The Minimal Surface Problem

Klaus Leppkes

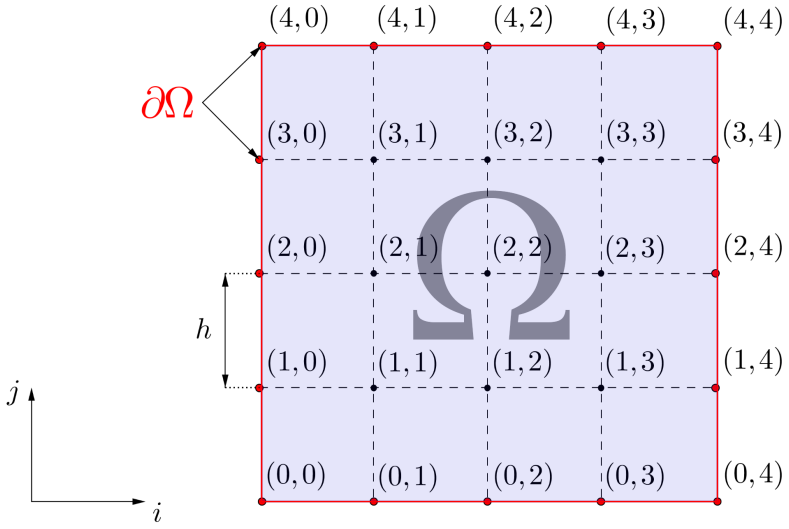
29 Oktober 2019

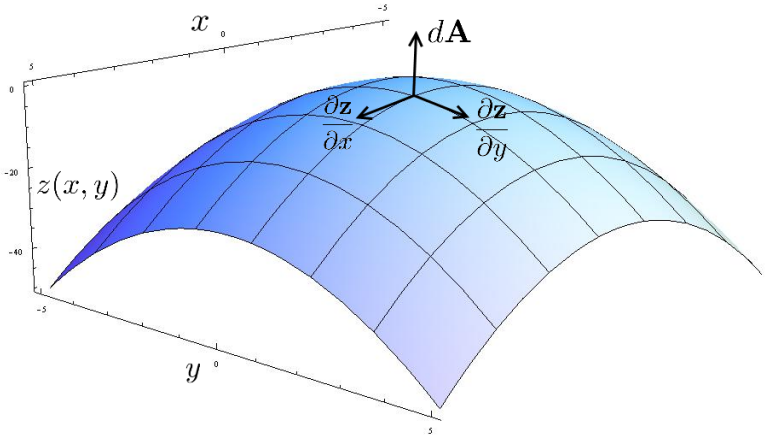
i12 Software and Tools for Computational Engineering
RWTH AACHEN UNIVERSITY

Currently

What one might want.

Modell (I) – Computational Domain





Coordinates $x, y, z \in \mathbb{R}$, where

$z = z(x, y)$

Modell (II) – Equation and Discretization

$$(1 + z_x^2) z_{yy} - 2 z_x z_y z_{xy} + (1 + z_y^2) z_{xx} = 0 \quad \text{auf } \Omega$$

$$\text{w/ boundary } z|_{\partial\Omega} = f$$

Modell (II) – Equation and Discretization

$$(1 + z_x^2) z_{yy} - 2 z_x z_y z_{xy} + (1 + z_y^2) z_{xx} = 0 \quad \text{auf } \Omega$$
$$\text{w/ boundary } z|_{\partial\Omega} = f$$

Equation
- nonlinear / linear?

Modell (II) – Equation and Discretization

$$(1 + z_x^2) z_{yy} - 2 z_x z_y z_{xy} + (1 + z_y^2) z_{xx} = 0 \quad \text{auf } \Omega$$
$$\text{w/ boundary } z|_{\partial\Omega} = f$$

Equation
– nonlinear / linear? ⇒ nonlinear elliptic differential equation

Modell (II) – Equation and Discretization

$$(1 + z_x^2) z_{yy} - 2 z_x z_y z_{xy} + (1 + z_y^2) z_{xx} = 0 \quad \text{auf } \Omega$$
$$\text{w/ boundary } z|_{\partial\Omega} = f$$

Equation
- nonlinear / linear? ⇒ nonlinear elliptic differential equation
- Of which order?

Modell (II) – Equation and Discretization

$$(1 + z_x^2) z_{yy} - 2 z_x z_y z_{xy} + (1 + z_y^2) z_{xx} = 0 \quad \text{auf } \Omega$$
$$\text{w/ boundary } z|_{\partial\Omega} = f$$

Equation
- nonlinear / linear? ⇒ nonlinear elliptic differential equation
- Of which order? ⇒ mixed derivatives of first and second order

Modell (II) – Equation and Discretization

$$(1 + z_x^2) z_{yy} - 2 z_x z_y z_{xy} + (1 + z_y^2) z_{xx} = 0 \quad \text{auf } \Omega$$
$$\text{w/ boundary } z|_{\partial\Omega} = f$$

Equation

- nonlinear / linear? ⇒ nonlinear elliptic differential equation
- Of which order? ⇒ mixed derivatives of first and second order

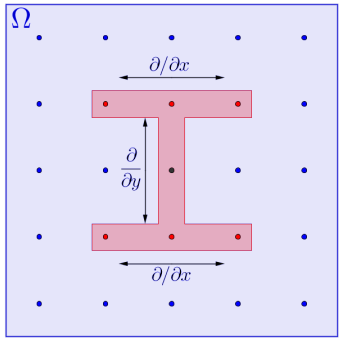
Discretization

Modell (II) – Equation and Discretization

$$(1 + z_x^2) z_{yy} - 2 z_x z_y z_{xy} + (1 + z_y^2) z_{xx} = 0 \quad \text{auf } \Omega$$
$$\text{w/ boundary } z|_{\partial\Omega} = f$$

Equation
- nonlinear / linear? ⇒ nonlinear elliptic differential equation
- Of which order? ⇒ mixed derivatives of first and second order

Discretization
- e.g. Finite Differences



mixed partial derivative z_{xy}

- or finite volume / elements

Solver

After discretization, the problem is given as n-dimensional system of nonlinear equations:

$$r^h = F(z^h) = 0, \quad z^h, r^h \in \mathbb{R}^n, \quad F : \mathbb{R}^n \to \mathbb{R}^n$$

please use approriate numerical method for solution, e.g,

Solver

After discretization, the problem is given as n-dimensional system of nonlinear equations:

$$r^h = F(z^h) = 0, \quad z^h, r^h \in \mathbb{R}^n, \quad F : \mathbb{R}^n \to \mathbb{R}^n$$

please use approriate numerical method for solution, e.g, $\Rightarrow$ Newton.

Solver

After discretization, the problem is given as n-dimensional system of nonlinear equations:

$$r^h = F(z^h) = 0, \quad z^h, r^h \in \mathbb{R}^n, \quad F : \mathbb{R}^n \to \mathbb{R}^n$$

please use approriate numerical method for solution, e.g, $\Rightarrow$ Newton.

For the computation of the Jacobian, please

- implement by-hand
- use algorithmic differentiation.

Solver

After discretization, the problem is given as n-dimensional system of nonlinear equations:

$$r^h = F(z^h) = 0, \quad z^h, r^h \in \mathbb{R}^n, \quad F : \mathbb{R}^n \to \mathbb{R}^n$$

please use approriate numerical method for solution, e.g, $\Rightarrow$ Newton.

For the computation of the Jacobian, please

- implement by-hand
- use algorithmic differentiation.

Required for validation.

Solver

After discretization, the problem is given as n-dimensional system of nonlinear equations:

$$r^h = F(z^h) = 0, \quad z^h, r^h \in \mathbb{R}^n, \quad F : \mathbb{R}^n \to \mathbb{R}^n$$

please use approriate numerical method for solution, e.g, $\Rightarrow$ Newton.

For the computation of the Jacobian, please

- implement by-hand
- use algorithmic differentiation.

Required for validation.

Use numerical library for respective solving (linear / nonlinear systems), e.g. Eigen or NAG Library.

Steps to do

work together in a team

use version management tool (`git.rwth-aachen.de`)

use testing framework and write tests for all functions (e.g. `googletest`)

Steps to do

work together in a team

use version management tool (`git.rwth-aachen.de`)

use testing framework and write tests for all functions (e.g. `googletest`)

1. discretize equation with finite differences
   - central finite differences for all derivatives

Steps to do

work together in a team

use version management tool (`git.rwth-aachen.de`)

use testing framework and write tests for all functions (e.g. `googletest`)

1. discretize equation with finite differences
   - central finite differences for all derivatives
2. implement residual function
   - use C++ and templates, i.e.

```
template <typename T>
  T residual(const std::vector<T>& z, std::vector<T>& r) {
    // computes residual entries in r[i]
    // returns sum of squares of r
  }
```

Steps to do

work together in a team

use version management tool (`git.rwth-aachen.de`)

use testing framework and write tests for all functions (e.g. `googletest`)

1. discretize equation with finite differences
   - central finite differences for all derivatives
2. implement residual function
   - use C++ and templates, i.e.

```
template <typename T>
  T residual(const std::vector<T>& z, std::vector<T>& r) {
    // computes residual entries in r[i]
    // returns sum of squares of r
  }
```

3. compute jacobian of residual function with `dco/c++` (algorithmic differentiation)

Steps to do

work together in a team

use version management tool (`git.rwth-aachen.de`)

use testing framework and write tests for all functions (e.g. `googletest`)

1. discretize equation with finite differences
   - central finite differences for all derivatives
2. implement residual function
   - use C++ and templates, i.e.

```
template <typename T>
  T residual(const std::vector<T>& z, std::vector<T>& r) {
    // computes residual entries in r[i]
    // returns sum of squares of r
  }
```

3. compute jacobian of residual function with `dco/c++` (algorithmic differentiation)
4. implement standard Newton's method

Steps to do

work together in a team

use version management tool (`git.rwth-aachen.de`)

use testing framework and write tests for all functions (e.g. `googletest`)

1. discretize equation with finite differences
   - central finite differences for all derivatives
2. implement residual function
   - use C++ and templates, i.e.

```
template <typename T>
  T residual(const std::vector<T>& z, std::vector<T>& r) {
    // computes residual entries in r[i]
    // returns sum of squares of r
  }
```

3. compute jacobian of residual function with `dco/c++` (algorithmic differentiation)
4. implement standard Newton's method
5. visualize results using `Paraview`

Steps to do

work together in a team

use version management tool (`git.rwth-aachen.de`)

use testing framework and write tests for all functions (e.g. `googletest`)

1. discretize equation with finite differences
   - central finite differences for all derivatives
2. implement residual function
   - use C++ and templates, i.e.

```
template <typename T>
  T residual(const std::vector<T>& z, std::vector<T>& r) {
    // computes residual entries in r[i]
    // returns sum of squares of r
  }
```

3. compute jacobian of residual function with `dco/c++` (algorithmic differentiation)
4. implement standard Newton's method
5. visualize results using `Paraview`
6. validate your code using symbolically known solutions (Sherk surface)
7. extend the model with dynamics; introduce mass and inertia.