Report
Simulation Sciences Laboratory

Minimal Surfaces

Chenfei Fan
Praveen Mishra
Sankarasubramanian Ragunathan
Philipp Schleich

February 4, 2020

Supervisor: Prof. Dr. Uwe Naumann
Klaus Leppkes

Abstract

Abstract might be unnecessary

Contents

1 Introduction

...

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.

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

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. [if there's time, we might write this up here]

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.

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

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

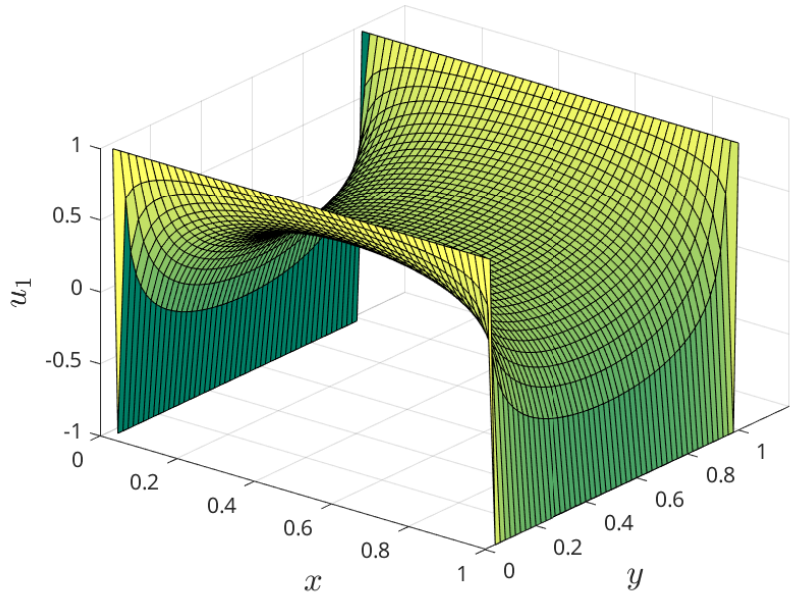

Figure 1: Scherk's first surface on $[0,1]^2$

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 will choose to test with

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

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

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

for $n = 1$.

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 $\Omega^h := \{(x,y) \in \mathbb{R}^2 : \ (x,y) = (ih, jh),\ 0 \leq i,j < N,\ hN = 1\}$.

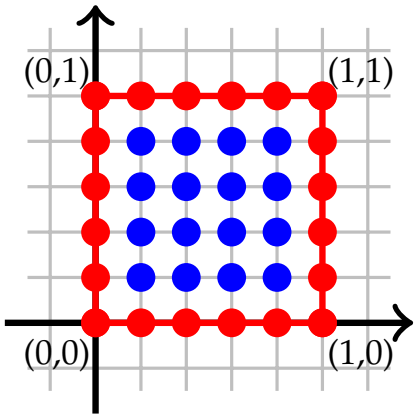

Figure 2: Depiction of Ω^h with $h = 0.2$, $hN = 1$. Blue: inner nodes, red: boundary nodes

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):

$$\begin{aligned}(1 + d_x[z^h]^2)d_{yy}[z^h] - d_x[z^h]d_y[z^h]d_{xy}[z^h] + (1 + d_y[z^h]^2)d_{xx}[z^h] = F^h\left[z^h\right] = 0 \quad &\text{in } \Omega^h \\ z^h = g \quad &\text{on } \partial\Omega^h,\end{aligned} \tag{6}$$

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

$$d_x[z] = \frac{z_{i+1,j} - z_{i-1,j}}{2h} \tag{7}$$

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

$$d_{xx}[z] = \frac{z_{i+1,j} - 2z_{i,j} + z_{i-1,j}}{h^2} \tag{9}$$

$$d_{yy}[z] = \frac{z_{i,j+1} - 2z_{i,j} + z_{i,j-1}}{h^2} \tag{10}$$

$$d_{xy}[z] = \frac{z_{i+1,j+1} + z_{i-1,j-1} - z_{i-1,j+1} - z_{i+1,j-1}}{4h^2}. \tag{11}$$

Since the stencils have only support to the nearest neighbors, there is no need tho treat nodes close to the boundary any different.

If totally bored, check consistency of the method

2.2.2 Solution

Note, that the discrete equation is not anymore consistent with the right-hand-side of (1), but yields $F^h\left[z^h\right] = r^h \neq 0$. This is due to the non-linear nature of the PDE, and the solution to (6) thus boils down to an iterative root-finding problem, i.e. in iteration k, we get $F^h\left[z_k^h\right] = r_k^h$, and we want $r_k^h \to 0$ for increasing, yet still reasonably small k. This can be achieved by the standard Newton-Raphson procedure as in Algorithm 1.

Algorithm 1 Newton's method applied on the discrete MSE

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

This algorithm involves the computation of the inverse of the gradient of $F^h\left[z_k^h\right]$ 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. To this end, recall the MSE, equation (1). Considering only linear terms, we get

$$\begin{aligned} \mathcal{L}[z] = z_{xx} + z_{yy} = 0 \quad \text{in} \qquad & \Omega \\ z(x,y) = g(x,y) \quad \text{on} \qquad & \partial\Omega \end{aligned} \tag{12}$$

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 $d_{xx}[\cdot], d_{yy}[\cdot]$ introduced before, and get $L[z^h] = 0$. 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 $z_0^h = L^{-1}0$. 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 ← 0
z_k^h ← L^{-1} 0                          ▷ Solve Laplace's Equ. as initial guess
r_k^h ← F^h [z_k^h]
while ‖r_k^h‖ > TOL do
    z_{k+1}^h ← z_k^h − (∇F^h [z_k^h])^{-1} r_k^h
    k ← k + 1
    r_k^h ← F^h [z_k^h]
end while
```

The latter two algorithms require both to determine the full Jacobian. But this would not necessarily be required to solve our problem. → matrix-free BiCGSTAB

3 Implementation

How we implemented things, ...

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 $\nabla F^h [z_k^h]$ and computing its inverse
  - 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 ∇F is not symmetric in general, need stabilized CG (better performance than GMRES according to Eigen library)

a nice flowchart (can be maybe done in powerpoint, easier here) on how the code works, plus explanations

test cases -¿ this might aswell be mentioned in results (the larger testcases)

4 Results, benchmark

A few nice pictures for stuff

timing benchmarks for different approaches (openMP and not), compare to matlab maybe

5 Conclusions and outlook

Some wise words, on what how can we extend stuff, ...