Applications of finite element methods in biomechanics

Krister Stræte Karlsen

December 14, 2016

All data belonging to this chapter can be found in:
https://github.com/krikarls/fun-with-fem.git

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. Se Figure 1.

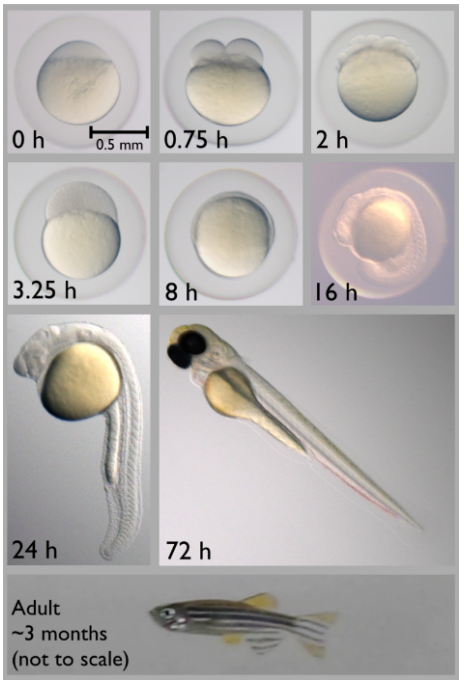

Figure 1: Stages of zebrafish development.

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. Some of the results in [1] are featured in Figure 2.

Suggested project: Very little is known about the pressure gradients driving the blood flows in zebrafish. With knowledge

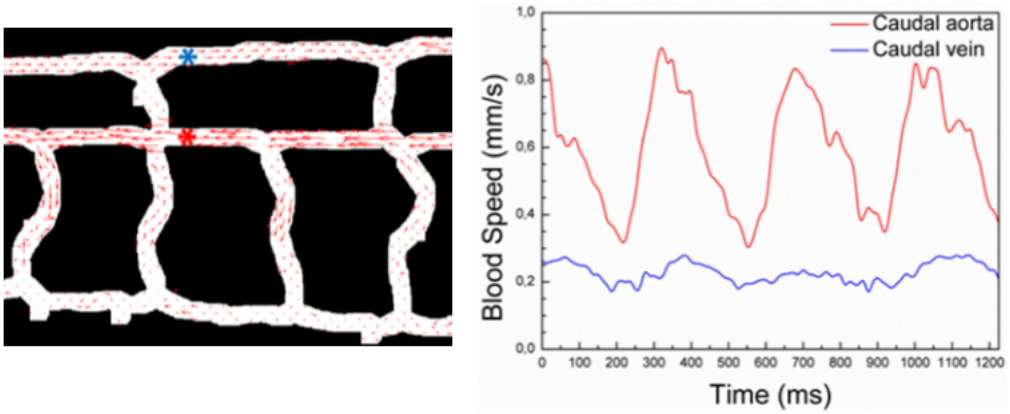

Figure 2: Some of results from [1].

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.

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 the development version of VMTK(this is strongly suggested) go through the follwing steps:

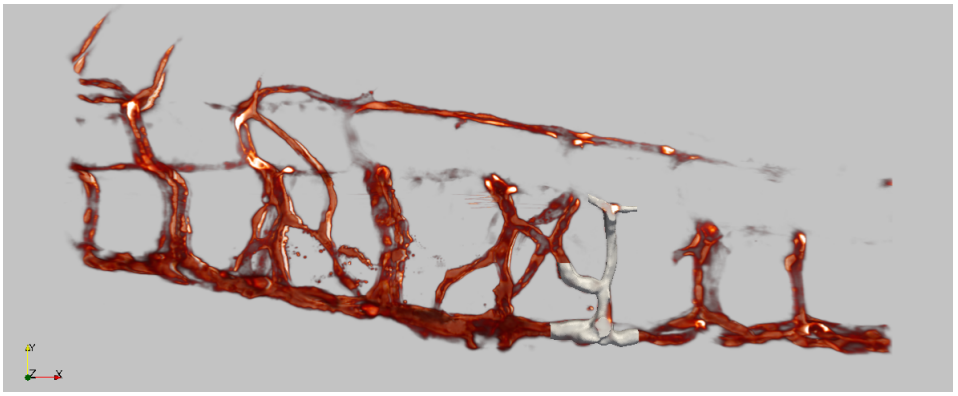

Figure 3: The circulatory system of a zebrafish where a small part is meshed.

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

2) Create a build directory and cd into it

3) Run `cmake ../vmtk` with a path to vmtk source tree

4) 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 pa-

rameters suggested here should serve only as a staring point. For more information see http://www.vmtk.org/tutorials/.

1) Select a volume of interest

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

2) Segmentation (this is tricky and very time consuming)

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

3) Create surface file

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

4) Smoothing of surface

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

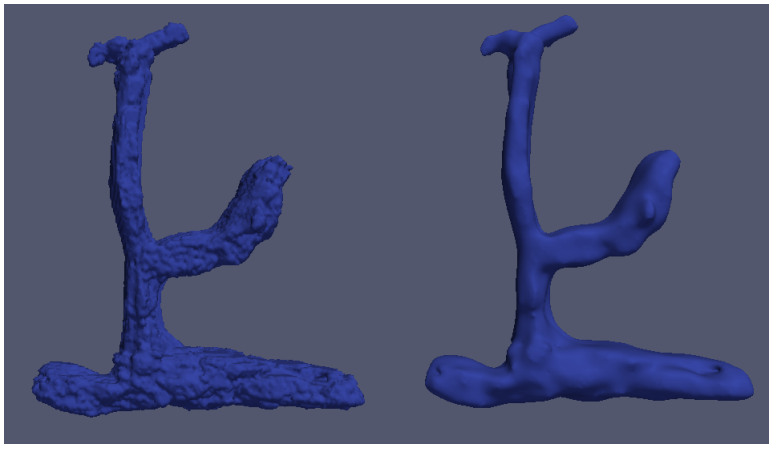

Figure 4: Surface before and after smoothing, i.e. `surf.vtp` and `sm_surf.vtp`.

5) Clip surface to create openings

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

Two of the openings in `cl_surface.vtp` should not be there. They appear because parts of the vessels lie outside the original image. These must be capped. For the capping to be successful the openings should be clipped first so that the edges are straight.

6) Cap openings

```
vmtksurfacecapper -ifile cl_surf.vtp -ofile cap_surf.vtp
```

7) Remesh. This is usually needed after clipping and capping, see Figure 5.

```
vmtksurfaceremeshing -ifile cap_surf.vtp -ofile remeshed_surf.stl
```

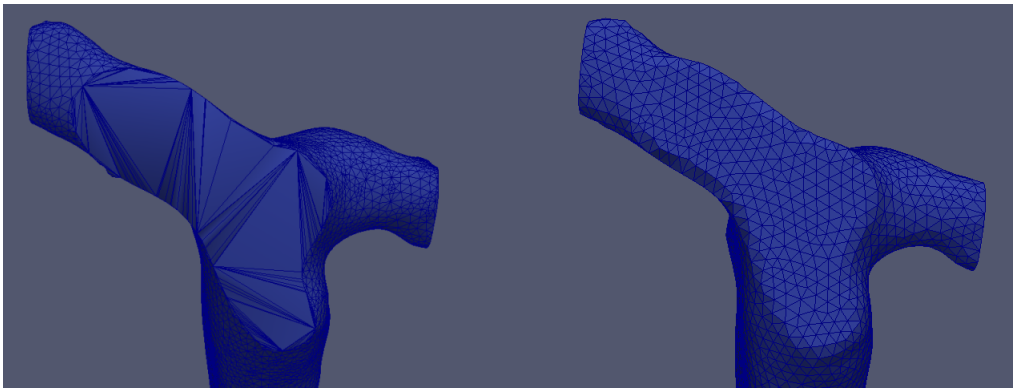

Figure 5: Before and after remeshing

In order to successfully generate a volumemesh using VMTK the surface has to perfect and there's a good chance that remeshing was not sufficient. Luckily there are other software with excellent cleaning filters to take care of the job. For example the open source software *MeshLab*, available in ubuntu repository.

Next MeshLabs clean filter called *remove isolated pieces(wrt diameter)* is used to create `cleaned_surf.stl`. This is now a perfectly good surface ready for mesh generation!

8) Generate mesh

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

You'll see that for this mesh `edgelength 1.0` gives a very fine mesh that calls for the need for a computer cluster. So some bigger edgelength is recommended.

9) Convert to dolfin-format

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

The mesh can now be imported with FEniCS and blood can start flowing.

```
1 from fenics import *
2 mesh = Mesh('mesh.xml')
3 plot(mesh,interactive=True)
```

Marking openings in FEniCS

Subdomains of the inlets and outlets need to be made and for complex geometries that might not be trivial. If the surface was clipped in a clever way, the job can still be pretty easy. A clever way to clip is placing the openings orthogonal to the coordinate axes(unless you love finding equations of planes).

1) Open the final surface file in ParaView.

2) Select the points around one opening and use *extract selection*.

3) The range of the points(*"bounds"*) can now be found and marked in FEniCS.

Example:

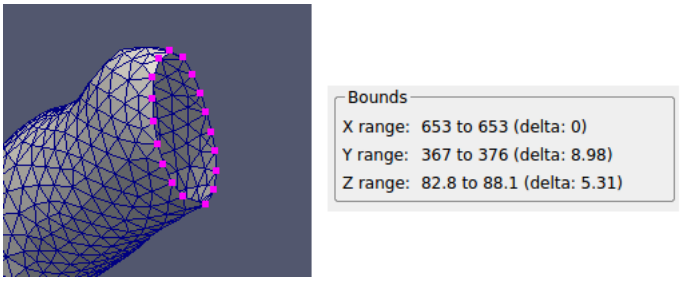

Figure 6: Marking of points and finding their range in ParaView.

From Figure 6 we can see that the points defining the opening is in the plane $x = 653$ and lie above some y-value. This can be marked in FEniCS as in the example below.

```
1 class Outlet(SubDomain):
2     def inside(self, x, on_boundry):
3         return (x[0] > 653-eps) and (x[1] > 360.) and
              on_boundry
```

Prescribing inlet and outlet velocities on such oddly shaped openings can be tricky. It is possible to extend the openings circular tubes so that parabolic velocity profiles can be used. Another option is to instead set the pressure.

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

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

where

$$a((u,p),(v,q)) = \int_\Omega \nabla u : \nabla v - (\nabla \cdot v)p + (\nabla \cdot u)q + \epsilon \nabla q \cdot \nabla p \, dx$$
$$L((v,q)) = \int_\Omega (v + \epsilon \nabla q) \cdot f \, dx$$

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

Boundary conditions:

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

1.4 Implementation

```
1  from dolfin import *
2
3  parameters["krylov_solver"]["relative_tolerance"] = 1.0e-8
4  parameters["krylov_solver"]["absolute_tolerance"] = 1.0e-8
5  parameters["krylov_solver"]["monitor_convergence"] = False
6  parameters["krylov_solver"]["maximum_iterations"] = 10000
7
8  mesh = Mesh('../mesh/zebrafish_mesh.xml.gz')     # micrometre
       (um)
9
10 # Mark opening(numbered bottom to top, left to right)
11
12 class NoSlip(SubDomain):
13     def inside(self,x,on_boundry):
14         return on_boundry
15
16 class Opening1(SubDomain):
17     def inside(self, x, on_boundry):
18         return (x[0] < 602.727) and on_boundry
19
20 class Opening2(SubDomain):
21     def inside(self, x, on_boundry):
22         return (x[0] > 710.) and on_boundry
23
24 class Opening3(SubDomain):
25     def inside(self, x, on_boundry):
26         return (x[0] < 640.) and (x[1] > 292.) and
              on_boundry
27
28 class Opening4(SubDomain):
29     def inside(self, x, on_boundry):
30         return (x[1] > 300.) and (x[0] < 648.) and
              on_boundry
31
```

```
32 class Opening5(SubDomain):
33     def inside(self, x, on_boundry):
34         return (x[1] > 300.) and (x[0] > 707.5) and
               on_boundry
35
36 mf = FacetFunction("size_t", mesh)
37 mf.set_all(6)                #       _______
38 NoSlip().mark(mf,0)          #     4__     __5
39 Opening1().mark(mf,1)        #        |   |
40 Opening2().mark(mf,2)        #       __|   |
41 Opening3().mark(mf,3)        #     3__     |
42 Opening4().mark(mf,4)        #        |    |
43 Opening5().mark(mf,5)        #     ___|    |____
44 #plot(mf,interactive=True)   #    1___________2
45
46 # Assign normal mesh function
47 n = FacetNormal(mesh)
48 ds = ds[mf]
49
50 # Define spaces and functions
51 V = VectorFunctionSpace(mesh, "Lagrange", 1)
52 Q = FunctionSpace(mesh, "Lagrange", 1)
53 W = V*Q
54 w = Function(W)
55 (u, p) = TrialFunctions(W)
56 (v, q) = TestFunctions(W)
57
58 # Parameters
59 h = CellSize(mesh)
60 beta = Constant(0.2)          # stabilization factor
61 mu = Constant(3.5E-9)
62
63 # Pressure
64 dp = 9.375E-8                 # pressure gradient in uPa/um
65 p1 = Constant(dp*20)
66 p2 = Constant(0)
67 p3 = Constant(0)
68 p4 = Constant(dp*150)
69 p5 = Constant(0)
70
71 # Boundary condition
72 noslip = DirichletBC(W.sub(0), Constant((0,0,0)), mf, 0)
73
74 # Define variational problem
75 f = Constant((0,0,0))
76
77 a = (mu*inner(grad(u), grad(v))*dx + div(v)*p*dx \
78     + div(u)*q*dx - beta*h*h*inner(grad(p), grad(q))*dx)
79
```

```
b = (mu*inner(grad(u), grad(v))*dx + p*q/mu*dx \
    + beta*h*h*inner(grad(p), grad(q))*dx)

L =  inner(v + beta*h*h*grad(q), f)*dx \
   + inner(v,p4*n)*ds(4) + inner(v,p5*n)*ds(5) \
   + inner(v,p3*n)*ds(3)  \
   + inner(v,p1*n)*ds(1) + inner(v,p2*n)*ds(2)

# Assemble system
(A, bb) = assemble_system(a, L, noslip)
(P, _) = assemble_system(b, L, noslip)

# Set solver and preconditioner
solver = KrylovSolver("gmres", "hypre_amg")
solver.set_operators(A, P)

U = Function(W)

# Solve
import time
start_time = time.time()

solver.solve(U.vector(), bb)

print ' \n Time used to solve system:', \
       (time.time()-start_time)/60, 'min'

# Get sub-functions
u, p = U.split()

ufile = File("velocity.pvd")
pfile = File("pressure.pvd")
ufile << u
pfile << p

plot(u)
plot(p)
interactive()
```

1.5 Results

Using the implementation suggested in the previous section realistic velocities can be obtained. See Figure 7.

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

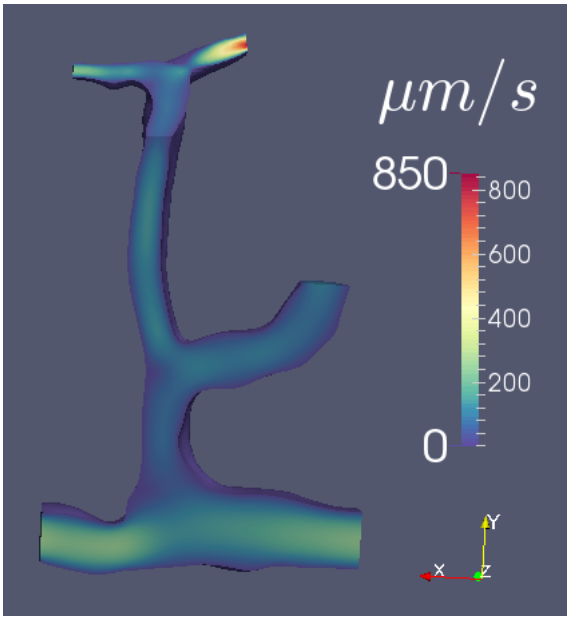

Figure 7: Velocity profile obtained from running the code from the implementation section, here viewed in ParaView. .

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.

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 generated using *FEniCS* can be studied in more detial with ParaView.

2.1.1 Load/save state

Being able to save your work and later properly load it is important. This is done using the *save/load state* function in ParaView. If you are switching between computers, or collaborating with others, you might get some errors loading state files. This is because the state file uses the path to `.pvd`-file on which is was originally built, and that path is usually different on different computers.

Example: Let's try to open the state file `screen_shot.pvsm` from where Figure [] is taken. ParaView will likely give you an error. To fix this open the file in a text editor, search for the name of the `.pvd`-file and you will fine one or more lines like:

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

Correct the path and load the state file in ParaView.

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

```
1 from fenics import *
2
3 [Elasticity code to be included here]
```

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.

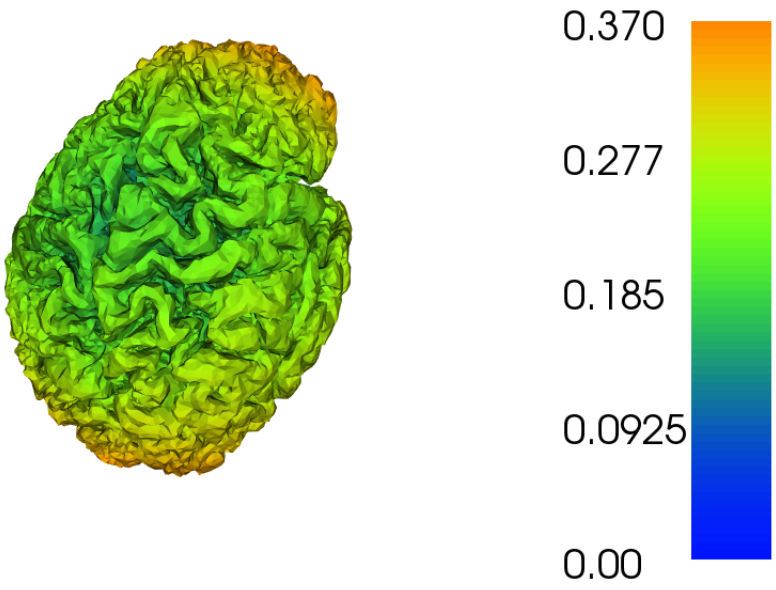

Figure 8: Numerical solution using FEniCS. Displacement measured in mm.