Solid drops – Supplementary material

S. Mora, C. Maurini, T. Phou, J.-M. Fromental, B. Audoly, Y. Pomeau

August 20, 2013

1 Numerical method

Minimizing the total energy of a capillary solid with non-trivial shape requires the use of numerical computations. For a solid occupying a domain Ω of arbitrary shape in its the reference configuration, we use the finite element method to predict its equilibrium configuration under the action of the capillary forces. In the numerical work, we model the body as an almost incompressible hyperelastic body with a surface energy proportional to the total area of its boundary in the current configuration and we accept that the body can undergo large displacement and deformations, by using a fully non-linear kinematical theory. We assume a Neo-Hookean elastic energy density in the form [2]

$$w(u) = \frac{\mu}{2}(I-3) - \mu \ln(J) + \frac{\lambda}{2}(\ln J)^2,$$

where $I = \operatorname{tr} \mathbf{C}$ and $J = \det \mathbf{C}$ with $\mathbf{C} = \mathbf{F}^T\mathbf{F}$. In the almost incompressible case where $\lambda \gg \mu$, the finite element method should be applied with caution because of the bad-conditioning and the kinematical locking induced by the finite element discretisation [3, 5]. We adopt here a classical solution to bypass these issues consisting in the use of a mixed formulation, where a pressure-like variable

$$p = -\lambda \ln(J)$$

appears as independent variable together with the displacement field u. Imposing the expression for the p as a function of J as a constraint through the Lagrange multiplier Λ, the energy density may be re-written as

$$w(u,p,\Lambda) = \frac{\mu}{2}(I-3) - \mu \ln(J) + \frac{\lambda}{2}p^2 + \Lambda(p + \lambda \ln(J))$$

The stationarity condition of the energy with respect to p implies that $\Lambda = -p/\lambda$. Hence the energy density may be rewritten in the final form

$$w(u,p) = \frac{\mu}{2}(I-3) - (\mu + p)\ln(J) - \frac{\lambda}{2}p^2.$$

which is a $u-p$ mixed form frequently used for almost incompressible hyperelastic solids [1]. Hence, our finite element formulation is based on the research of the stationary points of the mixed total energy functional

$$\mathcal{E}(u,p) = \underbrace{\int_\Omega w(u,p)dx}_{\text{Bulk energy}} + \underbrace{\gamma \int_{\partial\Omega} \left\| J\,\mathbf{F}^{-T}\mathbf{N} \right\| dS}_{\text{Surface energy}} \tag{1}$$

where the term $\left\| J\,\mathbf{F}^{-T}\mathbf{N} \right\| dS$ gives the element of area of the deformed configuration as a function of the quantities defined on the reference domain Ω, $\mathbf{N}$ being the unit normal to the reference boundary $\partial\Omega$.

An ad-hoc numerical code is developed for the purpose using the `FEniCS` finite element library [4]. The displacement vector u and the pressure-like variable p are discretized using Lagrange finite elements, with a quadratic interpolation for u and a linear interpolation for p. The resulting finite element formulation is very similar to the Taylor-Hood family of elements used for solving almost incompressible fluid flows. The nonlinear problem in $u-p$ is solved using a Newton algorithm based on a direct parallel solver (`MUMPS`). For a given geometry, quasi-static simulations are performed by setting $\mu = 1$, $\lambda = 1000$ and slowly increasing the surface tension γ up to the desired dimensionless value. The 3D results are obtained using tetrahedral meshes suitably refined in the regions close to edges and vertices of the boundary, where the deformations induced by capillarity are expected to be higher. In a 3D computation the typical number of total degrees of freedom varies between 1 to 10 millions. Parallel computations on high performance computing architectures are necessary to resolve the full non-linear problem in a reasonable time. The high dimensionality is a consequence of the use of quadratic finite elements for the displacement interpolation and the required level of mesh refinement in 3D.

2 Numerical simulations

Numerical simulations are performed on elongated 3D cylinders with different cross-sections. In each case, we perform the simulations in dimensionless units by setting the shear modulus μ to 1, the height of the cross section to 1, and varying the surface tension $\bar{\gamma}$. The lame parameter λ is fixed to 1000 to approach the incompressible limit. We tested that the value of λ has no influence on the results provided that it is sufficiently high.

Figure 1 reports the case of the square cross-section. On the right-end side we block only the axial (Z) component of the displacement, leaving free the displacement components in the plane of the cross-section. This is equivalent to consider a free-free cylinder of twice the length used in the simulation subjected to a symmetry condition with respect to the plane $Z = 0$. To get the quantitative results on the shape-change of the cross-section reported in the paper, we consider the right-end cross-section in the figure, whose shape is not perturbed by boundary effects. The mesh is refined in correspondence of geometrical discontinuities, which are expected to induce large capillary forces and then large deformations. Mesh refinement tests are performed to assure the convergence of global quantities as beam shortening and global cross-sectional shape change measures.

Figure 2 shows the bending deformations obtained for triangular cross-sections. In this case we set to 0 the three components of the displacement vector in order to imitate the clamping condition imposed in experiments. The symmetry conditions on the cross-section are not exploited in order to test the possibilities of symmetry breaking induced by bifurcations in the non-linear regime. The simulation is run on the full mesh reported in figure 2. Mesh refinement tests are performed in order to assure the convergence of the result for the tip displacement of the beam.

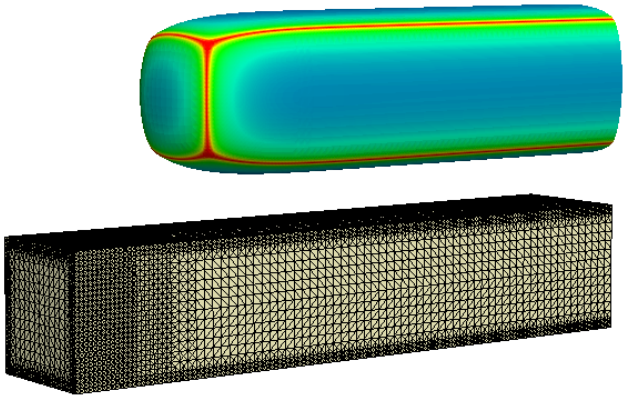

Figure 1: Capillary deformation of a cylinder with square cross-section of unit height for $\gamma = 0.5$, $\mu = 1$, and $\lambda = 1000$. The colors represent the pressure variable $p = -\lambda \ln J$ in a scale from 0 (blue) to 5 (red).

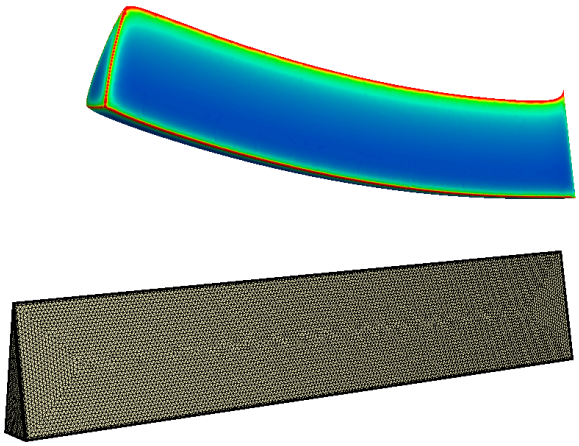

Figure 2: Capillary deformation of a cylinder with triangular cross-section of unit height for surface tension $\gamma = 0.084$, $\mu = 1$, and $\lambda = 1000$. The colors represent the pressure variable $p = -\lambda \ln J$ in a scale from 0 (blue) to 5 (red).

3 A minimal `FEniCs` script to solve the problem

We report below a minimal python code that can be used to perform a simulation. The code is based on the use of the finite element library `FEniCs`. FEniCS is a free software that may be downloaded at `http://fenicsproject.org/download/`. To Run the program:

1. Install `FEniCs` and test that the example of FEniCS are working. The code below is tested with `FEniCs` version 1.2.0.

2. Run this file by typing at the command line `python CapillarySolid.py` (in a terminal with FEniCS enabled).

To get a full list of command-line options type: `python CapyllaryBeam.py --help`

```
# Copyright (C) 2011-2013 Corrado Maurini, corrado.maurini@upmc.fr
# Licensed under the GNU LGPL Version 3.
#
# This file is free software: you can redistribute it and/or modify
# it under the terms of the GNU Lesser General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This file is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU Lesser General Public License for more details.
#
# A copy of the GNU Lesser General Public License is available at
#  <http://www.gnu.org/licenses/>.
```

```
#
# This software is based on the use of FEniCS version 1.2
#
# First added:  2011-05-31
# Last changed: 2013-08-20
#
# Created by Corrado Maurini, 2013
# For any further information contact corrado.maurini@upmc.fr
#
# ---------------------------------
# Description:
# ---------------------------------
# This demo program solves a problem for an almost incompressible
# hyperelastic solid with surface tension.
# The code below considers a 3D cantilever beam with rectangular cross-section.
# The loading is given only by the surface tension
# (acting on all the surfaces of the boundary, except the clamped one)
# To change boundary conditions and loading, refer to the FEniCS manual:
# http://fenicsproject.org/documentation/
# and the FEniCS book available for free at:
# https://launchpadlibrarian.net/83776282/fenics-book-2011-10-27-final.pdf
#
# ---------------------------------
# Technical details:
# ---------------------------------
# To solve the problem for the almost incompressible hyperelastic solid we use
# a mixed formulation including both the displacement and the pressure as state
# variables. We solve a series of problems for increasing surface tension.
# For each value of the surface tension, we solve a non-linear problem using a
# Newton algorithm. The linear solver used in the Newton iterations is a DIRECT solver
# like umfpack (for serial runs) or mumps (for parallel runs). An iterative solver would be prone to
# convergence issue because the problem is ill-conditioned.
#
# ---------------------------------
# To Run the program:
# ---------------------------------
# 1) Install FEniCS version 1.2 and test that the example of FEniCS are working.
#    FEniCS is a free software that may be dowloaded at http://fenicsproject.org/download/
#
# 2) Run this file by typing  at the command line (in a terminal with FEniCS enabled):
#       python CapillarySolid.py
#
# To get a full list of command-line options type:
#       python CapyllaryBeam.py --help
#
# Some examples to set command-line arguments
#       python CapillaryBeam.py --user.mesh_ref 15 --user.fe_order_u 2 --user.plot False --gamma_max 0.4 --gamma_nsteps 12
#       mpiexec -n 8 python CapillaryBeam.py --user.mesh_ref 30 --user.fe_order_u 1 --user.solver.newton_solver.relaxation_parameter 1 --solve
#
# To visualize the results open the .pvd (or .xdmf) files in the directory "results" using Paraview (downloadable for free at http://www.parav
#
# ---------------------------------
# Further support:
# ---------------------------------
# For any question or bug report, please e-mail to corrado.maurini@upmc.fr
#
""" This demo program solves a hyperelastic problem with surface energy to model the large deformations of a capillary solids
author: Corrado Maurini (corrado.maurini@upmc.fr)
date of the first version: 08/2011
last modified 08/2013
"""
# Load the required modules
from dolfin import *
import numpy as np
import os

#-------------------------------------------------------------
#           Parameters
#-------------------------------------------------------------
```

```
85
86  set_log_level(INFO)
87
88  # Optimization options for the form compiler
89  #parameters["num_threads"] = 1
90  parameters["mesh_partitioner"] = "SCOTCH"
91  parameters["form_compiler"]["quadrature_degree"] = 2
92  parameters["form_compiler"]["cpp_optimize"] = True
93  parameters["form_compiler"]["optimize"] = True
94  parameters["form_compiler"]["log_level"] = INFO
95  parameters["allow_extrapolation"] = True
96  ffc_options = {"optimize": True, \
97                 "eliminate_zeros": True, \
98                 "precompute_basis_const": True, \
99                 "precompute_ip_const": True, \
100                "quadrature_degree": 2}
101
102 # Some user parameters
103 user_par = Parameters("user")
104 user_par.add("bounds_xmin",-0.5)
105 user_par.add("bounds_xmax",0.5)
106 user_par.add("bounds_ymin",-0.5)
107 user_par.add("bounds_ymax",0.5)
108 user_par.add("bounds_zmin",0.)
109 user_par.add("bounds_zmax", 2.5)
110 user_par.add("fe_order_u",1)
111 user_par.add("fe_order_p",1)
112 user_par.add("gamma_min",0.)
113 user_par.add("gamma_max",.2)
114 user_par.add("gamma_nsteps",10)
115 user_par.add("mesh_ref",10)
116 user_par.add("save_dir","results")
117 user_par.add("output_type","pvd")
118 user_par.add("plot",True)
119
120
121 # Non linear solver parameters
122 solver_par = NonlinearVariationalSolver.default_parameters()
123 solver_par.rename("solver")
124 solver_par["symmetric"]=True
125 solver_par["linear_solver"]="umfpack"# use "mumps" in parallel
126 solver_par["lu_solver"]["same_nonzero_pattern"] = True
127 solver_par["lu_solver"]["verbose"] = True
128 solver_par["newton_solver"]["maximum_iterations"] = 20
129 solver_par["newton_solver"]["relaxation_parameter"] = .8
130 solver_par["newton_solver"]["relative_tolerance"] = 1e-5
131 solver_par["newton_solver"]["absolute_tolerance"] = 1e-5
132 # add user parameters in the global parameter set
133 parameters.add(user_par)
134 parameters.add(solver_par)
135
136 # Parse parameters from command line
137 parameters.parse()
138 info(parameters,True)
139 user_par = parameters.user
140
141 #-----------------------------------------------------------
142 #           Geometry
143 #-----------------------------------------------------------
144
145 # Create the geometry and the mesh
146 xmin,xmax = user_par.bounds_xmin,user_par.bounds_xmax
147 ymin,ymax = user_par.bounds_ymin,user_par.bounds_ymax
148 zmin,zmax = user_par.bounds_zmin,user_par.bounds_zmax
149 geom = Box(xmin,ymin,zmin,xmax,ymax,zmax)
150 mesh = Mesh(geom,user_par.mesh_ref)
151
152 #-----------------------------------------------------------
153 #           Definition of function spaces
```

```
#-------------------------------------------------------------

# Create function space
P2 = VectorFunctionSpace(mesh, "CG", user_par.fe_order_u) # Space for displacement
P1 = FunctionSpace(mesh, "CG", user_par.fe_order_p)       # Space for pressure
V = MixedFunctionSpace([P1,P2])
V_u = V.sub(1)
V_p = V.sub(0)
ndim = P2.cell().d

# Create functions to define the energy and store the results
up = Function(V)
(p,u)=split(up)

# Create test and trial functions for the variational formulation
dup = TrialFunction(V)
vq  = TestFunction(V)
(q,v) = TestFunctions(V)

#-------------------------------------------------------------
#          Boundary conditions
#-------------------------------------------------------------
# Mark boundary subdomains
xtol = mesh.hmin()/4.
class ClampedBoundary(SubDomain):
    def inside(self, x, on_boundary):
        return x[2]-zmin<xtol and on_boundary

# Define the boundary conditions
zero_vector = Constant((0.0,0.0,0.0))
bc1 = DirichletBC(V_u, zero_vector, ClampedBoundary())
bc_u = [bc1]

#-------------------------------------------------------------
#          Define boundaries with surface tension
#-------------------------------------------------------------
# Define the part on the boundary where surface tension should be applied
class SurfaceBoundary(SubDomain):
    def inside(self, x, on_boundary):
        return on_boundary

# Mark facets where apply surface tension with 1
boundary_parts = FacetFunction("size_t", mesh, 0)
surface_boundary = SurfaceBoundary()
surface_boundary.mark(boundary_parts, 1)

# Redefine element of area to include informations about surface tension
ds = ds[boundary_parts]

#-------------------------------------------------------------
#                   Kinematics
#-------------------------------------------------------------
I = Identity(ndim)          # Identity tensor
F = I + grad(u)             # Deformation gradient
C = transpose(F)*F          # Right Cauchy-Green tensor
E = 0.5*( C - I )           # Green-Lagrange tensor

# Invariants of deformation tensors
Ic = tr(C)
J  = det(F)

# Normal and tangent vectors in the reference configuration
N = FacetNormal(mesh)
# Element of area transformation operator
NansonOp = transpose(cofac(F))
# surface element vector in the deformed configuration
deformed_N = dot(NansonOp,N)
# norm of the surface element vector in the current configuration
current_element_of_area = sqrt(dot(deformed_N,deformed_N))
```

```
223
224 #-------------------------------------------------------------
225 #             Energy and variational formulation
226 #-------------------------------------------------------------
227
228 # Lame's parameters
229 mu, lmbda = Constant(1.), Constant(1000.)
230
231 # Bulk energy  (strain energy for an almost incompressible neo-Hookean model)
232 bulk_energy_density = mu*(Ic - ndim) -(mu+ p)*ln(J) - 1/(2*lmbda)*p**2
233 bulk_energy = bulk_energy_density*dx
234
235 # Surface energy
236 gamma=Expression("t",t=0.00)
237 surface_energy_density = gamma*current_element_of_area
238 surface_energy = surface_energy_density*ds(1)
239
240 # Total potential energy
241 potential_energy = bulk_energy + surface_energy
242
243 # First directional derivative of the potential energy (a linear form in the test function vq)
244 F=derivative(potential_energy,up,vq)
245
246 # First directional derivative of the potential energy (a bilinear form in the test function vq and the trial function dup)
247 dF=derivative(F,up,dup)
248
249 # Setup the variational problem
250 varproblem = NonlinearVariationalProblem(F, up, bc_u, J=dF,form_compiler_parameters=ffc_options)
251
252 #-------------------------------------------------------------
253 #               # Set up the solver (Newton solver)
254 #-------------------------------------------------------------
255 solver = NonlinearVariationalSolver(varproblem)
256 solver.parameters.update(parameters.solver)
257
258 #-------------------------------------------------------------
259 #               # Solve the problem
260 #-------------------------------------------------------------
261 # loading parameter (list of values for the surface tension)
262 gamma_list = np.linspace(user_par.gamma_min,user_par.gamma_max,user_par.gamma_nsteps) # list of values of surface tension for the simulations
263 # directory and files to save the results
264 save_dir = parameters.user.save_dir
265 file_u = File(save_dir+"/displacement."+parameters.user.output_type)
266 file_p = File(save_dir+"/pressure."+parameters.user.output_type)
267 # Solve with Newton solver for each value of the surface tension, using the previous solution as a starting point.
268
269 for t in gamma_list:
270     # update the value of the surface tension
271     gamma.t = t
272     # solve the nonlinear problem (using Newton solver)
273     solver.solve()
274     # Save solution to file (readable by Paraview or Visit)
275     (p,u) = up.split()
276     file_u << (u,t)
277     file_p << (p,t)
278     # Plot and save png image
279     if parameters.user.plot:
280             plot_u = plot(u, mode = "displacement",title="Displacement field gamma=%.4f"%t,elevate=25.0)
281         plot_u.write_png(save_dir+"/displacement_%.4f"%t)
282
283 # save the parameters to file
284 File(save_dir+"/parameters.xml") << parameters
285
286 # get timings and save to file
287 if MPI.process_number() == 0:
288     timings_str = timings().str("Timings")
289     text_file = open(save_dir+"/timings.txt", "w")
290     text_file.write(timings_str)
291     text_file.close()
```

References

[1] F. Auricchio, L. Beirao de Veiga, C. Lovadina, and A. Reali. The importance of the exact satisfaction of the incompressibility constraint in nonlinear elasticity: mixed fems versus nurbs-based approximations. *Comput. Methods Appl. Mech. Engrg.*, 199:314–323, 2010.

[2] Ciarlet. *Mathematical elasticity, Three dimensional elasticity*, volume 1. North-Holland, 1998.

[3] P. Le Tallec. *Numerical methods for nonlinear three-dimensional elasticity*, volume III of *Handbook of numerical analysis*. Elsevier Science B.V., 1994.

[4] A. Logg, K.-A. Mardal, Wells G. N., and al. *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2012.

[5] P. Wriggers. *Nonlinear finite element methods*. Springer, 2001.