Finite Element Method
Bake, Shake or Break; programming project

Student # 732095, 732064

November 17, 2014

1 Introduction

The linear elasticity equation

$$\nabla \boldsymbol{\sigma}(\boldsymbol{u}) = -\boldsymbol{f}. \tag{1}$$

describes deformation and motion on a solid object. In this report, we will present how the linear elasticity equation is applied to a geometry in 2D and 3D using the finite element method. We will look at the convergence rate on our methods, and visualize how a bridge is deformed when a car drives over it. We will also look at stress on the bridge and explain parts of our algorithm. Finally, we present a brief discussion about reducing the computational load.

2 Presentation of the 2D system

First, let us present some basic results in 2D. Using the finite element method we have made a 2D solver for the linear elasticity problem on a domain Ω. The domain is divided into triangles, called elements, each with 3 nodes. Each node represents an entry in the displacement vector

$$\boldsymbol{u}(x, y) = \begin{bmatrix} u_x \\ u_y \end{bmatrix}, \tag{2}$$

which is a measure of how much each spatial point has moved. The strain $\boldsymbol{\epsilon}$ measures how much each spatial point has deformed and the stress $\boldsymbol{\sigma}$ measures how much forces per area are acting on a spatial point. The relations between the displacement, strain and stress are listed in [4], and we will go thoroughly through this for the 3D case. Before doing that, let us show that our 2D solver is correct by looking at the error when increasing the grid size.

2.1 Convergence analysis

We test our code on the problem

$$f_x = \frac{E}{1-\nu^2}(-2y^2 - x^2 + \nu x^2 - 2\nu xy - 2xy + 3 - \nu) \tag{3}$$

$$f_y = \frac{E}{1-\nu^2}(-2x^2 - y^2 + \nu y^2 - 2\nu xy - 2xy + 3 - \nu) \tag{4}$$

with homogeneous Dirichlet boundary conditions on all boundaries. Compared to the analytical solution

$$\boldsymbol{u}(x, y) = \begin{bmatrix} (x^2-1)(y^2-1) \\ (x^2-1)(y^2-1) \end{bmatrix} \tag{5}$$

we get the error plot shown in Figure 1. Also, increasing the grid size in each spatial direction by a factor 2, we obtain an accuracy convergence of order 2, as shown in Figure 2. This implies that our code is correct.

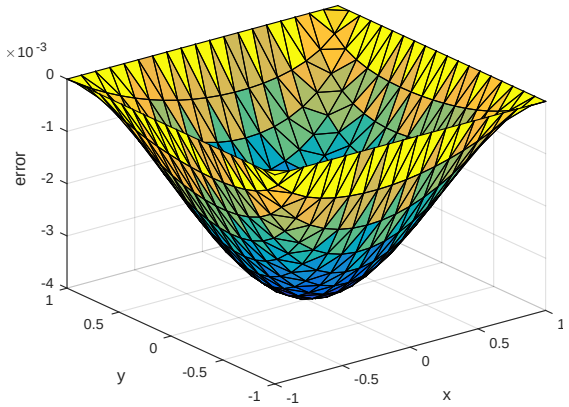


Figure 1: Error between the numerical solution to the linear elasticity problem and the analytical solution with $N = 20$ grid points in each spatial direction. The numerical solution is smaller than the analytical solution.

3 Presentation of the 3D system

Modifying the solver to 3D, the elements are now tetrahedrons with four nodes. The relation between stress and strain over one element is

$$\boldsymbol{\sigma}^e = \boldsymbol{C}\,\boldsymbol{\epsilon}^e, \tag{6}$$

where

$$\boldsymbol{\sigma}^e = \begin{bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{xy} \\ \sigma_{yz} \\ \sigma_{xz} \end{bmatrix}, \boldsymbol{\epsilon}^e = \begin{bmatrix} \epsilon_{xx} \\ \epsilon_{yy} \\ \epsilon_{zz} \\ \epsilon_{xy} \\ \epsilon_{yz} \\ \epsilon_{xz} \end{bmatrix}$$

and the stress-strain matrix [6]

$$\boldsymbol{C} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} (1-\nu) & \nu & \nu & 0 & 0 & 0 \\ \nu & (1-\nu) & 0 & 0 & 0 & 0 \\ \nu & \nu & (1-\nu) & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} \end{bmatrix}. \tag{7}$$

E is the Young's modulus which is a measure of stiffness, and ν is the Poisson's ratio describing the ratio between the compression and expansion of a material. Further the relation between strain and displacement over one element is

$$\boldsymbol{\epsilon}^e = \boldsymbol{B}\boldsymbol{u}^e \tag{8}$$

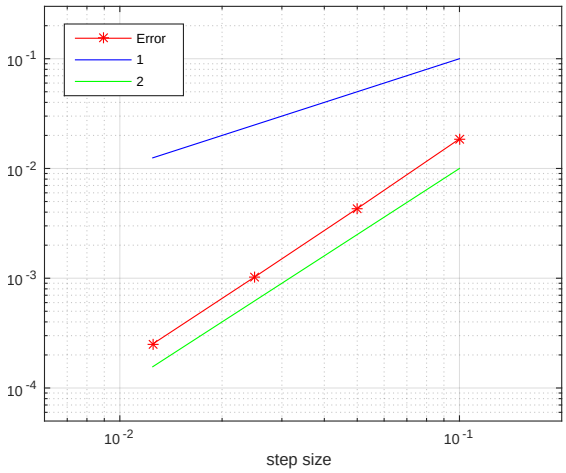


Figure 2: Loglog plot of the error showing convergence of order 2 of the linear elasticity problem with $N = 10$, 20, 40 and 80 grid points in each spatial direction.

where $\boldsymbol{u}^e$ is the displacement field over one element with four nodes, each with three spatial displacement directions, and $\boldsymbol{B}$ is the strain-displacement matrix [1].

$$\boldsymbol{u}^e = \begin{bmatrix} u_{1x} \\ u_{1y} \\ u_{1z} \\ \vdots \\ u_{4z} \end{bmatrix}, \boldsymbol{B} = \begin{bmatrix} \bar{\boldsymbol{\partial}}\boldsymbol{\phi_1} & \bar{\boldsymbol{\partial}}\boldsymbol{\phi_2} & \bar{\boldsymbol{\partial}}\boldsymbol{\phi_3} & \bar{\boldsymbol{\partial}}\boldsymbol{\phi_4} \end{bmatrix} \text{ and } \bar{\boldsymbol{\partial}} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 & 0 \\ 0 & \frac{\partial}{\partial y} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} & 0 \\ \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial x} \\ 0 & \frac{\partial}{\partial z} & \frac{\partial}{\partial y} \end{bmatrix}$$

with

$$\boldsymbol{\phi_i} = \begin{bmatrix} \phi_{i_x} \\ \phi_{i_y} \\ \phi_{i_z} \end{bmatrix}$$

being the i^{th} shape function on the current element. The shape functions are in our case linear, and $\boldsymbol{\phi_i}$ must be 1 at the i^{th} node and 0 at the three other nodes. Combining (6) and (8), we end up with the relation between displacement and stress,

$$\boldsymbol{\sigma}^e = \boldsymbol{C}\boldsymbol{B}\boldsymbol{u}^e. \tag{9}$$

This relation is later used when the displacement vector $\boldsymbol{u}^e$ is known from the solution of (1) and we want to look at the stress on each node.

4 The Finite Element Method in 3D

The linear elasticity equation (1) can be written in matrix form as

$$\begin{bmatrix} \frac{\partial}{\partial x}, & \frac{\partial}{\partial y}, & \frac{\partial}{\partial z} \end{bmatrix} \begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{xy} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{xz} & \sigma_{yz} & \sigma_{zz} \end{bmatrix} = - \begin{bmatrix} f_x, & f_y, & f_z \end{bmatrix}.$$

Let us for simplicity assume homogeneous Dirichlet boundary conditions on $\partial\Omega_D$. Multiplying with a test function $\boldsymbol{v}$ and integrating over the domain Ω, we end up with the weak formulation.

$$\int_\Omega \boldsymbol{\epsilon}(\boldsymbol{v})^T \boldsymbol{C}\boldsymbol{\epsilon}(\boldsymbol{u})\, d\Omega = \int_\Omega \boldsymbol{v}^T \boldsymbol{f}\, d\Omega \tag{10}$$

Now, we substitute the test function by linear basis functions $\boldsymbol{v} = \boldsymbol{\phi_i}$ and skipping some details we end up with the linear system for one element,

$$\boldsymbol{A}^e \boldsymbol{u}^e = \boldsymbol{b}^e. \tag{11}$$

$\boldsymbol{A}^e$ is the stiffness element matrix,

$$\boldsymbol{A}^e = \int_\Omega \boldsymbol{B}^T \boldsymbol{C}\boldsymbol{B}\, d\Omega = \boldsymbol{B}^T \boldsymbol{C}\boldsymbol{B} \cdot V^e \tag{12}$$

where V^e is the volume of the current element. Because the integrand is constant, we can substitute the integral by the volume of the domain. $\boldsymbol{b}^e$ is the element load vector with the i^{th} element

$$b_i^e = \int_\Omega \boldsymbol{\phi_i} \boldsymbol{f}\, d\Omega. \tag{13}$$

If $\boldsymbol{f}$ is a constant in the element (typically a result of mass density) then (13) can be calculated exact by a 1 order Gauss quadrature, since the basis functions are linear.

We have now presented a linear system for one element. Iterating over all elements, and mapping to a stiffness matrix $\boldsymbol{A}$ and load vector $\boldsymbol{b}$, we obtain a linear system for all the nodes in our geometry.

$$\boldsymbol{A}\boldsymbol{u} = \boldsymbol{b} \tag{14}$$

with $\boldsymbol{u}$ being the displacement vector for all the nodes in the system. The load vector $\boldsymbol{b}$ is a $3n$-vector with 3 spatial directions in each node.

4.1 Convergence analysis in 3D

Inspired by the convergence test in 2D we test our code on the problem

$$\begin{aligned}
f_x =& K^* \left[-2(y^2-1)(z^2-1) + (2\nu-1)(x^2-1)(z^2+y^2-2)\right. \\
&\left. -2x(y(z^2-1)+z(y^2-1))\right] && (15)\\
f_y =& K^* \left[-2(x^2-1)(z^2-1) + (2\nu-1)(y^2-1)(z^2+x^2-2)\right. \\
&\left. -2y(x(z^2-1)+z(x^2-1))\right] && (16)\\
f_z =& K^* \left[-2(x^2-1)(y^2-1) + (2\nu-1)(z^2-1)(y^2+x^2-2)\right. \\
&\left. -2z(x(y^2-1)+y(x^2-1))\right] && (17)\\
K^* =& \frac{E\nu}{(1+\nu)(1-2\nu)}, && (18)
\end{aligned}$$

on the reference cube $(-1,1)^3$ with homogeneous Dirichlet boundary conditions on all boundaries. This problem has the analytical solution

$$\boldsymbol{u} = \begin{bmatrix} (x^2-1)(y^2-1)(z^2-1) \\ (x^2-1)(y^2-1)(z^2-1) \\ (x^2-1)(y^2-1)(z^2-1) \end{bmatrix}. \tag{19}$$

As can be seen in Figure 3, the convergence is linear.

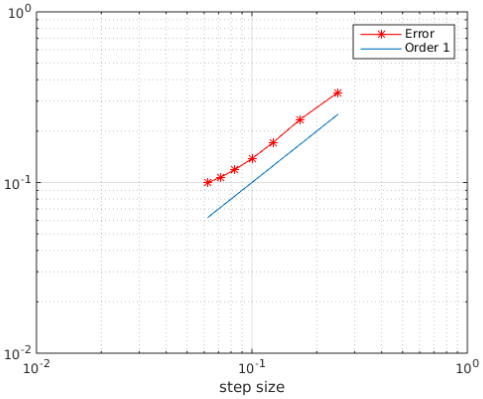


Figure 3: Logarithm error convergence plot for the 3D case, with N = 4, 6, 8, 10, 12, 14, 16 grid points in each spatial direction.

5 Stress analysis

The code has so far only calculated the displacement in the geometry. The stress $\boldsymbol{\sigma}$, which measures how much forces per area are acting on a particular spatial point, is very interesting when it comes to how much an object can withstand. To do a stress analysis we loop over all elements and using the relation (9) we get the element stress vector for each element.

Of special interest is the Von Mises stress [3], a scalar representation of the stress, since this can be directly compared to the materials yield strength to look for permanent displacement. The Von Mises stress is calculated as

$$\sigma_v = \sqrt{\frac{1}{2}\left[(\sigma_{xx}-\sigma_{yy})^2+(\sigma_{yy}-\sigma_{zz})^2+(\sigma_{zz}-\sigma_{xx})^2+6\left(\sigma_{xy}^2+\sigma_{yz}^2+\sigma_{xz}^2\right)\right]}.$$

The Von Mises stress is saved for every element, and to get the nodal stress we average the neighboring elements. If the nodal stresses are larger then the material yield strength, we'll get a permanent deformation.

6 Bridge

For this section we ventured ahead and built ourselves a bridge in minecraft as shown in Figure 4. Importing the bridge into matlab was, apart from some tedious work, quite straight forward. We defined the dimension of one minecraft block to be a cubic meter. Each block was divided into 64 elements, and this resulted in a total of $N = 6489$ nodes.

6.1 Short overview of algorithm

Since we wanted to have a time-dependency on our chosen problem, we incorporated a car driving over the bridge. This required us to manually link together the meshes for the bridge and car, and is done in the *mergeBridgeCar.m* matlab function. The supplied *hex2tetr.m* function is used to split our minecraft cubes into fitting tetrahedrons. We impose homogenous Dirichlet boundary conditions at $z = 0$ (the bottom of the bridge). *FEM.m* solves the linear elasticity problem using the finite element method further specified in Section 4. The function *stressRecovery.m* recovers the nodal Von Mises stresses according to Section 5.

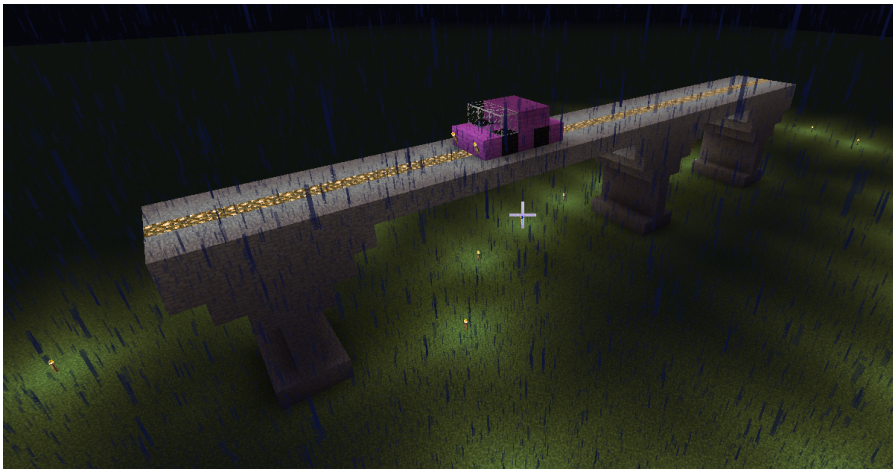


Figure 4: A picture of the bridge we built in minecraft.

Table 1: Material constants for oak wood used in the bridge.

Material constant	Value
Young's modulus	11 GPa
Density	650 kg/m^3
Compressive yield strength	46 MPa
Poissons ratio	0.3

6.2 Materials chosen

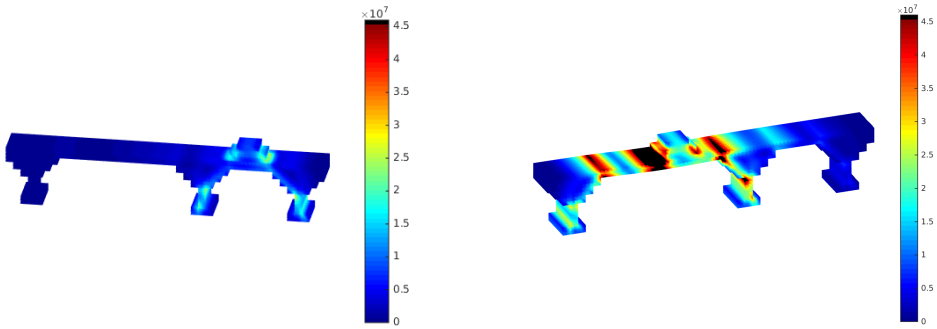
For this problem we chose the material of the bridge to be oak wood. The material constants we needed for wood were the averaged values we found at [5] and the value for the Poisson's ratio was conveniently set to 0.3. All the material constants are presented in table 1. We really wanted to see what happens when a heavy car drives over the bridge, so we defined the car to be extremely heavy. The result of this is visualized in Figure 5.

6.3 Results

The resulting Von Mises stress plot at two different times are plotted in Figure 5. A video of the car as it moves over the bridge is also produced, but is obviously not included in this text report.

7 Challenges

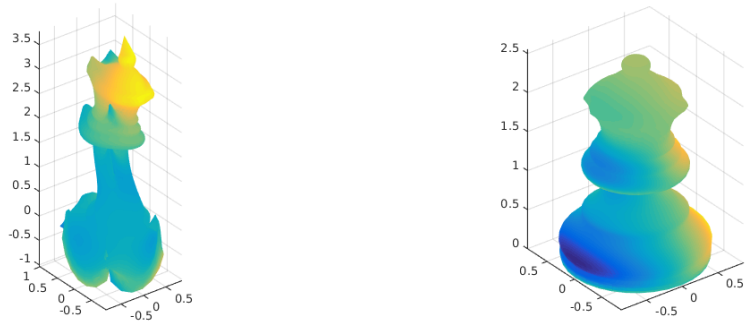
Throughout the process, we did face some challenges. We will here present some of the most important ones. When making the load vector in 3 dimensions, we forgot to map load vector to the right nodes. This resulted in weird displacements, see Figure 6, and took quite some time to figure out what was wrong. Also, the matrix (7) is only valid for an isotropic media, and is thus not valid for wood. Realizing this a little too late, we conclude that the exact same analysis could have been done with steel, which is isotropic. We also encountered a lot of computational challenges. We tried to run our algorithm on the Sydney Harbour bridge, see Figure 4, but $N = 274.461$ nodes was too large, and matlab ran out of memory.



(a) The car after 8 seconds have passed.

(b) The car after 28 seconds have passed.

Figure 5: The (really) heavy car driving over the bridge at 1 m/s. The Von Mises stress is plotted as the varying color. If the Von Mises stress is higher than the material yield, it is painted black (and the bridge will be permanently deformed). We see that at time = 28 the bridge has been severely damaged.



(a) The chess queen with faulty boundary conditioners.

(b) The same chess queen with the correct boundary conditions.

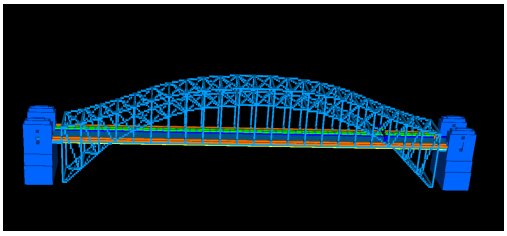
Figure 6: We see that the queen deforms quite badly when dealing with the wrong boundary conditions.

8 Computational discussions

During the project we realized we had to do some computational optimization because the problems we work can generate huge matrices. Some interesting tidbits found by running and timing matlab using the profiler we have seen that:

- Sparse matrices really outperform dense matrices in the backslash operation.
- Dense matrices really outperform sparse matrices in the assembly process of the stiffness matrix.
- Because of the previous two points, in some cases the performance is radically improved by assembling the stiffness matrix as a dense matrix and converting it to a sparse one before performing the backslash routine.
- No matter how many different combinations of sparse and dense matrices we tried, you can't get around the fact that the sparse matrices require less memory, and in some of the larger problems we tried there wasn't enough memory available to make a large enough dense matrix (even split

(a) The Sydney harbour bridge



(b) Minecraft model of the Sydney harbour bridge, downloaded from [7].

Figure 7: The Sydney harbour bridge. This bridge proved to be too computationally intensive to be solved, even when run in parallel. There was just shy of 1 million tetrahedrons in the mesh.

upon 24 instances of matlab with a clustered matrix). We gave the Sydney harbor bridge a try, see Figure 7, but the system was too large.

- Building of the element stiffness matrices can be run in parallel, which is really simple with the new parfor loop in matlab. The assembly process, however, can not as far as we can see. Since it's really the assembly that takes time (at least for sparse matrices), there's not really much performance improvement to get here.

- Our bridge problem is well suited for running in parallel, since the time steps don't depend on each other. 1419 seconds vs 86 seconds on 16 workers makes the run time 16 times faster.

References

[1] Carlos A. Felippa, *Introduction to Finite Element Methods (ASEN 5007) Fall 2014 - Chapter 14*
Colorado University,
http://www.colorado.edu/engineering/cas/courses.d/IFEM.d/IFEM.Ch14.d/
IFEM.Ch14.pdf

[2] Carlos A. Felippa, *Introduction to Finite Element Methods (ASEN 5007) Fall 2014 - Chapter 28*
Colorado University,
http://www.colorado.edu/engineering/cas/courses.d/IFEM.d/IFEM.Ch28.d/
IFEM.Ch28.pdf

[3] Wikipedia, *Article on Von Mises Stress*
http://en.wikipedia.org/wiki/Von_Mises_yield_criterion

[4] Kjetil André Johannesen, *Bake, shake or break - and other applications for the FEM*

[5] Wolfram alpha, *entry for "oak wood"*
http://www.wolframalpha.com/input/?i=oak+wood

[6] Prof. Suvranu De, *Introduction to 3D Elasticity*
Rensselaer Polytechnic Institute
http://www.learningace.com/doc/2131032/eca053bb652c3f33648a05955f1077c5
/3delasticity

[7] Minecraft user DaGamingAngel
The Sydney Harbor Bridge build project
http://www.planetminecraft.com/project/sydney-harbour-bridge-2130333/