Applications of finite element methods in biomechanics

Krister Stræte Karlsen

December 8, 2016

1 Blood flow in zebrafish

Since the 1960s, the zebrafish has become increasingly important to scientific research. It has many characteristics that make it a valuable model for studying human genetics and disease. It was the first vertebrate to be cloned and is particularly notable for its regenerative abilities. Zebrafish have a similar genetic structure to humans. They share 70 per cent of genes with us and they are cheaper to maintain than mice. The zebrafish adult is about 2.5 cm to 4 cm long.

To study the effect of different drugs being able to model the blood flow is important. For instance, if the drug actually never reaches the infected cells a potentially effective drug might be considered ineffective on wrong ground. Due to ethical reasons all experiments on zebrafish must be done at an very early stage of its development.

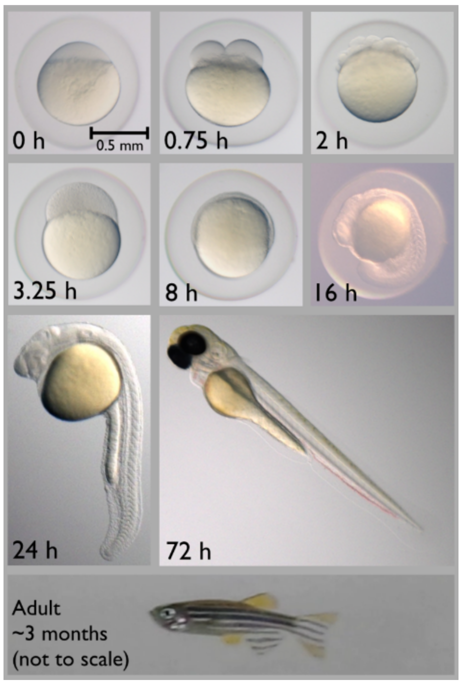

Figure 1: Stages of zebrafish development.

1.1 Measurement of blood velocities in zebrafish

The small and transparent zebrafish embryo provides an ideal animal model to get high-resolution imaging of vessels. In [1] a method referred to as *optical vector field tomography* is used to map in 3D the velocity of blood cells in the zebrafish vascular network.

Suggested project: Very little is known about the pressure gradients driving the blood flows in zebrafish. With knowledge of the velocities from [1], a mesh of geometry and a suitable model for the blood flow, do numerical experiments to find approximate values of the pressure gradients in zebrafish.

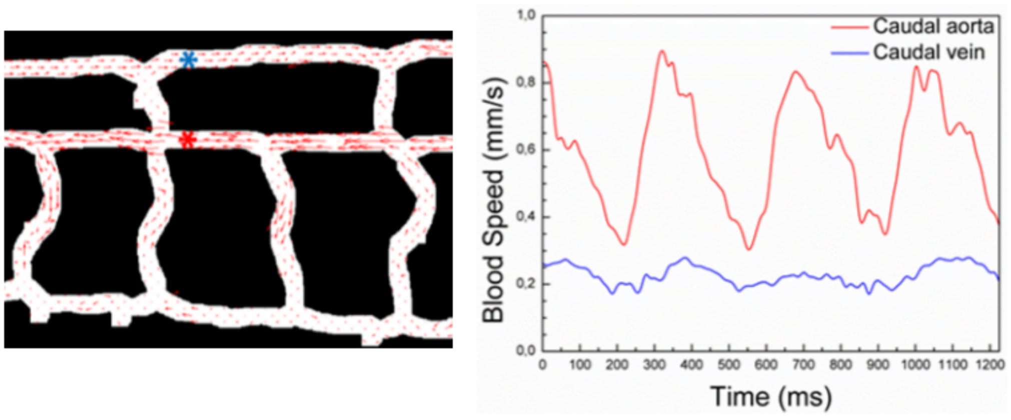

Figure 2: Some of results from [1].

1.2 Generating mesh from original MRI images

Kent: Kan du si noe om hva slags bilder dette er?

Starting from `original_zebrafish.vti` a finite element method mesh can be created using a software called *The Vascular Modeling Toolkit(VMTK)*. VMTK is a collection of libraries and tools for 3D reconstruction, geometric analysis, mesh generation and surface data analysis for image-based modeling of blood vessels. To install VMTK go to

`http://www.vmtk.org/download/`

and grab the development version in a few simple steps:

- Clone the git repository: `https://github.com/vmtk/vmtk.git`

- Create a build directory and cd into it

- Run `cmake ../vmtk` with the directory where the vmtk source tree is located as an argument

- Start the compiler in your build directory by running `make`

Next, the steps needed to create a mesh from the image-file is outlined. The reader is advised to experiment with the different parameteres used in the scripts, and the parameters suggested here should serve as a staring point. For more information see `http://www.vmtk.org/tutorials/`.

1) Select a volume of interest(VOI)

```
vmtkimagevoiselector -ifile original.vti -ofile voi.vti
```

2) Segmentation (this is the tricky and time consuming part)

```
vmtklevelsetsegmentation -ifile voi.vti -ofile levelsets.vti
```

3) Create surface file

```
vmtkmarchingcubes -ifile levelsets.vti -ofile surf.vtp
```

4) Smoothing of surface

```
dvmtksurfacesmoothing -ifile surf.vtp -passband 0.1 -iterations 30
-ofile sm_surf.vtp
```

5) Clip surface

```
vmtksurfaceclipper -ifile sm_surf.vtp -ofile cl_surface.vtp
```

6) Generate mesh

```
vmtkmeshgenerator -ifile cl_surface.vtp -ofile zebramesh.vtu
-edgelength 1.0
```

7) Convert to dolfin-format

```
vmtkmeshwriter -ifile zebramesh.vtu -entityidsarray CellEntityIds
-ofile zebra_mesh.xml
```

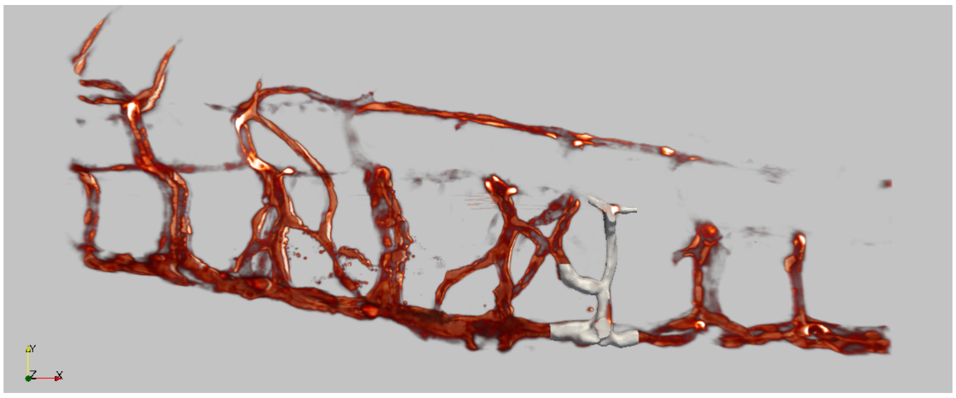

Figure 3: The circulatory system of a zebrafish where a small part is meshed.

1.3 Mathematical formulation

The blood velocities in a zebrafish are low thus using *Stokes flow* as a model is a fair approximation.

For for geometries with lots of cells using $P_1 - P_1$ formulations saves a lot of time and memory, and to even be able to run simulations on your own computer with the `zebrafish.xml` mesh such a formulation is needed.

Find $u, p \in W$, $W = V \times Q$ such that

$$a((u,p),(v,q)) = L((v,q)) \quad \forall \quad v,q \in W$$

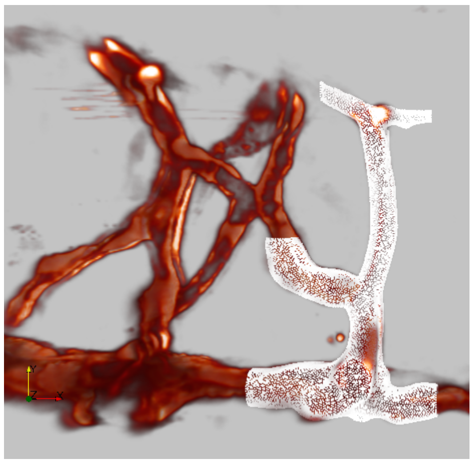
Figure 4: Zooming in to see the meshed region.

where

$$\begin{aligned} a((u,p),(v,q)) &= \int_\Omega \nabla u : \nabla v - (\nabla \cdot v)p + (\nabla \cdot u)q + \epsilon \nabla q \cdot \nabla p \, dx \\ L((v,q)) &= \int_\Omega (v + \epsilon \nabla q) \cdot f \, dx \end{aligned}$$

Here $\epsilon = \beta h^2$ and β is some number and h is the mesh cell size.

Boundary conditions:

$$\begin{aligned} u &= 0 \quad on \quad \partial\Omega_{\text{ no-slip}} \\ \sigma \cdot \mathbf{n} &= p_i \mathbf{n} \quad on \quad \partial\Omega_{\text{ opening(i)}}, \quad i = 1, .., 5 \end{aligned}$$

1.4 Results

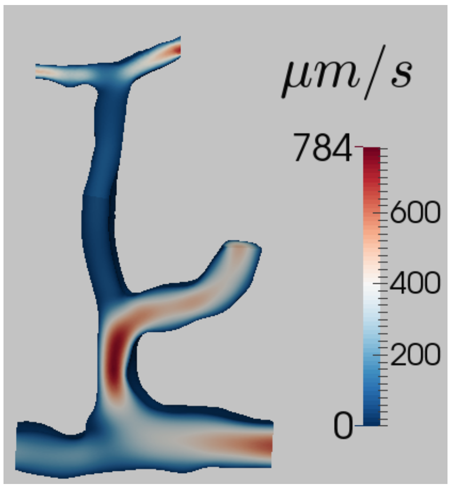

Figure 5: Zooming in to see the meshed region.

2 Squeezing a postdoc's brain

We would very much like to squeeze postdoc Erika Lindström's brain. Since she has refused to let us do this with our hands in her office, we must do this on a computer using her brain as our computational domain. The brain will be deformed as a result of the squeezing and to capture this effect we will use a *linear elastic* model.

A mesh of Erika's brain can be found in the git repository: https://github.com/krikarls/fun-with-fem.

The brain is not clamped in the skull, but in a sense floating around. This means that we must employ *neumann boundary conditions* on the entire boundary. As we know, there are no unique solution to such a problem since all *rigid motions* satify the equation. So in order to obtain a unique solution we must remove all rigid motions. All the possible rigid motions in 3D are: translations in x, y, z-direction and rotations around the corresponding axes. Thus six in total.

An example using *FEniCS* on how to remove these can be found in the same repository as the brain mesh.

2.1 Paraview

ParaView is an great open-source, multi-platform data analysis and visualization software. For instance results and other data obtained using *FEniCS* can be studied in more detial with ParaView.

2.1.1 Load/save state

Being able to save your work, and later load it, properly is important. This is done using the *save/load state* function in ParaView. If you are switching between computers, or collaborating with others, this might not work, but luckily there is an easy fix. The problem is that the state file contains information about specific path to the location to which it was saved.

As an example, let's try to open the state file `screen_shot.pvsm` from where Figure [] is taken. ParaView will very likely give you an error. To fix this open the file in some text editor, search for parts of original path, something like "/home.." and you will find lines like

```
<Element index="0"value="/home/krister/ ..../deformed_brain.pvd"/>
```

Replace

2.2 Mathematical formulation

Find u such that

$$\int_\Omega 2\mu(\epsilon(u) : \epsilon(v)) + \lambda(\nabla \cdot u)(\nabla \cdot v)\, dx = \int_\Omega f \cdot v\, dx \quad \forall v \in V$$

Boundary conditions

$$\sigma \cdot \mathbf{n} = p\mathbf{n} \quad on \quad \partial\Omega$$

The material parameters for Erika's brain are $E = 16000$ Pa and $\nu = 0.25$. The Lamè coefficients can be computed according to

$$\lambda = \frac{E\nu}{(1-2\nu)(1+\nu)} \quad and \quad \mu = \frac{E}{2(1+\nu)}.$$

2.3 Implementation

```python
from fenics import *

mesh = Mesh('mesh/res32.xdmf')  # mm

plot(mesh,interactive=True)

# Since the mesh is in mm pressure units in pascal must be
    scaled by alpha = (1e6)**(-1)
alpha = (1e6)**(-1)

# Mark boundaries
class Neumann_boundary(SubDomain):
    def inside(self, x, on_boundry):
        return on_boundry

mf = FacetFunction("size_t", mesh)
mf.set_all(0)

Neumann_boundary().mark(mf, 1)
ds = ds[mf]

# Continuum mechanics
E = 16*1e3 *alpha
nu = 0.25
mu, lambda_ = Constant(E/(2*(1 + nu))), Constant(E*nu/((1 +
    nu)*(1 - 2*nu)))
epsilon = lambda u: sym(grad(u))

p_outside = 133 *alpha
n = FacetNormal(mesh)
f = Constant((0, 0, 0))

V = VectorFunctionSpace(mesh, "Lagrange", 1)

# --------------- Handle Neumann-problem --------------- #
R = FunctionSpace(mesh, 'R', 0)          # space for one
    Lagrange multiplier
```

```
M = MixedFunctionSpace([R]*6)                    # space for all
    multipliers
W = MixedFunctionSpace([V, M])
u, mus = TrialFunctions(W)
v, nus = TestFunctions(W)

# Establish a basis for the nullspace of RM
e0 = Constant((1, 0, 0))                    # translations
e1 = Constant((0, 1, 0))
e2 = Constant((0, 0, 1))

e3 = Expression(('-x[1]', 'x[0]', '0')) # rotations
e4 = Expression(('-x[2]', '0', 'x[0]'))
e5 = Expression(('0', '-x[2]', 'x[1]'))
basis_vectors = [e0, e1, e2, e3, e4, e5]

a = 2*mu*inner(epsilon(u),epsilon(v))*dx + lambda_*inner(div
   (u),div(v))*dx
L = inner(f, v)*dx + p_outside*inner(n,v)*ds(1)

# Lagrange multipliers contrib to a
for i, e in enumerate(basis_vectors):
    mu = mus[i]
    nu = nus[i]
    a += mu*inner(v, e)*dx + nu*inner(u, e)*dx

# -------------------------------------------------------- #

# Assemble the system
A = PETScMatrix()
b = PETScVector()
assemble_system(a, L, A_tensor=A, b_tensor=b)

# Solve
uh = Function(W)
solver = PETScLUSolver('mumps') # NOTE: we use direct solver
    for simplicity
solver.set_operator(A)
solver.solve(uh.vector(), b)

# Split displacement and multipliers. Plot
u, ls = uh.split(deepcopy=True)
plot(u, mode='displacement', title='Neumann_displacement',
   interactive=True)

file = File('deformed_brain.pvd')
file << u
```

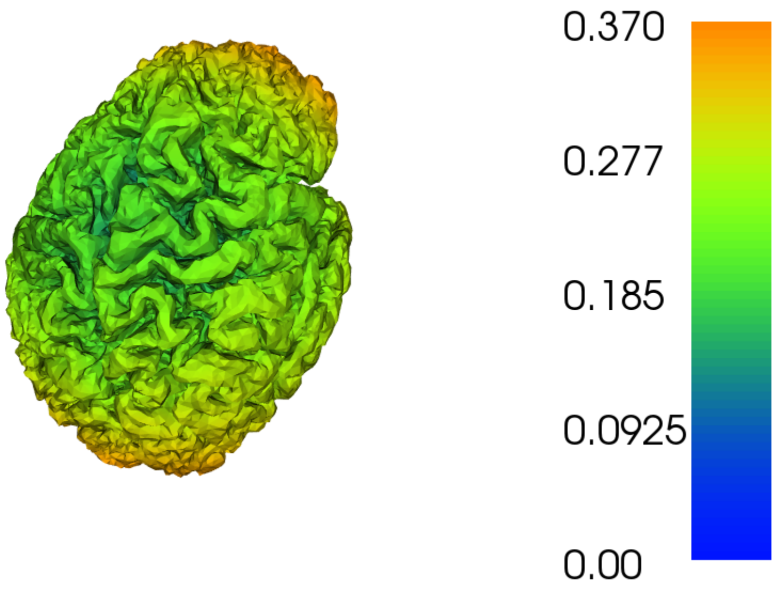

Figure 6: Numerical solution using FEniCS. Displacement measured in mm.

2.4 Results

References

[1] Luca Fieramonti, Efrem A Foglia, Stefano Malavasi, Cosimo D'Andrea, Gianluca Valentini, Franco Cotelli, and Andrea Bassi. Quantitative measurement of blood velocity in zebrafish with optical vector field tomography. *Journal of biophotonics*, 8(1-2):52–59, 2015.