Jeremy Palmerio
I6239656
PRA3021

Topics In Scientific Computing Week 5

Exercise : The Laplacian

The laplacian exercise uses the derivation of the finite-difference method to approximate the two dimensional laplacian operator. Since matlab deals with discrete arrays and matrices we can use the discrete laplacian operator in our Euler methods seen last week. The point of this method is to input a two dimensional scalar field and then the function outputs the laplacian of that scalar field. To write this method, I simply looped over every point using nested for loops and calculated the laplacian approximation and then stored that in the output matrix. One subtlety though, the edge cases require a bit more thinking. Using a modulus function we can create periodic boundaries by looping back around to the opposite side to calculate the laplacian. To test the function I used the test function given on canvas and it confirms that it passed.

Exercise : Heat Diffusion

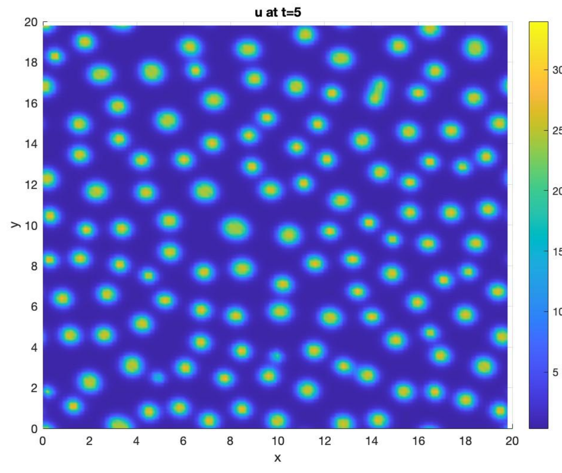
This function simulates how heat diffuses in a box. To do this, it uses the euler method described in the lecture and the previous function to calculate the laplacian of the scalar field. The simulation seems to be quite sensitive to the values of h and k. Indeed, I needed a value of $h = (k^2)/10$ to obtain meaningful results. Or else there would be errors in the output. With this function I can simulate any initial heat pattern using the input function f. The function then uses loops to initialize the t=0 step with the input f. Then the function uses the euler step to calculate the subsequent time steps and loops over all time steps. To make the output more manageable for my local memory, I added an fps input which serves to reduce the resolution of the output by sampling every fps frame and outputting that. Using this function I noticed that the kappa in the equation dictates the speed of the diffusion, the bigger the kappa the faster the equilibrium is reached. However, not every value of kappa can be simulated. I've noticed through trial and error that in general values of kappa larger than 3 break the simulation and the scalar field goes to infinity as a whole. I suppose this is due to the approximation made in the laplacian or maybe in the Euler Method.

Exercise : Wave Equation

The wave equation function is very similar to the previous diffusion equation, but with a different euler method and with two scalar fields. The two scalar fields are initialized by the input function f and g which, similarly to the diffusion function, make up the first time step of the simulation. Then using the euler method and laplacian function the subsequent time steps, or frames, are calculated and make up the output. Once again an fps argument has been supplied as an input to facilitate the creation of the animation. Some similar observations were made during this exercise than the previous. Indeed, the speed c of the wave is very iffy and can often cause the model to break down when it gets too high. Another issue I encountered was trying to vectorize the nested loop to initialize the first time slice. I could only remove one loop and apply the function to the whole vector on the y dimension but then I still needed a loop for the x dimension. Then I needed to make sure all my initial functions were written in order to process element wise operations rather than matrix operations. For the second initial conditions given in the lecture, i was unable to automate the derivation of u and v so i did them on paper and then imputed them into matlab

Exercise : Brusselator

Lastly, this exercise gave me trouble as it is very sensitive to the h and k parameters. Indeed k must be sufficiently larger than h for a working simulation. Furthermore, just like the heat diffusion, the values for kappa u and kappa v greatly impact the stability of the simulation. If either of them is greater than 10, the simulation crashes. Interestingly, I noticed the ratio of the two kappas influences the equilibrium pattern. The larger the ratio $\frac{\kappa_v}{\kappa_u}$, the smaller the dots in equilibrium. Below are some examples of equilibrium patterns for different initial parameters. Overall, this function is constructed in the exact same way as the previous two albeit a different euler method and inputs.



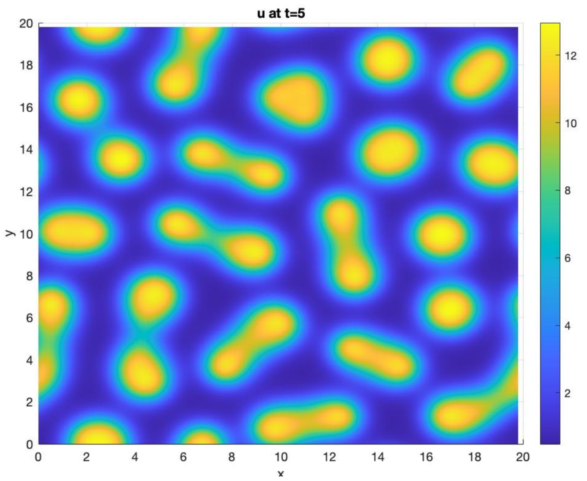
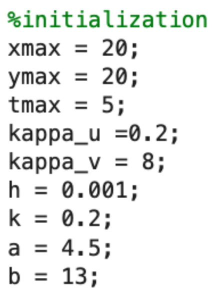
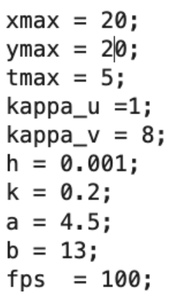


Figure 1 a & b: Left side is the brusselator solved for parameters set (a) and right side is for parameter set (b)



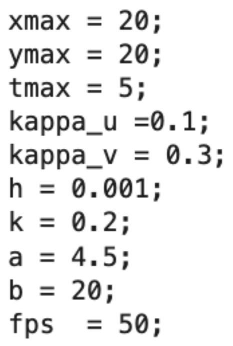
```
%initialization
xmax = 20;
ymax = 20;
tmax = 5;
kappa_u =0.2;
kappa_v = 8;
h = 0.001;
k = 0.2;
a = 4.5;
b = 13;
```



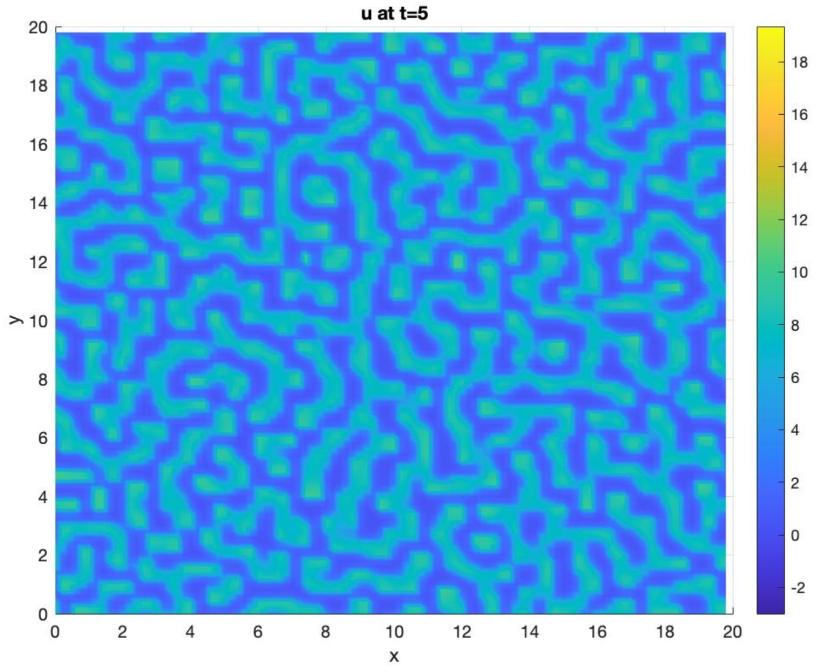
```
xmax = 20;
ymax = 20;
tmax = 5;
kappa_u =1;
kappa_v = 8;
h = 0.001;
k = 0.2;
a = 4.5;
b = 13;
fps  = 100;
```

Figure 2 a & b : Parameters set (a) and parameters set (b)

There were other interesting emerging patterns, some snake-like features which would 'eat up' other snakes and grow. In the end this resembles a cellular automaton.



```
xmax = 20;
ymax = 20;
tmax = 5;
kappa_u =0.1;
kappa_v = 0.3;
h = 0.001;
k = 0.2;
a = 4.5;
b = 20;
fps  = 50;
```


u at t=5
y
x