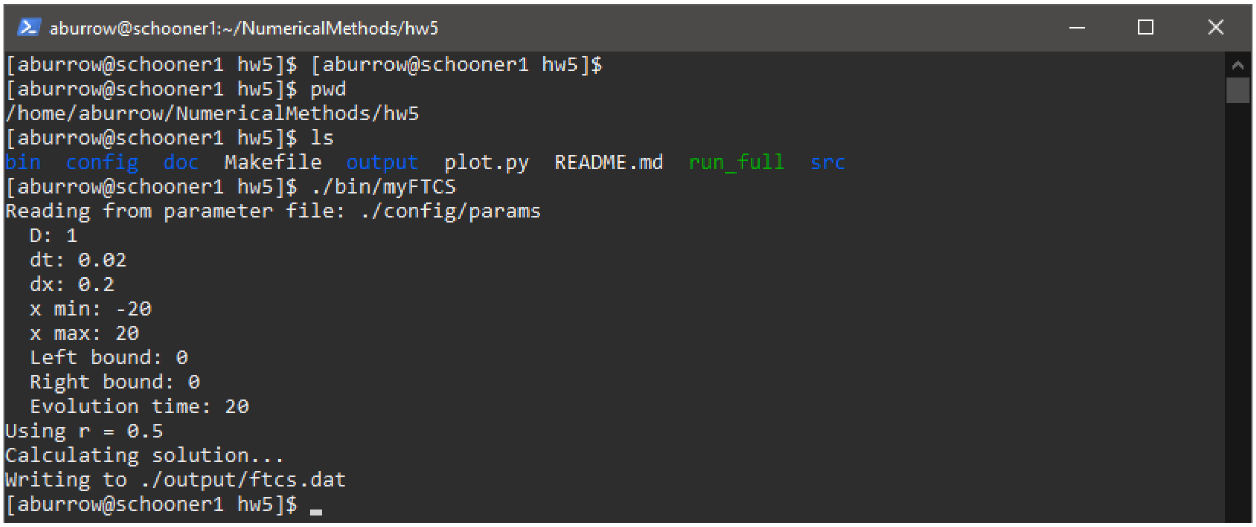
This generates executables `myFTCS` and `myCN` in the "./bin" directory, which may be run one at a time. There is also a `plot.py` in the root which may be run to generate plots from the output of the executables. However, to run all these programs at once, I have also included a `run_full` bash script that may be executed for convenience.

In this problem, I use (a) the FTCS method and (b) the Crank-Nicolson method to solve the diffusion equation

$$\frac{\partial n}{\partial t} = \frac{\partial^2 n}{\partial x^2}.$$

(a)

Below I run the program that demonstrates solving our differential equation using the FTCS method:



This was equation was solved over $t \in [0, 20]$ with the boundary conditions given in the problem. First, I solved this using $\Delta t = 0.2$ ($r = 0.5$), and the result is shown in Figure 1. We see clear diffusion in this solution, so the solver works as intended.

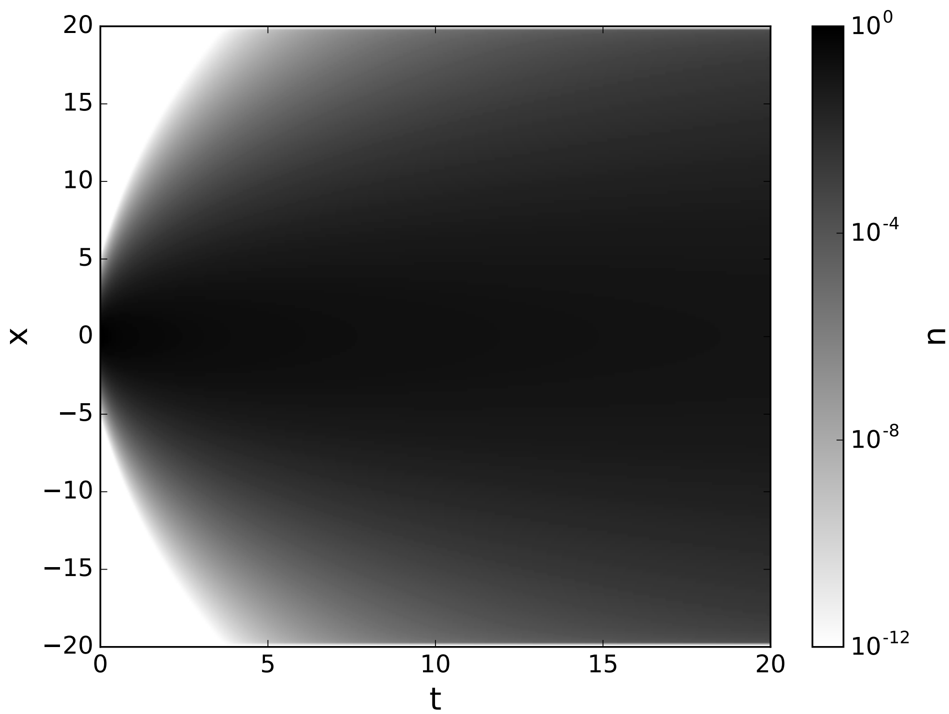


Figure 1: Solution using the FTCS method with $r = 0.5$.

Next, I change my time step to be $\Delta t = 0.266$ ($r = 0.665 > 0.5$), and the result is given in Figure 2. We see clearly that a divergence has occured as expected, due to the stability condition of the FTCS method. The solution apparently becomes completely unstable at about t = 2.5.

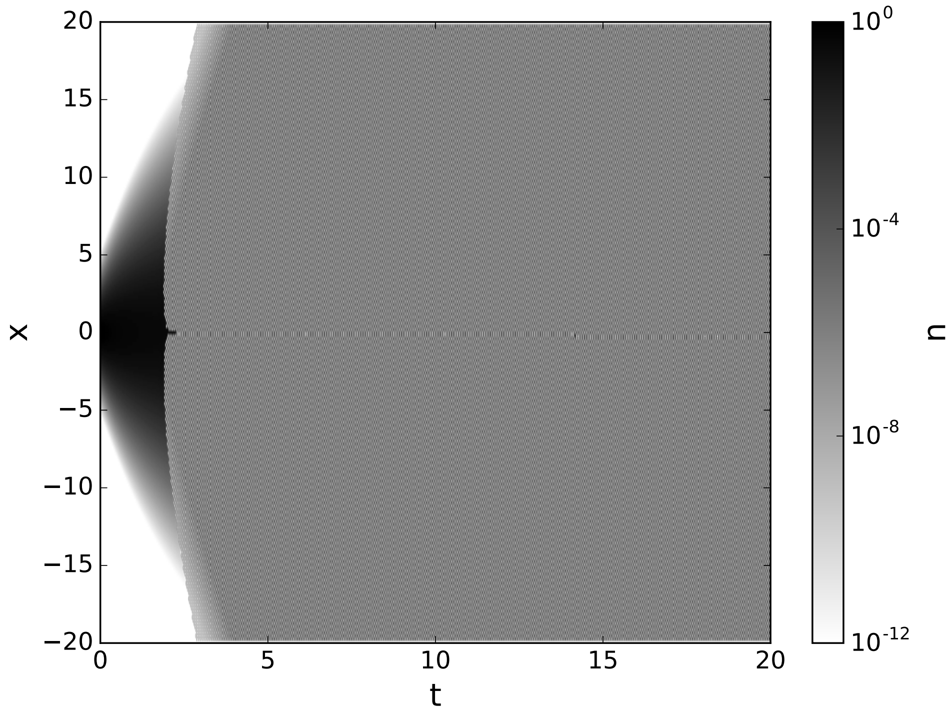


Figure 2: Solution using the FTCS method with $r > 0.5$.

(b)

Below I run the program that demonstrates solving our differential equation using the Crank-Nicolson method:

```
aburrow@schooner1:~/NumericalMethods/hw5
[aburrow@schooner1 hw5]$ [aburrow@schooner1 hw5]$ pwd
/home/aburrow/NumericalMethods/hw5
[aburrow@schooner1 hw5]$ ls
bin  config  doc  Makefile  output  plot.py  README.md  run_full  src
[aburrow@schooner1 hw5]$ ./bin/myCN
Reading from parameter file: ./config/params
  D: 1
  dt: 0.02
  dx: 0.2
  x min: -20
  x max: 20
  Left bound: 0
  Right bound: 0
  Evolution time: 20
Using r = 0.5
Calculating solution...
Writing to ./output/crankNicholson.dat
[aburrow@schooner1 hw5]$
```

For this method I use the same boundary conditions as before over the same time period. Again I calculate the solution with $\Delta t = 0.2$, and the result (shown in Figure 3) is the same as that in Figure 1. Note, however, that boundary conditions act slightly differently between the two methods (one with rigid values at the solution boundaries and one with "ghost" boundaries a distance Δt outside the solution boundaries). Because of this, the boundary values look slightly different between the two solutions.

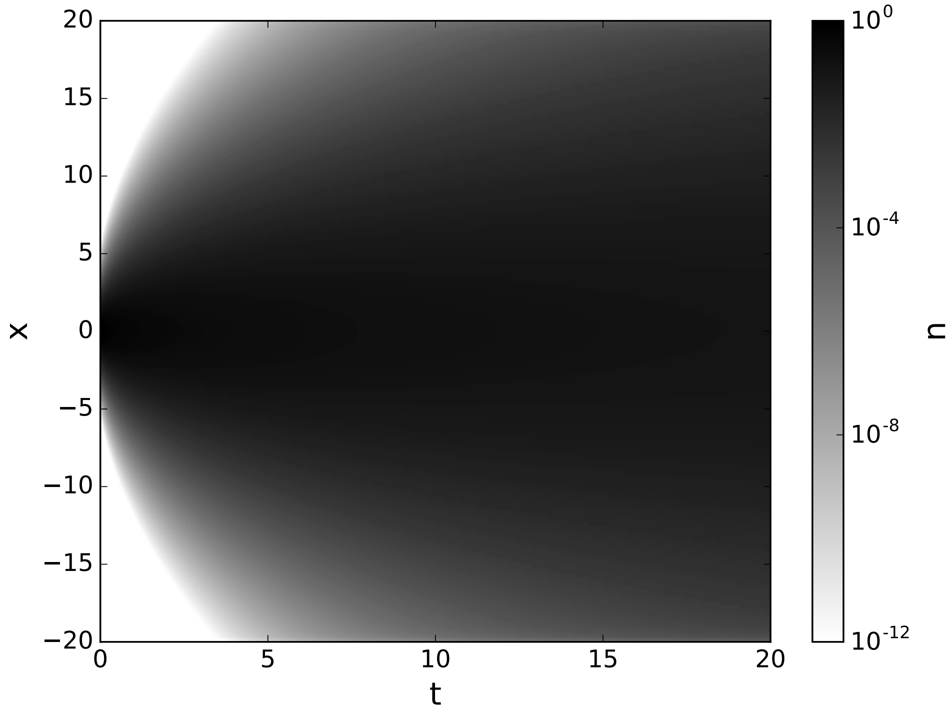


Figure 3: Solution using the Crank-Nicolson method with $r = 0.5$.

However, when I use $\Delta t = 0.266$ to calculate the solution (shown in Figure 4), we do not see any unstability, unlike for the FTCS case. Instead, we find the intended solution. I was not able to find any reasonable value of Δt that led to an unstable solution with the Crank-Nicolson method. In general, this should not happen, as it is shown to have no Courant condition for stability (however you still need it for accuracy).

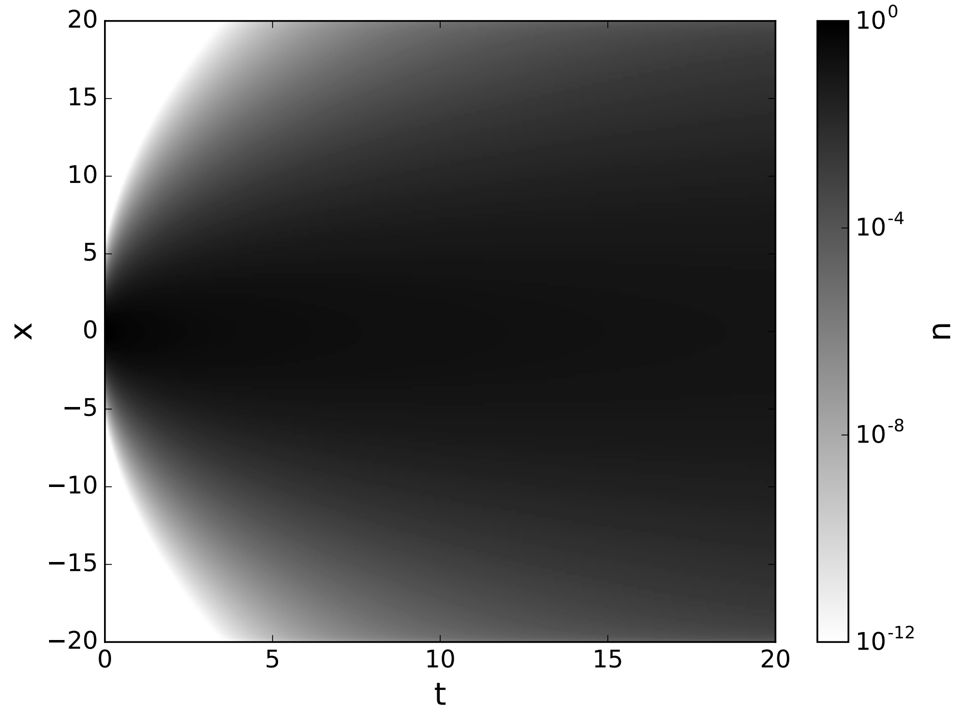


Figure 4: Solution using the Crank-Nicolson method with $r > 0.5$.