CompSim hand-in 1

CARL IVARSEN ASKEHAVE, (wfq585), University of Copenhagen

This report is divided into two parts. The first part is for week 1 of the course and the second part is for week 2. All the relevant formulas, concepts and figures are taken directly from the course material.

WEEK 1: INTRODUCTION

In this section of the report we will seek to find a numerical solution to the equation

$$\nabla^2 u - k^2 u = f, \tag{1}$$

where $u(\boldsymbol{x}) = u(x, y)$ is an unknown function of two variables, k is a constant, and $f(\boldsymbol{x}) = f(x, y)$ is a known function of two variables.

We will do this by dividing the domain of u into a grid of points of finite displacement, and then using the finite difference approximation to approximate the first order and second order derivatives of u. The value of u at a given point is then going to be a function of the values of u at the neighbouring points. This can be captured using a stencil to store all the influences of the points on each other. At the boundaries the points are missing either one or two of their neighbouring cells will need special attention. Here we will impose von Neumann boundary conditions and enforce them using a boundary layer of ghost nodes outside the domain.

In the end we will have translated the problem into a matrix equation, which can be solved in the computer. We will then use the `numpy.linalg` package in python to solve the problem.

1 THE FINITE DIFFERENCE APPROXIMATION

1.1 Motivation

When we want to turn a real world problem into a numerical problem that we can solve on a computer, we need to make some approximations. Often the real world problem is formulated as a mathematical model of continuous functions and their algebraic relations. In our case this is Equation (1). Since a computer doesn't have infinite computing power or infinite precision, we need to set a finite scale, and thus only approximate continuity. This is where the finite differnce approximation comes in handy.

Using *Taylor's theorem*, we can approximate a—well behaved—function while still knowing the magnitude of the error. Thus we can approximate the continuous functions of our mathematical model to finite precision and know the error that we should expect on our result. This yields theoretically incorrect results, but makes the problem viable to solve on a computer.

1.2 Derivation of 1st and 2nd order derivatives

Given a funcion $f(x)$, we can approximate it around a point p using *Taylor's theorem*:

$$f(x) = f(p) + f'(p)(x - p) + O\left((x - p)^2\right) \tag{2}$$

$$= f(p) + f'(p)\Delta x + O(\Delta x^2), \tag{3}$$

where we've defined $\Delta x = x - p$. If we move things around and call the difference we get the expression for the first derivative in the point p:

$$f'(p) = \frac{f(x) - f(p)}{\Delta x} + O(\Delta x). \tag{4}$$

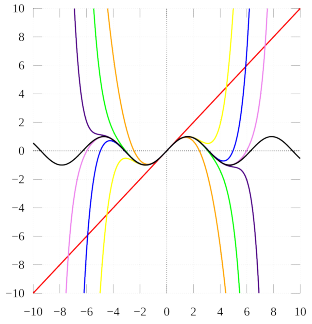

Fig. 1. Taylor approximation to of orders 1 to 7 of the function $\sin(x)$ (Source: Wikipedia)

Let's now suppose that our function is two-dimensional $f(\boldsymbol{x})$, and that it is defined on a discrete grid of $(I \times J)$ points $\boldsymbol{x}_{i,j} = (x_i,\ y_j)^T = (i\,\Delta x,\ j\,\Delta y)^T$, where $i = 1, \ldots, I$ and $j = 1, \ldots, J$. This is called our *computational mesh*, and is the domain on which our functions of interest will be defined.

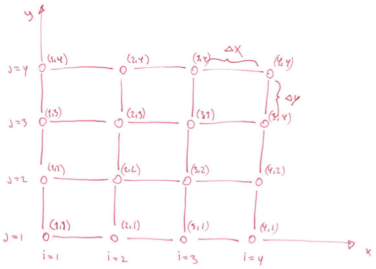

Fig. 2. The computational mesh

The value of the function in a point on the grid is denoted by $f(\boldsymbol{x}_{i,j}) = f_{i,j}$. Given that we know the value of the function in the points $\boldsymbol{x}_{i,j}$ and $\boldsymbol{x}_{i+1,j}$, we can approximate it's partial derivatives in $\boldsymbol{x}_{i,j}$ by using Equation (4)

$$\frac{\partial f_{i,j}}{\partial x} \approx \frac{f_{i+1,j} - f_{i,j}}{\Delta x}, \tag{5a}$$

$$\frac{\partial f_{i,j}}{\partial x} \approx \frac{f_{i+1,j} - f_{i,j}}{\Delta x}, \tag{5b}$$

This is called the *forward difference approximation* (FDA) as it looks *forwards* in the direction of the derivative to approximate. The *backward difference approximation* (BDA) is derived in a similar way,

but looks *backwards* in the direction of the derivative to approximate and is defined as such

$$\frac{\partial f_{i,j}}{\partial x} \approx \frac{f_{i,j} - f_{i-1,j}}{\Delta x}, \tag{6a}$$

$$\frac{\partial f_{i,j}}{\partial y} \approx \frac{f_{i,j} - f_{i,j-1}}{\Delta y}. \tag{6b}$$

If we take Equations (5) and (6) and add them together, we get

$$\frac{\partial f_{i,j}}{\partial x} \approx \frac{f_{i+1,j} - f_{i-1,j}}{2\Delta x}, \tag{7a}$$

$$\frac{\partial f_{i,j}}{\partial y} \approx \frac{f_{i+1,j} - f_{i-1,j}}{2\Delta y}. \tag{7b}$$

This is called the *central difference approximation* (CDA). One of the benefits of the CDA is that it doesn't favor any directions when approximating the derivative, and thus it is more reliable than the FDA and BDA.

Using the CDA we may approximate the second derivative of $f_{i,j}$ as

$$\frac{\partial^2 f_{i,j}}{\partial x^2} \approx \frac{\frac{\partial f_{i+\frac{1}{2},j}}{\partial x} - \frac{\partial f_{i-\frac{1}{2},j}}{\partial x}}{\Delta x} \approx \frac{f_{i+1,j} - f_{i,j} - (f_{i,j} - f_{i-1,j})}{\Delta x^2}, \tag{8}$$

where we've used discrete steps of size $\Delta x/2$ and $\Delta y/2$. This would imply a finer grid, but as we shall see this effect goes out in the final expression. Our expression now yields the approximation of the second order derivatives of our function f in the point $\boldsymbol{x}_{i,j}$

$$\frac{\partial^2 f_{i,j}}{\partial x^2} \approx \frac{f_{i+1,j} - 2f_{i,j} + f_{i-1,j}}{\Delta x^2}, \tag{9a}$$

$$\frac{\partial^2 f_{i,j}}{\partial y^2} \approx \frac{f_{i,j+1} - 2f_{i,j} + f_{i,j-1}}{\Delta y^2}. \tag{9b}$$

We see that any dependence on the finer grid of half steps has disappeared.

2 NUMERICAL SOLUTION

2.1 Formulation of the linear system

Using the first order and second order derivative approximations derived above, we can write the finite difference approximation to the governing equation (1) as

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y^2} - k^2 u_{i,j} = f_{i,j}. \tag{10}$$

Rearranging this we obtain the equation

$$\underbrace{\frac{1}{\Delta y^2}}_{c_{i,j-1}} u_{i,j-1} + \underbrace{\frac{1}{\Delta x^2}}_{c_{i-1,j}} u_{i-1,j} + \underbrace{\frac{1}{\Delta x^2}}_{c_{i+1,j}} u_{i+1,j} + \underbrace{\frac{1}{\Delta y^2}}_{c_{i,j+1}} u_{i,j+1} + \underbrace{\left(-k^2 - \frac{2}{\Delta x^2} - \frac{2}{\Delta y^2}\right)}_{c_{i,j}} u_{i,j} = f_{i,j}. \tag{11}$$

This equation is a linear combination the values of u at the neighbouring points and the point itself, with the coefficients $c_{i,j-1}$, $c_{i-1,j}$, $c_{i+1,j}$, $c_{i,j+1}$, and $c_{i,j}$ and holds true for all points on our computational mesh except the boundary poitns which we will deal with shortly. The collection of points $u_{i-1,j}$, $u_{i+1,j}$, $u_{i,j-1}$, $u_{i,j+1}$, and $u_{i,j}$ is called the stencil, and can be thought of as a geometrical "filter" moving around on top of our grid telling us which points affect the solution at the point in the center.

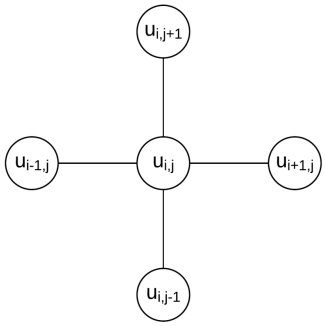

Fig. 3. The stencil of the point $u_{i,j}$

Equivalent to Equation (11) is the expression

$$u_{i,j} = g(u_{i,j}) = \frac{1}{c_{i,j}}\Big[f_{i,j} - c_{i-1,j}u_{i-1,j} - c_{i+1,j}u_{i+1,j} - c_{i,j-1}u_{i,j-1} - c_{i,j+1}u_{i,j+1}\Big], \tag{12}$$

which is called an *update formula* and defines our *update scheme* as such

$$u_{i,j}^{k+1} = g(u_{i,j}^{k}) \tag{13}$$

where g is an affine function and k is the iteration number. Our update scheme describes a *fixed point problem*, which can be reformulated as a matrix equation

$$\boldsymbol{A}\boldsymbol{u} = \boldsymbol{f}, \tag{14}$$

where $\boldsymbol{A}$ is an $(I \cdot J \times I \cdot J)$, where each row holds all the coefficients $c_{i,j}$ corresponding to a given point $u_{i,j}$ and $\boldsymbol{u}$ and $\boldsymbol{f}$ are $(1 \times I \cdot J)$ vectors that holds the values of u and f at all points on the grid. This is called the matrix assembly and in our case it looks like Figure 7.

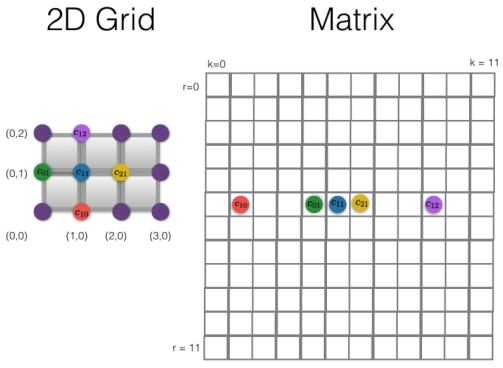

Fig. 4. The concatenation of multiple instances of the computational grid into a matrix.

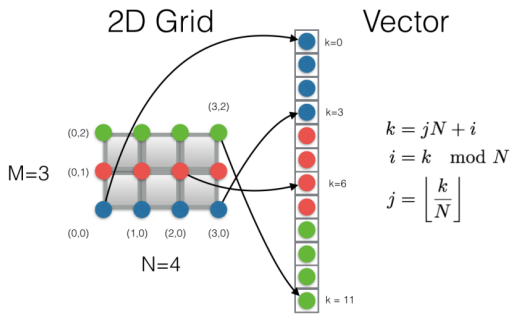

Fig. 5. The transformation of the computational grid into a vector.

2.2 Boundary conditions

While filling the $\boldsymbol{A}$ matrix in the matrix assembly, we need to pay special attention to the boundaries, as our stencil is not well defined here. At least one of the "arms" of the stecil will reach outside the domain, and thus have an undefined value.

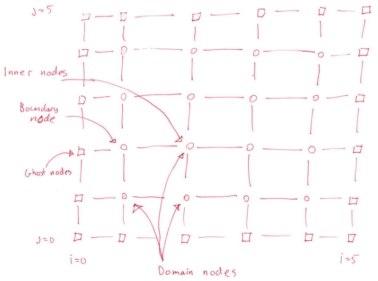

Fig. 6. Computational mesh with a layer of ghost nodes on the outside.

To deal with this, we will introduce a layer of *ghost nodes* outside our domain, the values of which we will set according to a set of von Neumann boundary conditions defined as

$$\frac{\partial u}{\partial n} = 0, \tag{15}$$

where n is the normal to the boundary. This means that the derivative of u across the boundary is zero. In the finite difference approximation, this equates to the value of the ghost nodes being equal to the value of the boundary nodes, since

$$\frac{\partial u}{\partial n} \approx \frac{u_{\text{boundary}} - u_{\text{ghost}}}{\Delta x} = 0 \tag{16}$$

$$\implies \quad u_{\text{ghost}} = u_{\text{boundary}}. \tag{17}$$

Using the FDA or BDA depending on which boundary we are dealing with, we can derive for each of the four boundaries of our domain that

$$\begin{aligned} \text{Left boundary:} \quad & u_{0,j} = u_{1,j}, \\ \text{right boundary:} \quad & u_{I-1,j} = u_{I,j}, \\ \text{lower boundary:} \quad & u_{i,0} = u_{i,1}, \\ \text{upper boundary:} \quad & u_{i,J-1} = u_{i,J}. \end{aligned} \tag{18}$$

Now we have a complete $\boldsymbol{A}$-matrix, and we can solve the linear system.

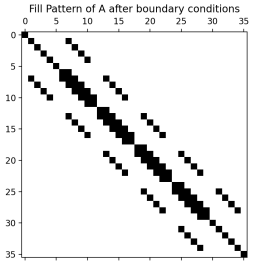

Fig. 7. Example of final $\boldsymbol{A}$-matrix for a domain of size (4×4).

3 VERIFICATION

To make sure that our interpretation of the mathematical model (1) as a matrix equation (14) is correct, we'll do two experiments trying to verify our implementation.

3.1 Experiment 1: Limiting case

One simple way to verify that our matrix equation represents our mathematical model, is to test the limiting case where $|-k^2 u| \gg |\nabla^2 u|$. Since $\nabla^2 \sim 1/dx^2$, this means that we want to set $k \gg 1/dx$.

Here Equation (1) becomes

$$-k^2 u = f, \tag{19}$$

which means that the solution u should be proportional to our input f. Let $k = 100$ and

$$f_1(x, y) = x + y \tag{20a}$$

$$f_2(x, y) = x^2 - y^2 \tag{20b}$$

$$f_3(x, y) = \sin(2\pi x) + \cos(2\pi y). \tag{20c}$$

In the first case we expect our solution to be a slanted plane, in the second it should be a saddle, and in the third it should be a sinusoidal wave in both dimensions.

With a grid of dimension (100×100) mapping onto $x, y \in [-1, 1]$, we have $1/dx = 1/dy = 50$, and thus $k \gg 1/dx$ holds for $k = 100.000$. We will use this value for all three cases.

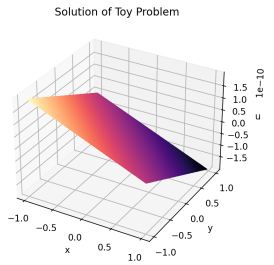

Fig. 8. Solution to the week 1 problem with $k = 10000$, $f(x, y) = x + y$, and computational mesh with dimensions $(100 \times 100) \longrightarrow [-1, 1]$.

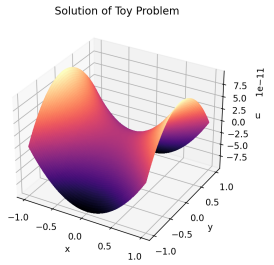

Fig. 9. Solution to the week 1 problem with $k = 10000$, $f(x, y) = x^2 - y^2$, and computational mesh with dimensions $(100 \times 100) \longrightarrow [-1, 1]$.

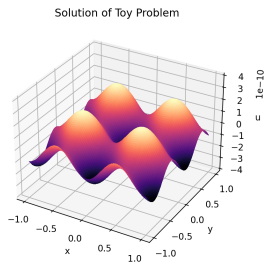

Fig. 10. Solution to the week 1 problem with $k = 10000$, $f(x, y) = \sin(2\pi x) + \cos(2\pi y)$, and computational mesh with dimensions$(100 \times 100) \longrightarrow [-1, 1]$.

In Figures 8, 9, and 10 we see that the solutions u are endeed proportional to the known f. This agrees wit the mathematical model, and is therefore a verification of our implementation.

3.2 Experiment 2:

As another experiment let's examine the contribution of the term with the laplacian. We will use the value $k = 1$ and do the simulation for the linear function f_1 and for a step function f_2, defined as such

$$f_1(x, y) = x + y \tag{21}$$

$$f_2(x, y) = \begin{cases} 1 & \text{if } x \in [-1, 0] \wedge y \in [0, 1] \\ 0 & \text{otherwise.} \end{cases} \tag{22}$$

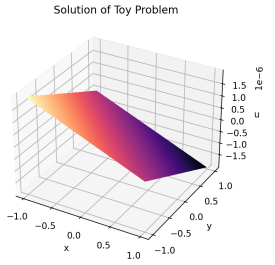

Fig. 11. Plot of $-k^2u = f$ with $k = 1$ and $f_1 = x + y$.

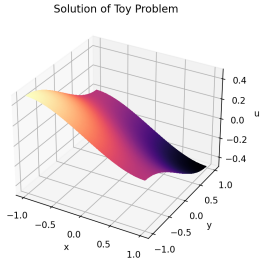

Fig. 12. Plot of full solution with $k = 1$ and $f_1 = x + y$.

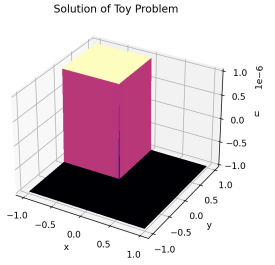

Fig. 13. Plot of $-k^2u = f$ with $k = 1$ and the step function f_2.

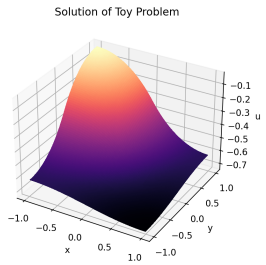

Fig. 14. Plot of full solution with $k = 1$ and the step function f_2.

We see that in the case where f has a big continuity, the full solution differs alot from only the $-k^2u = f$ plot. This is expected as the laplacian term will be of greater magnitude when the discontinuity is great. This agrees wit the mathematical model, and is therefore another verification of our implementation.

WEEK 2: INTRODUCTION

In this section of the report we will seek to find a numerical solution to the equations

$$\text{Advective flow:} \quad \frac{D\phi}{Dt} = (\boldsymbol{u} \cdot \nabla)\phi + \frac{\partial \phi}{\partial t} = 0, \tag{23}$$

$$\text{Mean curvature flow:} \quad \frac{\partial \phi}{\partial t} = -2\boldsymbol{\nabla} \cdot \frac{\boldsymbol{\nabla}\phi}{\|\boldsymbol{\nabla}\phi\|} = -2\kappa \tag{24}$$

4 ADVECTION

4.1 Semi-Lagrangian time integration

Advection is the process of transporting a quantity, such as a fluid variable or a scalar field, along the flow of a velocity field. Therefore it can be benificial to solve this problem from the perspective of the fluid particles. This is the idea behind the *semi-Lagrangian time integration*.

Given the velocity field $\boldsymbol{u}(\boldsymbol{x}, t)$, we can trace where the particle, that is currently in the grid point of our model, was at the previous time step with a simple implicit Euler step:

$$\boldsymbol{x}^{t-\Delta t} = \boldsymbol{x}^t - \boldsymbol{u}(\boldsymbol{x}^t, t)\,\Delta t. \tag{25}$$

Then we infer the value of the quantity we are interested in (in our case ϕ), at the point of the particle $\boldsymbol{x}^{t-\Delta t}$, by interpolating the values of ϕ at the nearest neighbouring grid points. We then update the value of the quantity at the grid point $\boldsymbol{x}^t$ to be the interpolated value with this expression

$$\phi^t = \phi^{t-\Delta t} + k\,\Delta t = \phi^{t-\Delta t}, \tag{26}$$

since in our case $k = (\boldsymbol{u} \cdot \nabla)\phi + \frac{\partial \phi}{\partial t} = 0$.

5 VERIFICATION

5.1 Experiment 1: Grid and time step sizes

When using the semi-Lagrangian integration scheme we have to be aware of any numerical diffusion effects caused by the interpolation and discrete time steps. We will now examine this effect by varying the time step Δt and the grid size Δx and keeping track of the numerical error. We know what the error should be, since we assume ϕ to be conserved, which means that the volume should stay the same over the integration period. We can then calculate the error of a given simulation as the difference in volume before and after simulating.

We will simulate for the values $\Delta t = [0.05, 0.01, 0.02]$ and $N = [32, 64, 128]$.

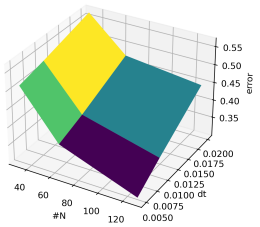

Fig. 15. A plot of the errors produced by the advection simulation for different grid divisions $N = [32, 64, 128]$ and time step sizes $\Delta t = [0.05, 0.01, 0.02]$. The minimum error was 0.302, and the maximum error was 0.568

It is clear from Figure 15 that the error decreases as the time step gets smaller and the grid gets finer, i.e. $N \to \infty$ and $dt \to 0$. This is expected, as the interpolation error decreases as the grid gets finer, and the numerical diffusion decreases as the time step gets smaller. We also see that increasing the grid size has a larger effect on the error than decreasing the time step size.

The velocity field we used to advect ϕ with, is given by $\boldsymbol{u} = (y, -x)$, which is a rotational field. This means that ideally ϕ should only rotate around the origin, and it's volume shouldn't change. We simulated a full rotation of the field. In reality, for the grid sizes and time steps we simulated with, we expect a significant error, and we expect it to be larger for the larger grid sizes due to numerical diffusion.

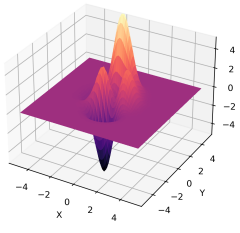

Fig. 16. Initial conditions for the advection problem with $N = 64$, $dt = 0.05$.

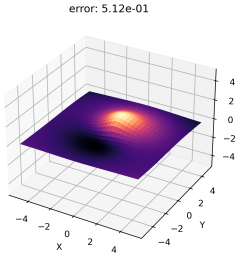

Fig. 17. Solution to the advection problem with $N = 32$, $dt = 0.05$.

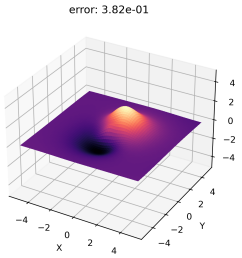

Fig. 18. Solution to the advection problem with $N = 64$, $dt = 0.05$.

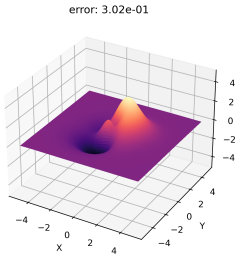

Fig. 19. Solution to the advection problem with $N = 128$, $dt = 0.05$.

It is seen from Figures 16 to 19 that the peaks are eventually more pronounced in the solutions for finer grids, just as we expected. This fits with our expectation of errors from the mathematical model, and is therefore a verification of our implementation.

5.2 Experiment 2: Rotation invariance

Another experiment we can do, is to check whether our simulation is sensitive to symmetric inputs. Here we'll do two simulations with the same initial conditions, but with the velocity fields rotating in opposite directions.

$$\boldsymbol{u}_1 = (y, -x) \tag{27}$$

$$\boldsymbol{u}_2 = (-y, x). \tag{28}$$

Ideally the error should be the same in both cases, but in reality we expect a slight numerical error between them. Again we simulate a full rotation of the field.

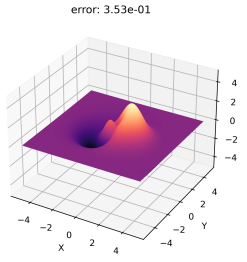

Fig. 20. Simulation of the advection problem with the velocity field $\boldsymbol{u}_1 = (y, -x)$.

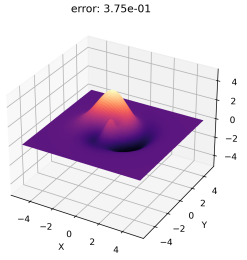

Fig. 21. Simulation of the advection problem with the velocity field $\boldsymbol{u}_1 = (y, -x)$.

It is seen from Figure 20 and Figure 21 that the difference in the errors is on around 0.02. The error we got from the (128×128) simulation in experiment 1 was around 0.4, which is much larger. This is an ok indication that our simulation is indeed rotation invariant. Although we'd need to do more experiments to be sure, with higher grid values to be completely sure.

6 MEAN CURVATURE FLOW

To solve Equation (24), we will have to rewrite κ since we cannot directly capture the concept of a divergence operator acting on the gradient of a field in terms of code. We refer to the slides for the derivation. The expression for κ is

$$\kappa = \frac{\boldsymbol{\nabla}\phi^T \boldsymbol{\nabla}\phi \ \mathrm{tr}(\boldsymbol{H}) - \boldsymbol{\nabla}\phi^T \boldsymbol{H} \boldsymbol{\nabla}\phi}{\|\boldsymbol{\nabla}\phi\|^3}, \tag{29}$$

where

$$\boldsymbol{H} = \begin{pmatrix} \frac{\partial^2\phi}{\partial x^2} & \frac{\partial^2\phi}{\partial x \partial y} \\ \frac{\partial^2\phi}{\partial y \partial x} & \frac{\partial^2\phi}{\partial y^2} \end{pmatrix}, \tag{30}$$

is the Hessian matrix of ϕ. We can now use the 1st and 2nd order FDA to approximate

$$\kappa_{ij} \approx \frac{(D_x\phi_{ij})^2 D_{yy}\phi_{ij} + (D_y\phi_{ij})^2 D_{xx}\phi_{ij} - 2D_{xy}\phi_{ij} D_x\phi_{ij} D_y\phi_{ij}}{g_{ij}^3} \tag{31}$$

whith

$$g_{ij} = \sqrt{(D_x\phi_{ij})^2 + (D_y\phi_{ij})^2}, \tag{32}$$

$$D_x\phi_{ij} = \frac{\phi_{i+1,j} - \phi_{i-1,j}}{\Delta x}, \quad D_{xx}\phi_{ij} = \frac{\phi_{i+1,j} - 2\phi_{ij} + \phi_{i-1,j}}{\Delta x^2}, \tag{33}$$

$$D_y\phi_{ij} = \frac{\phi_{i,j+1} - \phi_{i,j-1}}{\Delta y}, \quad D_{yy}\phi_{ij} = \frac{\phi_{i,j+1} - 2\phi_{ij} + \phi_{i,j-1}}{\Delta y^2}, \tag{34}$$

$$D_{xy}\phi_{ij} = \frac{\phi_{i+1,j+1} - \phi_{i+1,j-1} - \phi_{i-1,j+1} + \phi_{i-1,j-1}}{4\Delta x \Delta y}. \tag{35}$$

With κ in place, our mean curvature flow equation 24 can be integrated using a regular explicit Euler scheme

$$\phi_{ij}^{t+\Delta t} = \phi_{ij}^t - 2\kappa_{ij}\Delta t. \tag{36}$$

Note that we here keep the factor 2 which is absorbed into Δt in the slides.

6.1 Numerical considerations

Looking at Equation (31), we see that κ might become very large when $g_{ij} \to 0$, which in the limit might cause our simulation to crash. To remedy this, we can fix g_{ij} as to not go below a certain value. An example could be

$$g_{ij} = \begin{cases} g_{ij} & \text{if } g_{ij} > 1 \\ 1 & \text{otherwise} \end{cases}. \tag{37}$$

Another thing we could do is to always add a small value ϵ, i.e.

$$g_{ij} \to g_{ij} + \epsilon, \quad \epsilon > 0. \tag{38}$$

It is clear that, in this second case, ϵ should be chosen to be small enough to not affect the solution, but large enough to avoid numerical instability. We will examine the effect of these two methods in the experiments.

Another thing we realize is that if ϕ is a signed distance function, the value of g_{ij} would be 1 everywhere. This is because a signed distance function is a scalar field that assigns every point in space the distance to the nearest point on a given surface, and is negative when inside the surface, and since the distance to an object grows linearly in space, the spatial derivatives are 1 everywhere.

Therefore we elect to have the initial condition for ϕ be a signed distance function. There are some complications, though, in that, after just a single integration step, we cannot be sure whether ϕ is still a signed distance function, but we will ignore this for now.

Finally we need to make sure that the function we're trying to simulate doesn't have a curvature which is too large, as then our grid might not be able to accurately represent the geometry of the situation. The maximum curvature that our grid can represent is $\kappa_{\max} = 1/\max(\Delta x, \Delta y)$, and thus we clamp the value of the curvature as such $-\kappa_{\max} \leq \kappa \leq \kappa_{\max}$. This can also be a problem temporally, since the integration scheme we're using (36) might

make ϕ change with a greater magnitude than the size of the grid. The maximum curvature that our grid can represent is, as stated above, $\kappa_{\max} = 1/\max(\Delta x, \Delta y)$, and thus we want

$$\kappa_{\max} < C\frac{h}{2\Delta t}, \tag{39}$$

where $h = \min(\Delta x, \Delta y)$ and $0 < C < 1$ is a constant that we can set to control the stability of the simulation. This is called the Courant—Friedrichs—Lewy condition. Note that the factor 2 from (36) pops up here as well since this condition comes directly from that equation.

7 EXPERIMENTS

7.1 Experiment 1: Varying ϵ

Here we examine the effect of varying the value of ϵ in the simulation. If we set $\epsilon = 0$, we use the clamping case 37. For initial conditions we use the signed distance function provided in the exercise.

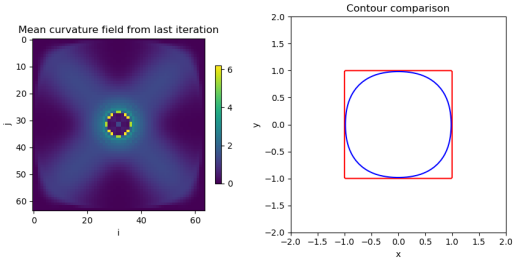

Fig. 22. Solution to the mean curvature problem with $T = 0.1$, $C = 0.5$, $I = 64$ and clamping g_{ij}.

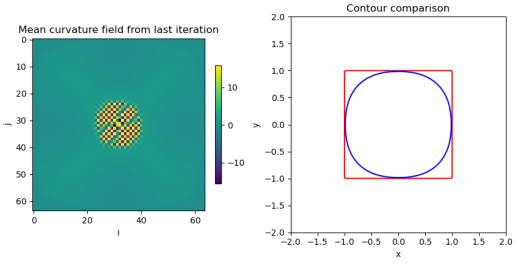

Fig. 23. Solution to the mean curvature problem with $T = 0.1$, $C = 0.5$, $I = 64$ and $\epsilon = 0.01$.

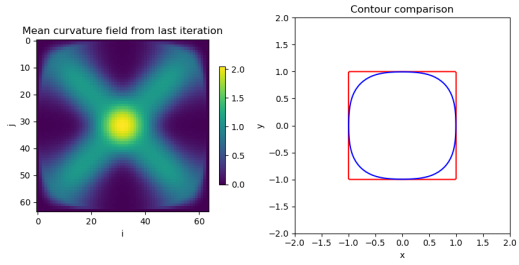

Fig. 24. Solution to the mean curvature problem with $T = 0.1$, $C = 0.5$, $I = 64$ and $\epsilon = 0.1$.

We see in Figure 22 that, in the case where we use the clamping condition on g_{ij} the discontinuities in the curvature, caused by the discontinuities in g_{ij} show up as sudden jumps in color.

In Figure 23, where $\epsilon = 0.01$, we see that the integration has become unstable, and as a result exhibits a checkered pattern in the middle with extreme discontinuities in the mean curvature. This happens since an ϵ with this value is not large enough to mitigate the consequences of g_{ij} being very small, and thus $1/(g_{ij} + \epsilon)$ blows up.

In Figure 24, where $\epsilon = 0.1$, we see that this problem has been solved. The mean curvature field is now smooth and continuous, meaning that we've hit a sweet spot with ϵ. It should be said that even though the clamping case exhibits discontinuities, it's only inside the object, and thus the solution outside should be just as valid as the one for $\epsilon = 0.1$, which is also apparent in the contour plots in the two figures for these cases, which are very similar.

7.2 Experiment 2: Varying the grid size

As with all numerical simulation and as also touched upon numerous times in this report, the grid fineness is a crucial parameter to achieving accuracy. Here we will examine the effect of varying the grid size on the solution of the mean curvature problem. We again use the initial conditions provided in the exercise.

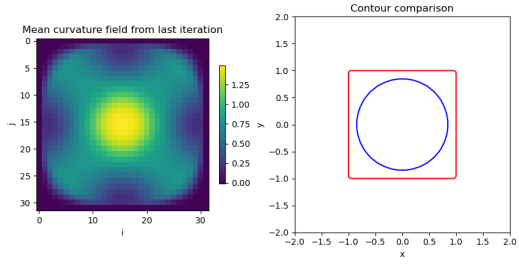

Fig. 25. Solution to the mean curvature problem with $T = 0.1$, $C = 0.25$, $\epsilon = 0.05$ and $I = 32$.

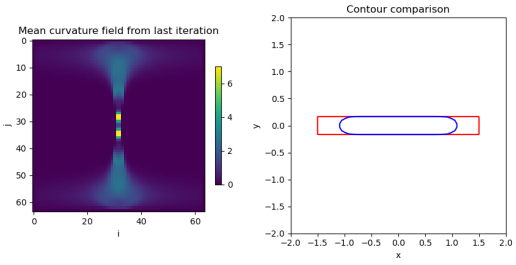

Fig. 28. First step of the solution to the mean curvature problem with a rectangular polygon.

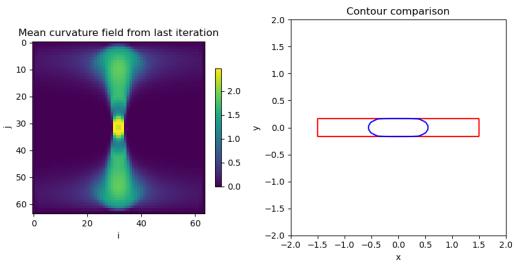

Fig. 29. Second step of the solution to the mean curvature problem with a rectangular polygon.

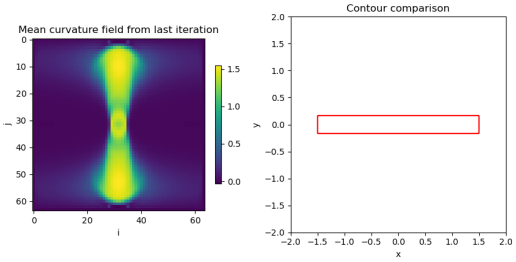

Fig. 30. Third step of the solution to the mean curvature problem with a rectangular polygon.

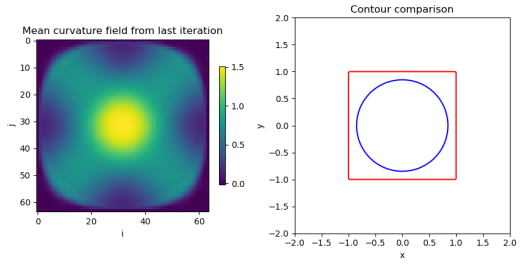

Fig. 26. Solution to the mean curvature problem with $T = 0.1$, $C = 0.25$, $\epsilon = 0.05$ and $I = 64$.

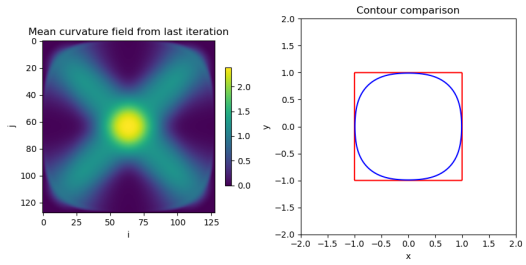

Fig. 27. Solution to the mean curvature problem with $T = 0.1$, $C = 0.25$, $\epsilon = 0.05$ and $I = 128$

We see in Figures 25 to 27 that already after $T = 0.1$, the solution is indeed sensitive to the grid fineness. This is especially seen in the respective contour plots. Here, the finest grid with $N = 128$ is still retaining some of the original square shape, while the other two cases have already colapsed into spheres. The coarsest grid with $N = 32$ even even has slight bumps on the circle showcasing it's inferior precision.

These observation fit with our mathematical predictions of the errors, and is therefore a verification of our implementation.

7.3 Experiment 3: A rectangle

In this experiment we will examine the effect the assymmetry of an object on the local mean curvature and thus the speed of shrinking due to the integration. For initial conditions replace the square with a rectangle with sides 3 and 1/3. We will use the values $I = 64$, $T = 0.1$, $C = 0.1$, and $\epsilon = 0.1$, and plot the same solution in three steps of length $t = T/3$ each. We expect the short side to shrink faster than the long side because the curvature will be greatest here.

We see that the curvature is larger on the short side, and thus the short side shrinks faster than the long side, just as expected. This fits with the mathematical model, and is therefore a verification of our implementation.