Modelling evolving surface tension in FEniCS

Candidate Number: 1003876

July 4, 2016

Contents

Introduction

In this project we are concerned with shapes corresponding to minimal surfaces. We also consider possible time evolving extensions and simulate change of the minimal surface in time. Furthermore, to be able to model surface tension we use the Laplace-Young equation that is somewhat similar to the initial minimal surface equation. This is done by assuming that the pressure difference is proportional to the height of a shape. Following this, we want to enforce dynamic behaviour on the surface tension. We employ the same approach as Gauglitz and Radke in [1], which is exteneded for the 2D domain. Solutions are considered for various initial conditions and meshes generated by *mshr* package.

All of the numerics is performed in the Python/C++ library FEniCS [2]. This is a powerful framework that allows the user to specify the problem in high-level mathematical abstractness and then use Finite Elements solvers to find an approximate solution. Although we write code in Python, FEniCS is able to compile parts of it in C++ which makes the implementation much more time efficient. On one hand, this allows for a very short and concise code, while the C++ implementation ensures that our code is executed in a well-optimized low-level language. Our entire code and results can be seen electronically in the anonymized Github repository[1].

[1] `https://github.com/can1003876/python`

1 Minimal surface equation and its evolution in time

We consider a smooth surface in $\mathbb{R}^3$, which we represent as the function $u : \Omega \to \mathbb{R}$, defined on a bounded open set $\Omega \subset \mathbb{R}^2$. The area of this surface can be computed by the following functional:

$$A(u) = \int_\Omega \left(A + |\nabla u|^2\right)^{1/2} \mathrm{d}x. \tag{1}$$

The minimal surface problem than corresponds to the minimization of $A(u)$ given that we have prescribed boundaries, i.e. we fix the surface at boundary $\partial\Omega$. Using the calculus of variations [3] we derive that the necessary condition for the solution is the PDE equation:

$$\nabla \cdot \left(\underbrace{\left(1 + |\nabla u|^2\right)^{-1/2}}_{q} \nabla u \right) = 0. \tag{2}$$

Note that term q, which represents curvature of the surface also causes non-linearity in this equation.

If we wish to have an evolving surface, we can add additional time dependence on the right hand side:

$$\nabla \cdot \left(\underbrace{\left(1 + |\nabla u|^2\right)^{-1/2}}_{q} \nabla u \right) = \frac{\partial u}{\partial t}. \tag{3}$$

We derive the weak formulation of the problem at every time and solve it using the Newton method. To find the evolving solution, we discretize $\frac{\partial u}{\partial t}$ in time and update the solution using the zero-stable Backwards-Euler.

$$\frac{u_n - u_{n-1}}{\Delta t} = \nabla \cdot (q_n \nabla u_n)$$
$$u_n - \Delta t \nabla \cdot (q_n \nabla u_n) = u_{n-1}$$
$$\int_\Omega u_n v \, \mathrm{d}x - \Delta t \int_\Omega \nabla \cdot (q_n \nabla u_n)\, v \, \mathrm{d}x = \int_\Omega u_{n-1} v \, \mathrm{d}x$$
$$\int_\Omega u_n v \, \mathrm{d}x - \Delta t \int_\Omega \nabla \cdot (q_n \nabla u_n v) - q_n \nabla u_n^T \nabla v \, \mathrm{d}x = \int_\Omega u_{n-1} v \, \mathrm{d}x.$$

To get the last line we use the product rule for divergence. Now we can apply the divergence theorem to get the final weak form:

$$\int_\Omega u_n v \, \mathrm{d}x - \Delta t \int_{\partial\Omega} q_n \underbrace{\nabla u_n^T \vec{n}}_{B} v \, \mathrm{d}S + \Delta t \int_\Omega q_n \nabla u_n^T \nabla v \, \mathrm{d}x = \int_\Omega u_{n-1} v \, \mathrm{d}x. \tag{4}$$

The boundary term denoted as B corresponds to the Neumann boundary condition. Using this, we can prescribe an additional condition on the surface in the form of its contact angle θ with the boundaries. We formulate the following non-linear variational problem which can be solved in FEniCS:

$$\int_\Omega u_n v \, \mathrm{d}x - \Delta t \int_{\partial\Omega} q_n B v \, \mathrm{d}S + \Delta t \int_\Omega q_n \nabla u_n^T \nabla v \, \mathrm{d}x = \int_\Omega u_{n-1} v \, \mathrm{d}x \tag{5}$$

$$u = u_0 \qquad \text{at } t = 0 \tag{6}$$

$$u = u_0 \qquad \text{on } \partial\Omega \tag{7}$$

$$B = \left(1 + |\nabla u_n|^2\right)^{1/2} \cos\theta \qquad \text{on } \partial\Omega. \tag{8}$$

We then iterate through time and solve using the Newton's method at every time step until the difference of two consecutive surface areas is less than 10^{-5} or until the time $T = 20$ is reached.

The weak form for the static minimal surface can be derived in the same way and gives us the following problem formulation:

$$\int_{\partial\Omega} q B v \, \mathrm{d}S + \int_\Omega q \nabla u^T \nabla v \, \mathrm{d}x = 0 \tag{9}$$

$$B = \left(1 + |\nabla u|^2\right)^{1/2} \cos\theta \tag{10}$$

$$u = u_0 \qquad \text{on } \partial\Omega. \tag{11}$$

Both of these problems were implemented in FEniCS for the rectangle and unit circle domains Ω. The initial guess has been chosen so that it is changing over the domain's boundary. The scripts used to get these solutions are in the files *min-surface1.py* and *min-surface2.py*, which can be found in the Appendix or in the Github repository. The initial conditions are:

$$u_0 = 0.7\left(1 - \left(\frac{x}{b}\right)^2 - \left(\frac{y}{a}\right)^2\right) \qquad \text{for the rectangle domain,}$$

$$u_0 = 0.7\left(2x^2 - 2xy\right) \qquad \text{for the ellipse domain,}$$

with only Dirichlet boundary conditions specified. The constants a, b denote the width and height of the rectangle.

Figure 1 shows the solution surfaces to a non-evolving minimal surface equation. On the left side of the figure we solved the equation on the rectangular domain with fixed boundaries. The solution surface is located bellow the initial guess, which is plotted in transparent color. Finally, bellow them there is another minimal surface solution that corresponds to fixing the boundary conditions only at two sides of the rectangle. As expected, this solution is close to zero everywhere.

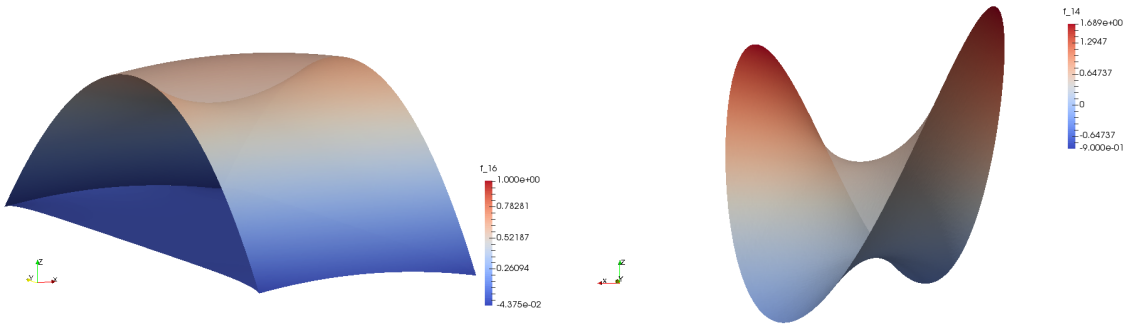

Figure 1: Left: Minimal surfaces on the rectangle domain. Right: Minimal surface the unit circle.

On the right hand side we have a minimal surface on a circular domain. Again, the solution is plotted in transparent color and is located above the initial guess, which is plotted in full color.

We look at the evolving minimal surface problem for the unit circle domain setup. As mentioned earlier, we use the Newton solver at every time step and we stop our scheme once the difference of the two surface areas at two consecutive times is less than 10^{-5}. The evolving surface solution is a continuous transition between the initial guess and the solution shown in the Figure 1. On the left side of Figure 2 we show three solutions at times $t = 0.15, 0.5, 1.5$ and on the right side we can see how the surface area evolved in time. The surface area starts decreasing very quickly at first and the decrease rate gets slower once we are closer to the stable point.

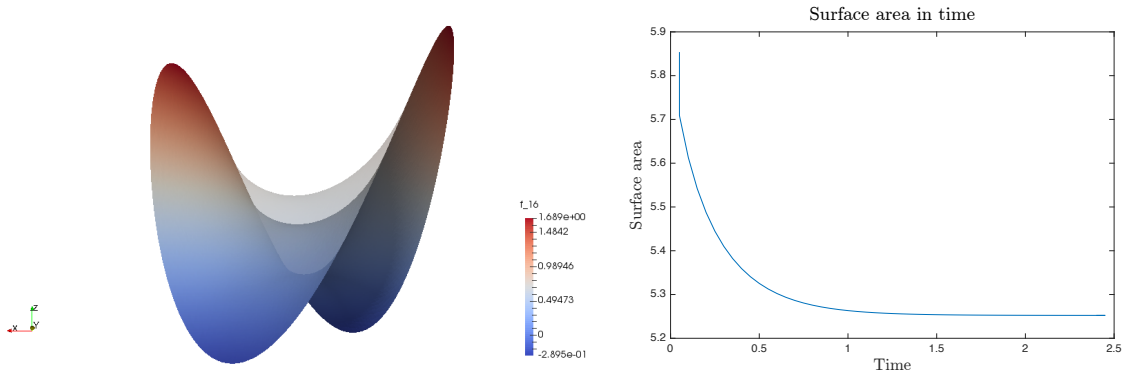

Figure 2: Left: We see continuous transition from u_0 to the same solution as in Figure 1. Right: The surface area is changing in time.

2 Laplace Young equation

In the previous section we solved the minimal surface equation and extended it to make it time evolving. Here we consider the somewhat similar Laplace-Young equation that describes the capillary pressure difference, for example, between some type of fluid and air. This leads to the effect of surface tension and will relate pressure difference to the shape of the surface:

$$\nabla p = -\nabla \cdot \left(\frac{\nabla u}{\sqrt{1 + |\nabla u|^2}} \right), \tag{12}$$

where p is the pressure and u is the deflection that determines the shape of the membrane. For now we will assume that the pressure difference is proportional to the height of the membrane, $\nabla p \sim -u$. After appropriate scaling we get:

$$u = \nabla \cdot \left(\frac{\nabla u}{\sqrt{1 + |\nabla u|^2}} \right). \tag{13}$$

The equation (13) describes a non-evolving static solution for the stable surface. A weak variational form is derived in a similar way as the previously mentioned minimal surface equation:

$$\begin{aligned} u &= \nabla \cdot (q \nabla u) \\ \int_\Omega uv \,\mathrm{d}x &= \int_\Omega \nabla \cdot (q \nabla u)\, v \,\mathrm{d}x \\ \int_\Omega uv \,\mathrm{d}x &= \int_\Omega \nabla \cdot (q \nabla uv) - q \nabla u^T \nabla v \,\mathrm{d}x \\ \int_\Omega uv \,\mathrm{d}x &= \int_{\partial\Omega} q \underbrace{\nabla u^T \vec{n}}_{B} v \,\mathrm{d}S - \int_\Omega q \nabla u^T \nabla v \,\mathrm{d}x. \end{aligned}$$

We implement a solution to this problem in *laplace-young1.py*. Although the solutions look similar to the static minimal surface solutions, they are slightly different. The minimal surface area in the case of the rectangle domain is $A = 1.024$ and for the Laplace-Young equation area is $A = 1.029$.

The high level of abstractness of the FEniCS solver enables us to solve this problem on a different kind of irregular meshes without needing to write a lot of additional code. We generate a mesh on a domain resembling triangle[2] using the package *mshr*. The *mshr* package allows to generate a few basic shapes which can be combined into more complicated domains through manipulation with basic set operations like

[2]So called pizza-triangle

intersection, union and difference. This can be very useful for example for adaptive mesh generation. Based on where the error is located we can generate a new mesh that is more detailed at certain locations. Here we present a short code excerpt from *laplace-young1.py*.

Code listing 1: Pizza-triangle mesh generation.

```
# Triangular pizza mesh (Figure 3)
a = 1.0;
b = 3.0;
triangle = mshr.Polygon([Point(-a, 0.0), Point(a, 0.0), Point
    (0.0, b)])
half_circle = mshr.Ellipse(Point(0.0, 0.0), a, a, xn) - \
                mshr.Rectangle(Point(-a, 0.0), Point(a,a))
domain = triangle + half_circle
mesh = mshr.generate_mesh(domain, xn, "cgal")
u_0 = Expression('''(1 - 0.5*x[0]*x[0]+ 0.5*sin(x[1]*x[1]))''',
    a = a, b = b)
```

As the initial guess u_0 we choose a function periodic in y direction :

$$u_0 = 1 - \frac{x^2}{2} + \frac{\sin y^2}{2}. \tag{14}$$

In Figure 3 we show two surfaces. The left one corresponds to our initial guess described in (14), whose area is $A(u_0) = 6.88$. The other surface is the solution to the Laplace-Young equation with the area $A(u^*) = 5.86$.

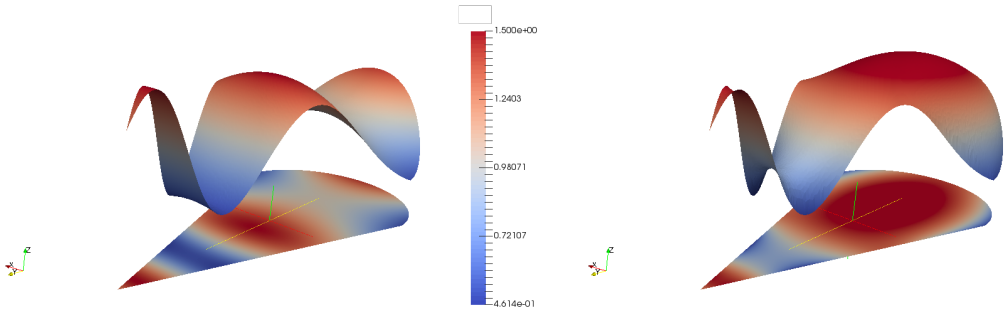

Figure 3: Left: Initial guess u_0 (Surface area: $A(u_0) = 6.88$) . Right: Optimal surface surface on domain resembling a triangle (Surface area: $A(u^*) = 5.86$) .

3 Evolving Laplace-Young equation

We wish to further enforce dynamic behaviour on the Laplace-Young equation. Gauglitz and Radke considered 1D domain and made many approximations in [1], which resulted in the following 1D kinematic equation:

$$\frac{\partial h}{\partial t} = -\frac{1}{1-h}\frac{\partial}{\partial z}\left(-\frac{\partial p}{\partial z}h^3\right), \tag{15}$$

where $p(z)$ represents pressure and $h(z)$ is deflection of the 1D membrane. In our case, we have 2D membrane $u(x)$, which is located on $\Omega \subset \mathbb{R}^2$. Therefore we will rather work with the equation:

$$\frac{\partial u}{\partial t} = -\frac{1}{1-u}\nabla \cdot \left(-(\nabla p)u^3\right). \tag{16}$$

As is the case in [1], we must assume that $0 \leq u < 1$ on Ω. We substitute the pressure from the Laplace-Young equation in (12) to get the final 2D kinematic equation:

$$\frac{\partial u}{\partial t} = -\frac{1}{1-u}\nabla \cdot \left(\frac{u^3 \nabla u}{\sqrt{1+|\nabla u|^2}}\right). \tag{17}$$

The equation in (17) can be solved with additional boundary conditions:

$$u = u_0 \qquad \text{at } t = 0 \tag{18}$$

$$u = u_0 \qquad \text{on } \partial\Omega \tag{19}$$

$$(\nabla u)^T \vec{n} = \sqrt{1+|\nabla u_n|^2}\cos\theta \qquad \text{on } \partial\Omega. \tag{20}$$

Where $\vec{n}$ is an outward vector normal to the domain boundaries and θ is the contact angle between the surface and domain. We derive the weak form for the problem, which is then implemented in FEniCS.

$$\frac{u_n - u_{n-1}}{\Delta t} = -\frac{\nabla \cdot (q_n\, u_n^3 \nabla u_n)}{1-u}$$

$$u_n - u_{n-1} + \frac{\Delta t}{1-u}\,\nabla \cdot \left(q_n\, u_n^3\, \nabla u_n\right) = 0$$

$$\int_\Omega (u_n - u_{n-1})v \,\mathrm{d}x + \Delta t \int_\Omega \nabla \cdot \left(q_n\, u_n^3 \nabla u_n\right)\frac{v}{1-u}\,\mathrm{d}x = 0$$

$$\int_\Omega (u_n - u_{n-1})v \,\mathrm{d}x - \Delta t \int_\Omega \nabla \cdot \left(q_n\, u_n^3 \nabla u_n \frac{v}{1-u}\right) - q_n\, u_n^3 (\nabla u_n)^T \nabla\left(\frac{v}{1-u}\right)\mathrm{d}x = 0$$

$$\int_\Omega (u_n - u_{n-1})v \,\mathrm{d}x - \Delta t \int_{\partial\Omega} q_n\, u_n^3 \frac{v}{1-u_n}(\nabla u_n)^T \vec{n}\,\mathrm{d}S + \Delta t \int_\Omega q_n\, u_n^3 (\nabla u_n)^T \nabla\left(\frac{v}{1-u_n}\right)\mathrm{d}x = 0.$$

Now we substitute the Neumann boundary conditions from (20) and apply the quotient rule for gradients to get the final non-linear variational form:

$$\int_\Omega u_n - u_{n-1} v \, \mathrm{d}x - \Delta t \int_{\partial\Omega} \cos\theta \, u_n^3 v \, \mathrm{d}S + \Delta t \int_\Omega q_n \, u_n^3 \nabla u_n^T \frac{(1-u_n)\nabla v + \nabla u_n v}{(1-u_n)^2} \, \mathrm{d}x = 0.$$

We solve this evolving Laplace-Young equation for different setups. First, we look at the problem resembling a droplet on the unit circle domain. The initial value u_0 is a rescaled half-sphere whose boundary values we fix with the Dirichlet boundary conditions.

$$u_0 = \frac{1}{2}\sqrt{1 - x^2 - y^2 + 0.01}. \tag{21}$$

Note that we have to ensure not only that the initial solution satisfies $0 \leq u_0 < 1$, but also that this inequality is satisfied by all the solutions at all times. We also add 0.01 in the square root in order to be sure that the function is well defined on the unit circle.

In Figure 4 we can see the solution surfaces at three consecutive times alongside a plot depicting the the evolution of the surface area in time. The solution starts from

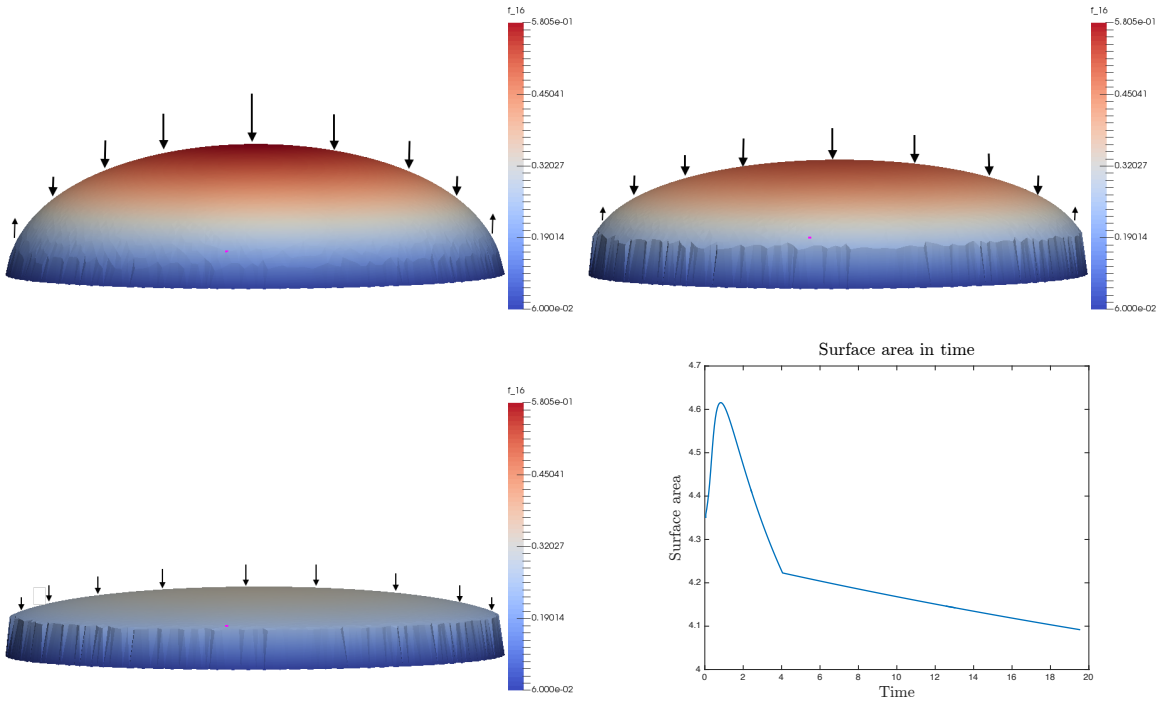

Figure 4: Evolution of a case resembling the surface tension of a droplet. The solution decrease in center is quicker than at the boundaries, where it starts rising until it flattens out. Afterwards, the surface proceeds to uniformly decrease to a zero function.

a scaled half-sphere and starts to flatten out. Remember that the pressure from the Laplace-Young equation is dependent on deflection and local curvature. This causes the points in the centre to start to decrease much more quickly in comparison to the points near the boundary. In fact, the points very close to the boundary start rising until they reach the same level as the central part. Once the surface gets flat in the middle, it starts to decrease uniformly at a similar speed.

If we look at the time evolution of the surface area we can see that the initial elevation of the boundary causes spike of the the surface area around time $t \sim 0.5$. We stop the solution at $t = 20$, but if we continued we would see a uniform decrease to the zero function which corresponds to the surface area: $A(0) = |\Omega| = \pi$.

We are also interested in a more complicated Ω such as the previously-constructed domain resembling a triangle. We have to change the initial conditions from the previously chosen in (14), due to the $0 \leq u < 1$ restriction. The initial guess is constructed in a way that it is periodic in y direction and we only specify the Dirichlet boundary conditions.

$$u_0 = \frac{1}{2}\left(1 - \frac{x^2}{2} + \frac{\sin y^2}{2}\right) \tag{22}$$

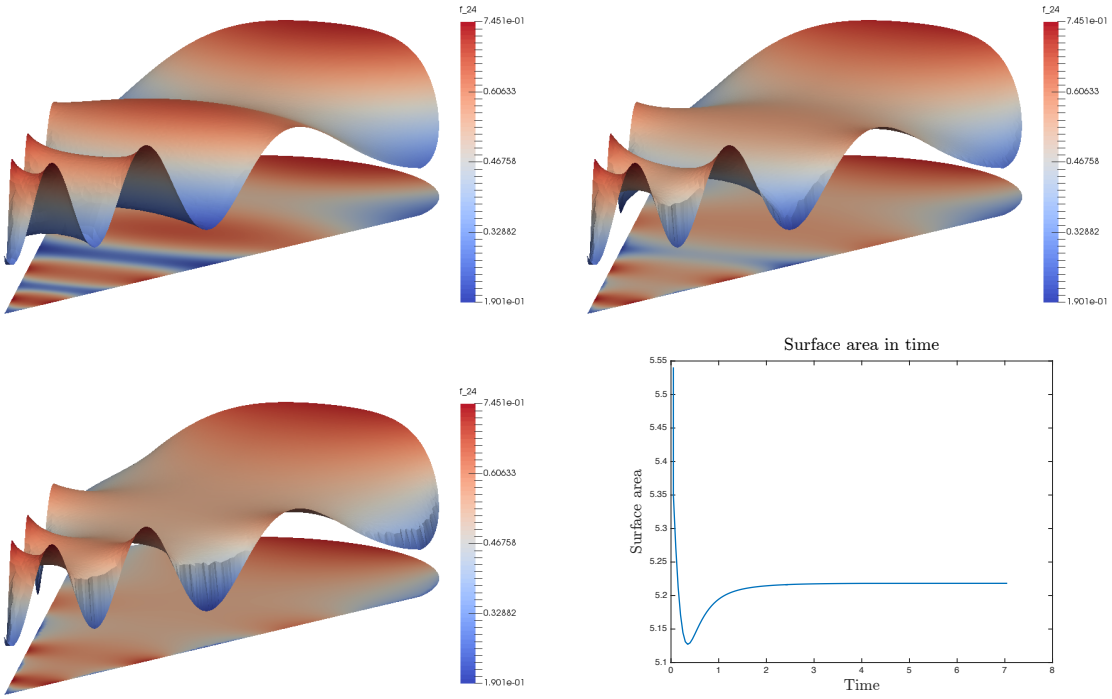

Figure 5: Evolution of surface tension with Dirichlet boundary conditions. The points near the boundary try to break from the values fixed by the Dirichlet conditions, which causes an increase of the surface area right after the initial drop.

As you can see in Figure 5, the solution surface is flattening out in time, which makes the points near the boundary to increase as the solver is trying to break from them. This is reflected in the increase of the total surface area right after the initial drop.

Next we look at the problem without the Dirichlet boundary conditions but with the Neumann conditions. We start from a similar surface as in (22) with different scaling and higher periodicity in y direction. The contact angle of the surface is chosen to be $\theta = \pi/2$ and $\theta = 3\pi/4$.

$$u_0 = \frac{1}{3}\left(1 - \frac{x^2}{2} + \frac{\sin 2x^2}{2}\right) \tag{23}$$

In Figure 6 we can see the solutions for both contact angles θ. The solution for $\theta = \pi/2$, which is located above the solution for $\theta = 3\pi/4$, flattens until it converges to a zero function as expected. The solution corresponding to $\theta = 3\pi/4$ is also flattening. However, the final stable point has the contact angles of $\theta = 3\pi/4$. This causes the second solution to converge to a surface with a larger area.

We also consider starting from a wrinkled surface on a unit circle. The initial

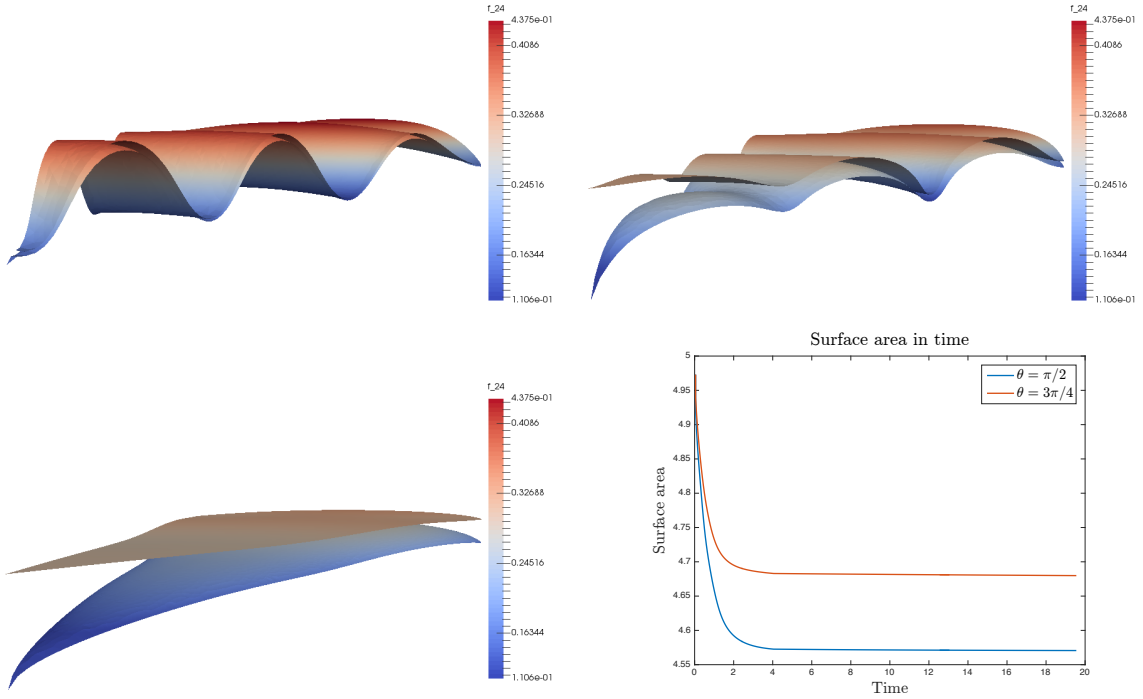

Figure 6: The solution on a triangle-like domain for two different Neumann conditions $\theta = \pi/2$ and $\theta = 3\pi/4$. Solution for $\theta = \pi/2$ is above the solution for $\theta = 3\pi/4$ and converges to a surface with a smaller area.

guess is:

$$u_0 = \frac{1}{2}\left(1 - \frac{1}{2}\sin\left(2x^2 + 10y^2\right)\right). \tag{24}$$

We can see in Figure 7 that the surface straightens out. There is also a similar effect present as before with the surface increasing rapidly near the Dirichlet boundary conditions. Figure 7 shows that there is a slight increase of the surface area after its initial drop.

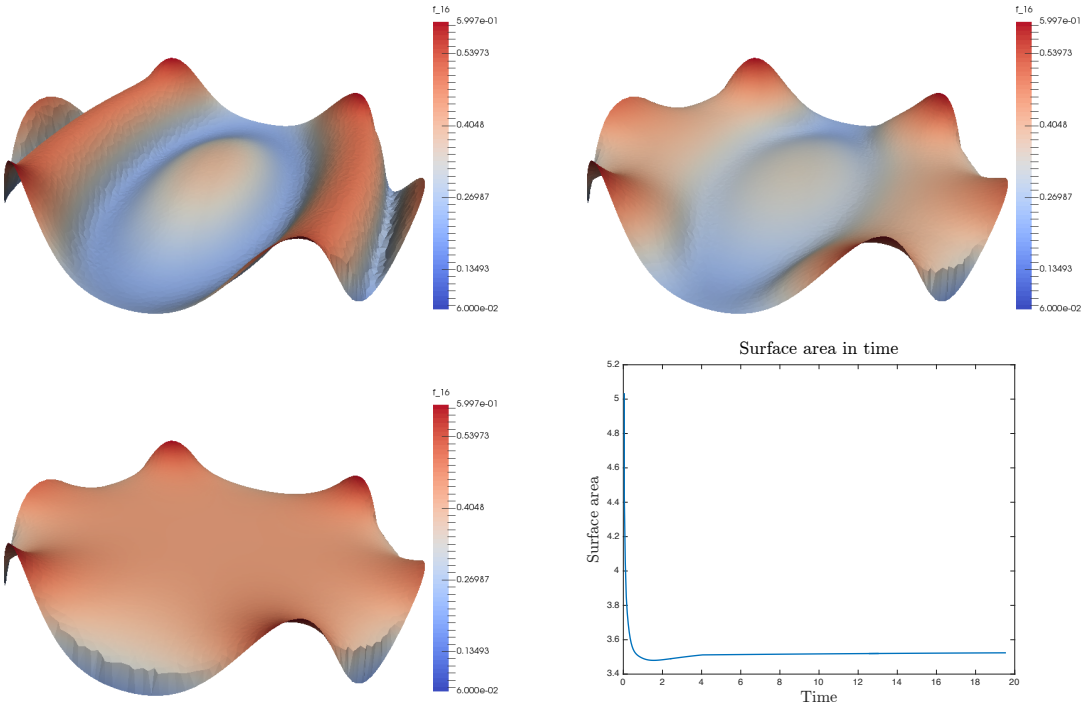

Figure 7: The solution in the unit circle starting from a wrinkled disc as described in (24).

Conclusion

In this work we solved and derived various PDEs describing the shapes of minimal surfaces and shapes that are the results of surface tension. We started from the basic minimal surface PDE corresponding to the Euler-Lagrange equation that was derived using the calculus of variations as shown in [3]. Following this, we were interested in enforcing a certain dynamic on the surface such that it evolve in time. This was done by adding a time derivative on the right hand side.

In order to model surface tension we solved the Laplace-Young equation on 2D domain, where we assumed that the pressure difference ∇p is proportional to the height of the surface. This led to solutions that were close to the previously considered minimal surface solutions. We solved this equation on a non-trivial mesh on a domain resembling a triangle[3]. To be able to model surface tension in time we considered a similar equation as in [1]. However, we formulated it for a 2D domain. This equation was than solved for various problem setups including the one that resembled surface tension of a circular droplet.

There is much further work that could be undertaken on the basis of our findings. Concretely we could enforce both, the Dirichlet and Neumann boundary conditions, which would prevent a rapid change near the fixed boundaries, which most likely caused the increase of the surface area at certain times. This would also allow for a problem formulation with time-evolving contact angles that would produce much more interesting solutions. Moreover, we could formulate the problem with an additional forcing function F_Ω at a certain subset of Ω by substituting $u' := u + F_\Omega$.

The fact that these problems can be implemented in FEniCS using only a few lines of code proves how powerful a tool it is. Nevertheless, there were many setups for which our Newton solver did not converge. These issues can be sometimes solved by continuation methods or by using different solvers. FEniCS is a very convenient framework, but one should always be careful when solving non-linear problems as there is no such thing as a universal method.

References

[1] P. A. Gauglitz and C. J. Radke. An extended evolution equation for liquid film breakup in cylindrical capillaries. *Chemical Engineering Science*, 43(7):1457–1465, 1988.

[3] So called pizza-triangle domain.

[2] A. Logg, K.-A. Mardal, and G. Wells. *Automated Solution of Differential Equations by the Finite Element Method*. 2011.

[3] E. Süli. Minimal surface equation. *The PDE Coffee Table Book*, 2001.

Appendix

Here we present the scripts used to find the solutions to the minimal surface and the Laplace-Young equations. All of the code with the solutions can be electronically accessed in the Github repository: `https://github.com/can1003876/python`.

Code listing 2: Static minimal surface.

```
"""
Candidate: 1003876

Static minimal surface equation:
-div( q grad u) = 0,

where q = q(grad u) = (1 + |grad u|^2)^{-1/2}

using Newton solver
"""
from dolfin import *
import mshr
import math
import numpy as np


# Solver settings:
solv_parameters = {"newton_solver": {"relative_tolerance": 1e
    -5,
                                     "report": True,
                                     "maximum_iterations": 20}}
compiler_parameters =  {"optimize": True,
                        "cpp_optimize": True,
                        "cpp_optimize_flags": "-O3 -ffast-math
                           -march=native"}
output = File("solutions/min-surface1/sol1.pvd")


```

```
# Mesh properties (for ellipse and rectangle)
a = 1.0;
b = 1.0;
xn = 100
yn = 100

# Rectangle (figure 1)
u_0 = Expression('''0.7*(1 - x[0]*x[0]/(b*b) - x[1]*x[1]/(a*a)
    )''', a = a, b = b)
mesh = RectangleMesh(Point(-a/2,-b/2), Point(a/2, b/2), xn, yn)

# Ellipse (figure 2)
# domain = mshr.Ellipse(Point(0.0, 0.0), a, b, xn)
# mesh = mshr.generate_mesh(domain, 50, "cgal")
# u_0 = Expression('''0.7*(x[0]*x[0]/(b*b) - x[1]*x[1]/(a*a) +
    (x[0]-x[1])*(x[0]-x[1]) )''', a = a, b = b)

V = FunctionSpace(mesh, 'Lagrange', 2)

# Solution is u0
u0 = interpolate(u_0, V);
v = TestFunction(V)
q0 = (1+inner(grad(u0),grad(u0)))**(-.5)

# Save initial guess
output << u0

S1 = assemble(sqrt(1+inner(grad(u0), grad(u0)))*dx)
print 'Initial surface:', S1

# Boundary conditions
bc = DirichletBC(V, u0, "on_boundary");
theta = Constant(pi/2)
B = cos(theta)/q0

# Weak form formulation
F = q0*B*v*ds - q0*inner(grad(u0),grad(v))*dx
solve(F == 0, u0, bc,
            solver_parameters = solv_parameters,
            form_compiler_parameters = compiler_parameters)
output << u0
print 'Surface area 2:', assemble(sqrt(1+inner(grad(u0),grad(u0
    )))*dx)
```

```
# Now fix only x[1] boundaries
# bc = DirichletBC(V, u0, "on_boundary && (near(x[1], -1.0) ||
    near(x[1], 1.0))" );
# solve(F == 0, u0, bc,
#               solver_parameters = solv_parameters,
#               form_compiler_parameters = compiler_parameters)
# output << u0
# print 'Surface area 3:', assemble(sqrt(1+inner(grad(u0),grad(
    u0)))*dx)
```

Code listing 3: Evolving minimal surface.

```
"""
Candidate: 1003876

Evolving minimal surface equation:
-div( q grad u) = u_t,

where q = q(grad u) = (1 + |grad u|^2)^{-1/2}

using Newton solver
"""

from dolfin import *
import mshr
import math
import numpy as np

# Solver settings:
solv_parameters = {"newton_solver": {"relative_tolerance": 1e
    -5,
                                     "report": True,
                                     "maximum_iterations": 10}}
compiler_parameters =  {"optimize": True,
                        "cpp_optimize": True,
                        "cpp_optimize_flags": "-O3 -ffast-math
                            -march=native"}
output = File("solutions/min-surface2/sol1.pvd")

```

```
# Mesh properties (for ellipse and rectangle)
a = 1.0;
b = 1.0;
xn = 100
yn = 100

# Ellipse (figure 2)
domain = mshr.Ellipse(Point(0.0, 0.0), a, b, xn)
mesh = mshr.generate_mesh(domain, 50, "cgal")
u_0 = Expression('''0.7*(x[0]*x[0]/(b*b) - x[1]*x[1]/(a*a) + (x
    [0]-x[1])*(x[0]-x[1]) )''', a = a, b = b)

V = FunctionSpace(mesh, 'CG', 2)
u0 = interpolate(u_0, V);
u1 = Function(V)
v = TestFunction(V)

S1 = assemble(sqrt(1+inner(grad(u0),grad(u0)))*dx)
print 'Initial surface:', S1

# Change in consecutive times
dt = Constant(0.05);
t = float(dt); T = 1
# Tolerance of change of surface area
toldS = 1.e-5
# Variable to store change in surface area
dS = toldS          # to go into first for-loop

q1 = (1+inner(grad(u1),grad(u1)))**(-.5)

# Boundary settings
bc = DirichletBC(V, u0, "on_boundary");
theta = Constant(pi/2)
B = cos(theta)/q1
F = (u1-u0)*v*dx - dt*q1*B*v*ds + dt*q1*inner(grad(u1),grad(v))
   *dx

areas = np.array([S1])
times = np.array([t])
while dS >= toldS:
    solve(F == 0, u1, bc,
                    solver_parameters = solv_parameters,
```

```
                    form_compiler_parameters =
                        compiler_parameters)
    # Look at change of surface
    S0 = S1
    S1 = assemble(sqrt(1+inner(grad(u1),grad(u1)))*dx)
    V1 = assemble(u1*dx)
    dS = abs(S1 - S0)
    # Print report
    print 'At time:', t, ' surface area is :',S1
    print 'Volume is :', V1
    areas = np.append(areas, [S1])
    times = np.append(times, [t])
    # Save this time solution
    output << u1
    # Prepare next iteration
    u0.assign(u1)
    t += float(dt)

print 'surface area:', assemble(sqrt(1+inner(grad(u0),grad(u0))
    )*dx)

np.savetxt("solutions/min-surface2/areas.csv",areas)
np.savetxt("solutions/min-surface2/times.csv",times)
```

Code listing 4: Static Laplace-Young.

```
"""
Candidate: 1003876

Static Laplace-Young equation:
-div( q grad u) = u,

where q = q(grad u) = (1 + |grad u|^2)^{-1/2}

using Newton solver
"""
from dolfin import *
import mshr
import math
import numpy as np
```

```
15
# Solver settings:
solv_parameters = {"newton_solver": {"relative_tolerance": 1e
    -5,
                                     "report": True,
20                                   "maximum_iterations": 20}}
compiler_parameters =  {"optimize": True,
                        "cpp_optimize": True,
                        "cpp_optimize_flags": "-O3 -ffast-math
                           -march=native"}
output = File("solutions/laplace-young1/sol1.pvd")
25

# Mesh properties (for ellipse and rectangle)
a = 1.0;
b = 1.0;
30 xn = 100
yn = 100

# Rectangle
#mesh = RectangleMesh(Point(-a/2,-b/2), Point(a/2, b/2), xn, yn
    )
35 #u_0 = Expression('''0.7*(1 - x[0]*x[0]/(b*b) - x[1]*x[1]/(a*a)
     )''', a = a, b = b)

# Ellipse
# domain = mshr.Ellipse(Point(0.0, 0.0), a, b, xn)
# mesh = mshr.generate_mesh(domain, 50, "cgal")
40 # u_0 = Expression('''0.7*(x[0]*x[0]/(b*b) - x[1]*x[1]/(a*a) +
    (x[0]-x[1])*(x[0]-x[1]) )''', a = a, b = b)

# Triangular pizza mesh (Figure 3)
a = 1.0;
b = 3.0;
45 triangle = mshr.Polygon([Point(-a, 0.0), Point(a, 0.0), Point
    (0.0, b)])
half_circle = mshr.Ellipse(Point(0.0, 0.0), a, a, xn) - \
                  mshr.Rectangle(Point(-a, 0.0), Point(a,a))
domain = triangle + half_circle
mesh = mshr.generate_mesh(domain, xn, "cgal")
50 u_0 = Expression('''(1 - 0.5*x[0]*x[0]+ 0.5*sin(x[1]*x[1]))''',
     a = a, b = b)
```

```
V = FunctionSpace(mesh, 'Lagrange', 2)

boundaries = MeshFunction('size_t',mesh, 1)
ds = Measure('ds')[boundaries]

# Solution is u0
u0 = interpolate(u_0, V);
u = TrialFunction(V)
v = TestFunction(V)
q0 = (1+inner(grad(u0),grad(u0)))**(-.5)

# Save initial guess
output << u0

S1 = assemble(sqrt(1+inner(grad(u0), grad(u0)))*dx)
print 'Initial surface:', S1

# Boundary conditions
bc = DirichletBC(V, u0, "on_boundary");
theta = Constant(pi/2)
B = cos(theta)

# Weak form formulation
F =  B*v*ds - q0*inner(grad(u0), grad(v))*dx + u0*v*dx
solve(F == 0, u0, bc,
            solver_parameters = solv_parameters,
            form_compiler_parameters = compiler_parameters)

print 'Surface area 2:', assemble(sqrt(1+inner(grad(u0),grad(u0
    )))*dx)
output << u0
```

Code listing 5: Evolving Laplace-Young.

```
"""
Candidate: 1003876

Evolving Laplace-Young surface tenstion equation:
```

```
-div( q u^3 grad u)/(1-u) = u_t,

where q = q(grad u) = (1 + |grad u|^2)^{-1/2}

using Newton solver
"""

from dolfin import *
import mshr
import math
import numpy as np

# Solver settings:
solv_parameters = {"newton_solver": {"relative_tolerance": 1e
    -5,
                                     "report": True,
                                     "maximum_iterations": 10}}
compiler_parameters =  {"optimize": True,
                        "cpp_optimize": True,
                        "cpp_optimize_flags": "-O3 -ffast-math
                            -march=native"}
output = File("solutions/laplace-young2/sol5.pvd")

# Mesh properties (for ellipse and rectangle)
a = 1.0;
b = 1.0;
xn = 100
yn = 100

# Ellipse (figure 1)
domain = mshr.Ellipse(Point(0.0, 0.0), a, b, xn)
mesh = mshr.generate_mesh(domain, 50, "cgal")
# Droplet
#u_0 = Expression('''0.5*sqrt(1 - x[0]*x[0]/(a*a) - x[1]*x[1]/(
    b*b) + 0.01 )''', a = a, b = b)
# Wrinkled disc
u_0 = Expression('''0.5*(1 - 0.5*sin(2*x[0]*x[0]/(b*b)
                        + 10*x[1]*x[1]/(a*a)) )''', a = a, b =
                            b)

# Triangular pizza mesh (Figure 2/3/4)
# a = 1.0;
# b = 3.0;
```

```
# triangle = mshr.Polygon([Point(-a, 0.0), Point(a, 0.0), Point
    (0.0, b)])
# half_circle = mshr.Ellipse(Point(0.0, 0.0), a, a, xn) - \
#                  mshr.Rectangle(Point(-a, 0.0), Point(a,a))
# domain = triangle + half_circle
# mesh = mshr.generate_mesh(domain, xn, "cgal")
# u_0 = Expression('''0.3*(1 - 0.5*x[0]*x[0]
#                     + 0.5*sin(2*x[1]*x[1]))''', a = a, b = b)

V = FunctionSpace(mesh, 'CG', 2)
u0 = interpolate(u_0, V);
u1 = Function(V)
v = TestFunction(V)

S1 = assemble(sqrt(1+inner(grad(u0),grad(u0)))*dx)
V1 = assemble(u0*dx)
print 'Initial surface:', S1

# Change in consecutive times
dt = Constant(0.05);
t = float(dt); T = 20
# Tolerance of change of surface area
toldS = 1.e-5
# Variable to store change in surface area
dS = toldS          # to go into first for-loop

q1 = (1+inner(grad(u1),grad(u1)))**(-.5)

# Boundary settings
bc = DirichletBC(V, u0, "on_boundary");
theta = Constant(pi/4)
grad2 = ((1-u1)*grad(v) + grad(u1)*v)/(1-u1)**2
F = (u1-u0)*v*dx - dt*cos(theta)*(u1**3)*v*ds + dt*q1*(u1**3)*
   inner(grad(u1),grad2)*dx

areas = np.array([S1])
volumes = np.array([S1])
times = np.array([t])
while dS >= toldS and t < T:
    solve(F == 0, u1, bc,
                  solver_parameters = solv_parameters,
                  form_compiler_parameters =
                     compiler_parameters)
```

```
    # Look at change of surface
    S0 = S1
    S1 = assemble(sqrt(1+inner(grad(u1),grad(u1)))*dx)
    V1 = assemble(u1*dx)
    dS = abs(S1 - S0)
    # Print report
    print 'At time:', t, ' surface area is :',S1
    print 'Volume is :', V1
    areas = np.append(areas, [S1])
    times = np.append(times, [t])
    volumes = np.append(volumes, [V1])
    # Save this time solution
    output << u1
    # Prepare next iteration
    u0.assign(u1)
    if (t > 4):
        dt = Constant(0.5)
    t += float(dt)

print 'surface area:', assemble(sqrt(1+inner(grad(u0),grad(u0))
    )*dx)

np.savetxt("solutions/laplace-young2/areas5.csv",areas)
np.savetxt("solutions/laplace-young2/volumes5.csv",volumes)
np.savetxt("solutions/laplace-young2/times5.csv",times)
```