Report
Simulation Sciences Laboratory

Minimal Surfaces

add your matriculation numbers here!
Chenfei Fan
Praveen Mishra
Sankarasubramanian Ragunathan
Philipp Schleich 391779

February 22, 2020

Supervisor: Prof. Dr. Uwe Naumann
Klaus Leppkes

Contents

1. Introduction **3**

2. Background **3**
- 2.1. General background (don't like this title) 3
- 2.2. Numerical solution 5
  - 2.2.1. Discretization 5
  - 2.2.2. Solution 7
- 2.3. Consequences for a implementation 9

3. Implementation **9**
- 3.1. Software/Solver Design 9
- 3.2. Computing the Jacobian 9
- 3.3. Solving linear systems 11
- 3.4. Testing 11

4. Results **12**

5. Conclusions and outlook **12**

A. User manual @Philipp **12**

1. Introduction

@Sankar, sth like in the presentation

2. Background

2.1. General background (don't like this title)

We look at surfaces in $\mathbb{R}^3$, defined over an open set $\Omega \subset \mathbb{R}^2$. The surface of desire should contain the least possible area among all possible surfaces, that assume given values on the boundary of Ω, denoted by $\partial\Omega$. **?**

Lagrange showed in 1760, that such a surface is characterized by the graphic of a function $z(x, y)$, $z : \mathbb{R}^2 \to \mathbb{R}$, which is twice continuously differentiable on a twodimensional domain, particularly in a subset of $\mathbb{R}^2$. This function z has to fulfill the so called *Minimal-Surface Equation* (MSE) stated below.

$$\begin{aligned} (1 + z_y^2)z_{xx} - 2z_x z_y z_{xy} + (1 + z_x^2)z_{yy} = \mathcal{F}[z] &= 0 && \text{in } \Omega \qquad (1) \\ z(x, y) &= g(x, y) && \text{on } \partial\Omega \end{aligned}$$

Clearly, this formulation satisfies the prescribed boundary values given by $g(x, y)$ due to the Dirichlet boundary condition on $\partial\Omega$. As to why the graphic of functions solving this equation describes a minimal surface, we refer to the literature. For example **?** gives a very straightforward proof.

In the following, we will call the differential operator $\mathcal{F}[\cdot]$ the Minimal-Surface Operator (MSO). The resulting partial differential equation (PDE) in eq. (1) turns out to be an elliptic PDE of second order, which is in particular *non-linear*. The solution of such a PDE is not trivial, and typically requires numerical treatment. For certain cases, analytical descriptions are available, such as for Scherk's surface.

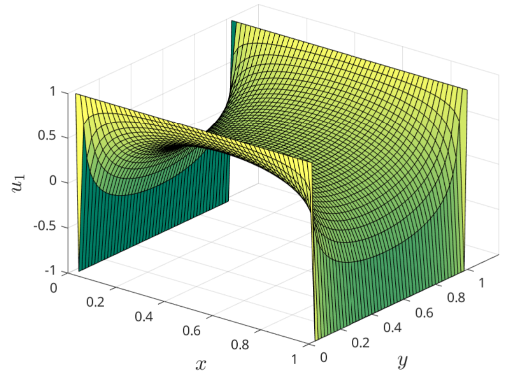

Figure 1: Scherk's first surface on $[0, 1]^2$

As an example, Scherk's first surface (compare to figure 1) Σ rescaled on $\Omega = [0, 1]^2$ is given by

$$\Sigma = \left\{ \left(x, y, \log\left(\frac{\cos(\pi(x - \frac{1}{2}))}{\cos(\pi(y - \frac{1}{2}))} \right) \right) \in \mathbb{R}^3 \middle| \, 0 < x, y < 1 \right\}. \tag{2}$$

which is the limit $n \to \infty$ of

$$\Sigma_n = \left\{ (x, y, u_n(x, y)) \in \mathbb{R}^3 \middle| \, 0 < x, y < 1 \right\}, \tag{3}$$

$$\lim_{y \to \pm 1} \to n \quad , 0 \le x \le 1 \tag{4}$$

$$\lim_{x \to \pm 1} \to -n \quad , 0 \le y \le 1. \tag{5}$$

The numerical solution of the MSE will require setting approbiate boundary conditions. Since $\log\left(\frac{\cos(\pi(x - \frac{1}{2}))}{\cos(\pi(y - \frac{1}{2}))} \right) \to \pm\infty$ on $\partial\Omega$, this is numerically not very practical.

We thus introduce $\beta_x, \beta_y \in (0, 1)$ s.th.

$$\Sigma_{\text{trunc}} = \left\{ \left(x, y, \log\left(\frac{\cos(\pi\beta_x(x - \frac{1}{2}))}{\cos(\pi\beta_y(y - \frac{1}{2}))} \right) \right) \in \mathbb{R}^3 \middle| \, 0 < x, y < 1 \right\}. \tag{6}$$

By these means, we solve the MSE on $\Omega_{\text{trunc}} = [0, 1]^2 \subset \Omega$ with $\Omega = [0, \frac{1}{\beta_x}] \times [0, \frac{1}{\beta_y}]$. Exact boundary values on $\partial\Omega_{\text{trunc}}$ ensure the correctness of the results. We choose $\beta_x = \beta_y = \beta$, as we restrict ourselves to solving on a square domain.

Later on, we will use this surface as a test-case to verify our numerical results.

2.2. Numerical solution

2.2.1. Discretization

In this project, we are supposed to solve the MSE numerically on $\Omega \equiv (0,1) \times (0,1)$. In the following, discretized quantities are indicated by a superscript h. The spatial domain is to be discretized using a structured mesh with equidistant grid spacing both in x, y, i.e. we have the same number of grid points in both directions, $N = N_x = N_y$. Thus, we define $\Omega^h := \{(x,y) \in \mathbb{R}^2 : \ (x,y) = (ih, jh), \ 0 \leq i,j < N, \ hN = 1\}$.

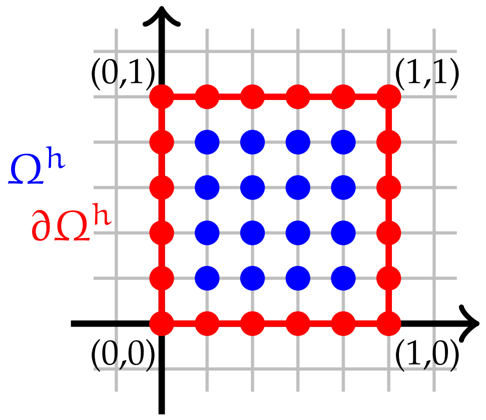

Figure 2: Depiction of Ω^h with $N = 5$, $hN = 1$. Blue: inner nodes, red: boundary nodes

We choose to discretize the MSO on Ω^h by Finite Differences, since this is usually the easiest way to go, and on a structured grid, would anyways yield similar discrete equation as in Finite Volume or Finite Element methods.

To obtain a second order consistent discrete MSO ($F^h[\cdot]$), we use central difference stencils on the first, second and mixed derivative. Since all these stencils have make only use of immediate neighours, there is no need to treat nodes close to the boundary differently, since the boundary is given by $g(\cdot)$.

This way, we obtain a discrete version of (1):

$$\begin{aligned} \left(1 + d_x[z^h]^2\right) d_{yy}[z^h] - d_x[z^h] d_y[z^h] d_{xy}[z^h] + \left(1 + d_y[z^h]^2\right) d_{xx}[z^h] = F^h\left[z^h\right] &= 0 \quad \text{in } \Omega^h \\ z^h &= g \quad \text{on } \partial\Omega^h, \end{aligned} \tag{7}$$

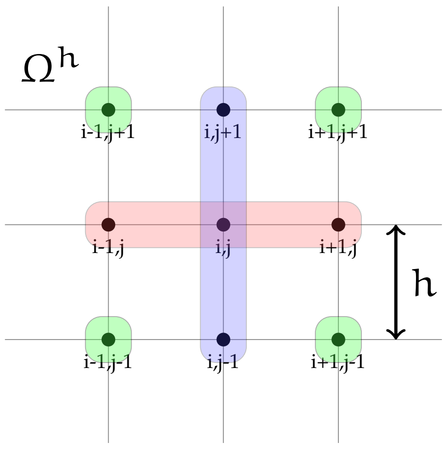

Figure 3: Snippet of the mesh with Finite Differences stencils
red/blue: first and second order stencils in x, y
green: mixed stencil

while the stencils defined on the inner nodes are given as follows ($1 \leq i, j \leq N-1$):

$$\mathtt{d}_x[z] = \frac{z_{i+1,j} - z_{i-1,j}}{2h} \tag{8}$$

$$\mathtt{d}_y[z] = \frac{z_{i,j+1} - z_{i,j-1}}{2h} \tag{9}$$

$$\mathtt{d}_{xx}[z] = \frac{z_{i+1,j} - 2z_{i,j} + z_{i-1,j}}{h^2} \tag{10}$$

$$\mathtt{d}_{yy}[z] = \frac{z_{i,j+1} - 2z_{i,j} + z_{i,j-1}}{h^2} \tag{11}$$

$$\mathtt{d}_{xy}[z] = \frac{z_{i+1,j+1} + z_{i-1,j-1} - z_{i-1,j+1} - z_{i+1,j-1}}{4h^2}. \tag{12}$$

Figure 3 depicts the stencils on a segment of Ω^h. Since the stencils have only support to the nearest neighbors, there is no need tho treat nodes close to the boundary any different. Furthermore, this gives already rise to the conclusion, that the associated matrix will be sparse (for each of the N gridpoints, only 9 instead of N partners contribute).

2.2.2. Solution

The main difficulty in solving the MSE lies in the non-linearity of $\mathtt{F}^h[\cdot]$. Since it is not possible, to directly invert for $z^h = \left(\mathtt{F}^h\right)^{-1} 0$, one needs to use a procedure such as Newton-Raphson iterations. The main idea is to use some initial guess z_0^h, while in general, $\mathtt{F}^h\left[z_0^h\right] = r_k^h \neq 0$. The goal is then to generate a sequence of z_k^h such that $r_k^h \to 0$ for increasing, but reasonably small k. Algorithm 1 shows the standard Newton-Raphson procedure, that takes as input a specific initial guess (here: 0), a tolerance for convergence TOL and the nonlinear operator $\mathtt{F}^h[\cdot]$, outputting an approximate solution to the MSE $z^h_{k_{\text{fin}}}$.

Algorithm 1 Newton's method applied on the discrete MSE

```
$k \leftarrow 0$
$z_k^h \leftarrow 0$
$r_k^h \leftarrow \mathtt{F}^h\left[z_k^h\right]$
while $\|r_k^h\| > \mathrm{TOL}$ do                    ▷ We choose $\|\cdot\| = \|\cdot\|_2$
    $z_{k+1}^h \leftarrow z_k^h - \left(\nabla \mathtt{F}^h\left[z_k^h\right]\right)^{-1} r_k^h$
    $k \leftarrow k+1$
    $r_k^h \leftarrow \mathtt{F}^h\left[z_k^h\right]$
end while
```

This algorithm involves the computation of the inverse of the gradient of $\mathtt{F}^h\left[z_k^h\right]$ for each iteration k. Later, we will present and compare two different ways on how to contrive this.

It is well known, that the Newton-Raphson procedure yields fast convergence, but the success of this convergence is highly dependent on the initial guess z_0^h. Furthermore, convergence can be improved by choosing an initial guess that is closer to the solution. One possible initial guess is to take the average boundary value, $z^h_{0,\text{ave}} = \frac{1}{|\partial\Omega^h|} \sum_{(x_i,y_j)\in\partial\Omega^h} \left(g(x_i,y_j)\right)$. While this might help convergence for surfaces, that have a certain offset with respect to the x,y-plane, it does not provide any additional information (such as preshaping curvature).

To obtain a more educated initial guess, recall the MSE, equation (1). Considering only linear terms, we get

$$\begin{aligned} \mathcal{L}[z] = z_{xx} + z_{yy} &= 0 \qquad \text{in } \Omega \\ z(x,y) &= g(x,y) \quad \text{on } \partial\Omega \end{aligned} \tag{13}$$

This linear, second order PDE is well known as the Laplace-equation, and can be solved by discretizing the second derivatives using the definitions for $\mathtt{d}_{xx}[\cdot], \mathtt{d}_{yy}[\cdot]$ introduced before, yielding $\mathtt{L}[z^h] = 0$ with the usual boundary conditions. Since Laplace's equation can be regarded as a linearization of the MSE, we suspect to get a better initial guess (and thus faster and a more robust convergence behaviour) by first solving for $z_0^h = \mathtt{L}^{-1}0$. This way, we obtain a slightly modified version of Algorithm 1, stated in Algorithm 2.

Algorithm 2 Newton's method using Laplace's Equ. as initial guess

```
$k \leftarrow 0$
$z_k^h \leftarrow \mathtt{L}^{-1}0$                                    ▷ Solve Laplace's Equ. as initial guess
$r_k^h \leftarrow \mathtt{F}^h\left[z_k^h\right]$
**while** $\|r_k^h\| > \mathrm{TOL}$ **do**
    $z_{k+1}^h \leftarrow z_k^h - \left(\nabla\mathtt{F}^h\left[z_k^h\right]\right)^{-1} r_k^h$
    $k \leftarrow k+1$
    $r_k^h \leftarrow \mathtt{F}^h\left[z_k^h\right]$
**end while**
```

2.3. Consequences for a implementation

So far we discussed the abstract numerical setting to solve the MSE. But when implementing this as a software solution, one has to consider several additional points, which especially involve the interface to the user.

3. Implementation

How we implemented things, ...

3.1. Software/Solver Design

@Sankar, here the UML chart + explain the "flow"

either change the name of "Dependencies" of make sure to explain it like during the Q&A-session

3.2. Computing the Jacobian

@Chenfei, briefly explain like in the presentation the three methods, and also name that one could use Finite Differences, but this is expected to perform bad (because...), and thus we do not use it (the next lines are already a few months old, use them or get rid of them)

Within the Newton iterations, it is necessary to compute the inverse of the Jacobian of the discrete MSO. We implemented this procedure following two different strategies:

- Based on generating the Jacobian matrix $\nabla F^h \left[z_k^h\right]$ and computing its inverse

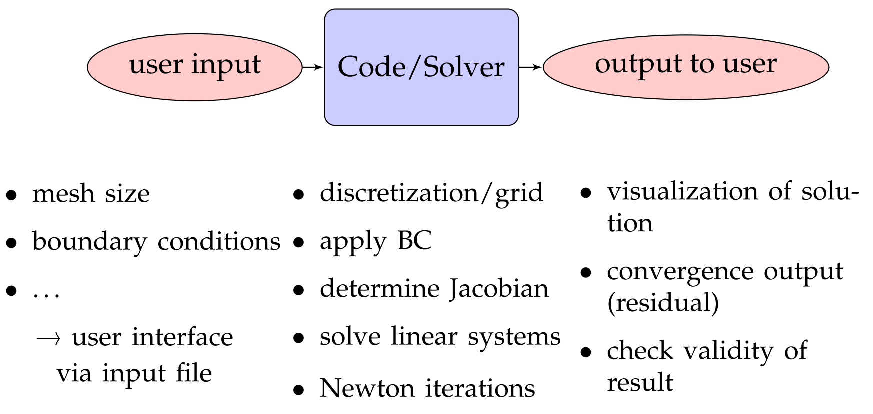

Figure 4: Requirements on a potential implementation

  - Hard-code a manually calculated Jacobian (to verify results)
  - Build a Jacobian matrix from Automatic Differention (AD by hand)

- Based on a linear iterative scheme, where we do not explicitly build the Jacobian, but extract its action on certain vectors from AD and feed this to a linear iterative solver

Argue that, since $\nabla \mathtt{F}$ is not symmetric in general, need stabilized CG (better performance than GMRES according to Eigen library)

Algorithm 3 Newton's method

```
$y \leftarrow \mathtt{F}(z)$
while $\|y\| > \text{TOL}$ do
    $A = \nabla\mathtt{F}(z)$
    $dz = A^{-1}y$          ▷ Instead of assembling the whole $\nabla\mathtt{F}$ above, do this matrix-free
    $z \leftarrow z + dz$
    $\mathtt{res} \leftarrow \mathtt{F}(z)$
end while
```

Algorithm 4 Matrix-free BiCGSTAB to get dz (adapted from **?**)

```
$z^{(1)} \leftarrow dz$
$(y, y^{(1)}) \leftarrow F^{(1)}(z, z^{(1)})$                                ▷ DCO
$p \leftarrow -y - y^{(1)}$
$r \leftarrow p, r_0 \leftarrow r, \rho \leftarrow 1, \alpha \leftarrow 1, \omega \leftarrow 1$
while $\|r\| > \text{TOL}$ do
    $\rho_{new} = (r_0, r)$
    $z^{(1)} \leftarrow p$
    $\beta \leftarrow \rho_{new}/\rho \cdot \alpha/\omega$
    $\rho \leftarrow \rho_{new}$
    $p \leftarrow r + \beta(p - \omega v)$
    $z^{(1)} \leftarrow p$
    $(y, y^{(1)}) \leftarrow F^{(1)}(z, z^{(1)})$                            ▷ DCO
    $\alpha \leftarrow \rho/(r_0, y^{(1)})$
    $dz \leftarrow dz + \alpha p$
    $s \leftarrow r - \alpha y^{(1)}$
    $z_1 \leftarrow z + dz$
    $(y_1, y_1^{(1)}) \leftarrow F^{(1)}(z_1, z_1^{(1)})$                    ▷ DCO
    if $\|y_1\| < \text{TOL}$ then
        abort                                  ▷ Converged at intermediate level
    end if
    $z^{(1)} \leftarrow s$
    $(y, y^{(1)}) \leftarrow F^{(1)}(z, z^{(1)})$                            ▷ DCO
    $\omega \leftarrow (y^{(1)}, s)/(y^{(1)}, y^{(1)})$
    $dz \leftarrow dz + \omega s$
    $r \leftarrow s - \omega y^{(1)}$
    $z \leftarrow z + dz$
end while
```

3.3. Solving linear systems

I tested a bit now, and will push also to the code, timings ideally after that: seems like sparseLU is faster for getting the initial guess, so I will implement that.
for the loops, bicgstab is still advantageous as we can greatly reduce TOL_linsolver.

3.4. Testing

@Praveen here the figure as in the presentation, short explanation (basically what you said)

maybe 1-2 results of tests, short explanation checking of validity by means of scherk, present a result here

4. Results

@Praveen you can refer to Scherk from General background, results for the 2 testcases, some timings (for all solvers for a choice of #gridpoints, not too many), scaling of the three (fix #gridpoints to sth $>$ 500 for jacOption 0,1 and maybe max. 200-300 for 2, and run for 1,2,4,8,16 threads) once timings are implemented, you can also ask us to help determine times (consistent TOL's etc...)

5. Conclusions and outlook

pretty much what we had in the presentation

@all, feel free to add ideas here, I would merge it together in the end

A. User manual @Philipp

short explanation of the params inthe input file, and how to run simulations