Mathematica® and COMSOL Multiphysics®: A Powerful Workflow for Creating General FE Formulations

Francesco Costanzo and Priyanka Patki

Center for Neural Engineering
Penn State, University Park, USA

2019 COMSOL Conference
Boston, MA
Octtober 2–4, 2019

Acknowledgments

Collaborators

- Ravi T. Kedarasetti, Ph.D. Candidate, Center for Neural Engineering, Penn State
- Nitesh Nama, former Ph.D. student now a postdoc at the University of Michigan.
- Eric Abercrombie, recently graduated M.S. student

We acknowledge partial funding from

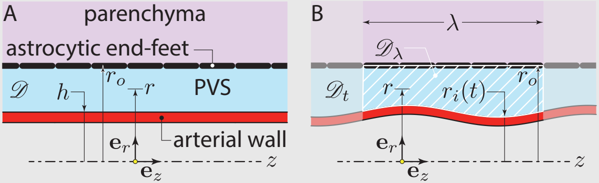
- US National Science Foundation: CBET, Engineering of Biomedical Systems Program (EBMS), Grant #1705854.

Motivation

- Our workplace: **Penn State Center for Neural Engineering**
- Current projects:
  - Microacoustofluidic mixers/cell manipulation
  - Metabolite exchange in brain
  - Acute stroke: blood clotting/lysis, embolization, clot removal
  - Biodegradation of tissue engineered scaffolds
- Math/computation interests:
  - Mixture theory
  - Traditional and complex fluid flow with fluid-structure interaction and stabilization
  - large-strain poroelasticity
  - Reaction-diffusion-advection
- **Need**: to **quickly** develop accurate and reliable multiphysics numerical schemes
- **Preferred tool**: the **Mathematics Interfaces** in COMSOL Multiphysics — we can focus on FE formulations and stabilization and spend less time in low-level programming
- **Main obstacle**: typying very complex formulations into COMSOL without typographical errors.
- **Solution**: **We leaned on Wolfram's Mathematica** — combining Mathematica and COMSOL opened the door to significant progress for us and this presentation is meant to tell you about our experience.

VMM-Stabilized ALE Formulation of Darcy-Brinkman Flow in Axisymmetric Geometry

An example



Strong form: $\underbrace{\frac{\epsilon\mu}{\kappa}(\mathbf{v}-\mathbf{v}_m)+\nabla p-\nabla\cdot[2\mu(\nabla\mathbf{v})_{\mathrm{sym}}]-\mathbf{b}}_{\mathbf{r}\text{: residual}}=\mathbf{0}$ and $\nabla\cdot\mathbf{v}-q=0$ in $\mathscr{D}_\lambda$

Stabilized weak form with companion fine-scale problem:

$$\left(\tilde{\mathbf{v}},\frac{\epsilon\mu}{\kappa}(\mathbf{v}-\mathbf{v}_m)-\mathbf{b}\right)+(\nabla\tilde{\mathbf{v}},\boldsymbol{\sigma})+(\tilde{p},\nabla\cdot\mathbf{v}-q)+(\boldsymbol{\phi},\boldsymbol{\tau}\mathbf{r})=0,\quad \boldsymbol{\sigma}=-p\mathbf{I}+2\mu(\nabla\mathbf{v})_{\mathrm{sym}},$$

$$\boldsymbol{\phi}=-\frac{\epsilon\mu}{\kappa}\tilde{\mathbf{v}}+\nabla\cdot[\tilde{p}\mathbf{I}+2\mu(\nabla\tilde{\mathbf{v}})_{\mathrm{sym}}],\quad \left(\tilde{\boldsymbol{\tau}}\mathbf{w}_i,\frac{\epsilon\mu}{\kappa}\mathbf{w}_i-\mathbf{w}_i\right)+([\nabla(\tilde{\boldsymbol{\tau}}\mathbf{w}_i)]_{\mathrm{sym}},2\mu[\nabla(\boldsymbol{\tau}\mathbf{w}_i)]_{\mathrm{sym}})=0,$$

with $\mathbf{w}_1=-\mathbf{e}_r$ and $\mathbf{w}_2=-\mathbf{e}_z$. **Boundary conditions:** $\mathbf{v}$ given at the inner and outer radii; λ-periodicity; zero-pressure average. The $\boldsymbol{\tau}$ problem is posed in a space of bubble functions.

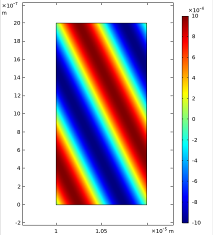
These equations hold over the (deformed) physical domain: They must be pulled back to the computational domain — remapping second order differential operators: chain rule galore.

VMM-Stabilized ALE Formulation of Darcy-Brinkman Flow in Axisymmetric Geometry

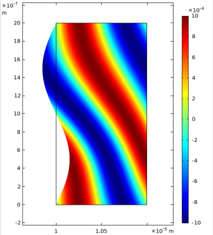
Convergence Study with the Method of Manufactured Solutions

- Figures: **Method of Manufactured Solutions**, radial velocity component.
- Application interest: flow of interstitial fluid in the paravascular space of the brain.
- ALE: Arbitrary Lagrangian–Eulerian
  - Physical domain $\neq$ computational domain
  - **Eqs. are complex on the solution's domain**
  - **Eqs. are "simple" on the physical domain**
- VMM: **Virtual Multiscale Method**

  [for Darcy-Brinkman flow, see A. Masud, IJNMF, **54**(2007), pp. 665–681]
- **VMM "Cost"**
  - Needs the residual (2nd order derivatives).
  - Needs an auxiliary (stabilization) field $\boldsymbol{\tau}$.
- **VMM "Benefits"**
  - Strongly consistent
  - Effective in convection-dominated problems
  - Computing $\boldsymbol{\tau}$ is **transparent** (cf. SUP/G)
  - **Lessens the restrictions of the inf-sup condition**



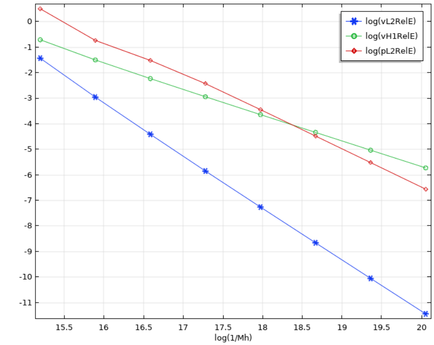
v_r in the computational domain



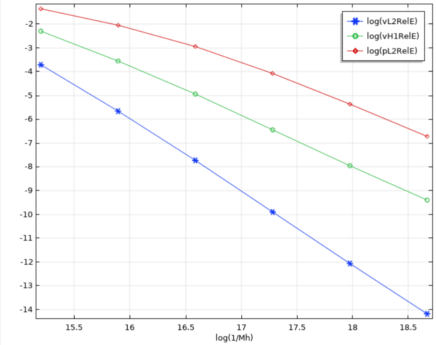
v_r in the physical domain

Example: Convergence Study

- **FE Grid**: squares with side of length **Mh**.
- Parametric sweep over a range of values for Mh.



Q1 elements for p and $\mathbf{v}$.
Cubic bubbles for $\boldsymbol{\tau}$. 8 cycles of uniform refinement. Maximum number of dofs.: 1,577,475. Duration: 105 s on my MacBook Pro.

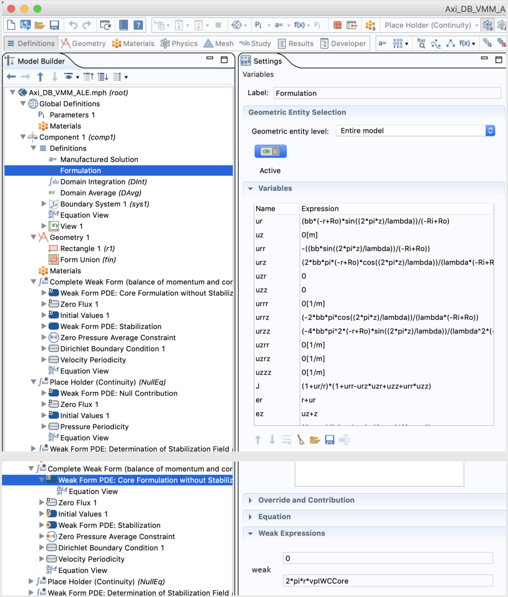


Q2 elements for p and $\mathbf{v}$.
Cubic bubbles for $\boldsymbol{\tau}$. 6 cycles of uniform refinement. Maximum number of dofs.: 395,523. Duration: 25 s, again on my MacBook Pro.

How Were the Calculations Done?

COMSOL's powerful input syntax

- COMSOL provides **Mathematics Interfaces**
- Interfaces we typically use:
  - Weak Form PDE
  - ODE and DAE
- Our "standard approach":
  - **Component Node**: add a physics using a Mathematics Interface — (typically) this instantiates a FE field
  - **Definitions sub-node**: add **Variables Table(s)**: **it is here that we define the formulation expressions**
  - We invoke one such definitions in the appropriate equation interface.
- **We do NOT type definitions**: **we import them from text file(s)**
- **We create the text files using Mathematica**



Quick Reminder about COMSOL's Syntax

- Default coordinate system in COMSOL:
  - Cartesian: (x, y, z)
  - Cylindrical: (r, phi, z)
- **COMSOL allows a user to label an FE field as a whole and to independently name the field's components**. For example, let "$vector\_u$" be the label for a vector-valued field in 3D. We can then name the components $(first, second, third)$.
  - If, say, we are using (x, y, z) for the coordinates, it may make more sense to name the vector components (ux, uy, uz) — again, these are just names.
- Once names are given, COMSOL has an intuitive syntax to refer to derivatives:

$$\frac{\partial uy}{\partial x} \to uyx, \quad \frac{\partial^3 ux}{\partial y \partial z^2} \to uxyzz, \quad \frac{\partial uz}{\partial t} \to uzt, \quad \text{etc.}$$

- Note: time derivatives must go last.
- The test functions associated to a particular field are invoked by simply setting the field in question as the argument of the `test( )` operator.
- So, what does the input for our VMM-stablized Example look like? . . . Let's consider a very simple example first . . .

Input for 3D Elastodynamics

```
IWC
(-bx+rho*uxtt)*test(ux)+(2*mu*uxx+lambda*(uxx+uyy+uzz))*test(uxx)+(-by+
rho*uytt)*test(uy)+mu*(uxy+uyx)*(test(uxy)+test(uyx))+(2*mu*uyy+lambda*(
uxx+uyy+uzz))*test(uyy)+(-bz+rho*uztt)*test(uz)+mu*(uxz+uzx)*(test(uxz)+
test(uzx))+mu*(uyz+uzy)*(test(uyz)+test(uzy))+(2*mu*uzz+lambda*(uxx+uyy+
uzz))*test(uzz)
BWC -(sx*test(ux))-sy*test(uy)-sz*test(uz)
Energy
(lambda*(uxx+uyy+uzz)^2+mu*(2*uxx^2+(uxy+uyx)^2+2*uyy^2+(uxz+uzx)^2+(uyz
+uzy)^2+2*uzz^2))/2.
exx uxx
exy (uxy+uyx)/2.
exz (uxz+uzx)/2.
eyx (uxy+uyx)/2.
eyy uyy
eyz (uyz+uzy)/2.
ezx (uxz+uzx)/2.
ezy (uyz+uzy)/2.
ezz uzz
sxx (exx+eyy+ezz)*lambda+2*exx*mu
sxy 2*exy*mu
sxz 2*exz*mu
syx 2*eyx*mu
syy (exx+eyy+ezz)*lambda+2*eyy*mu
syz 2*eyz*mu
szx 2*ezx*mu
szy 2*ezy*mu
szz (exx+eyy+ezz)*lambda+2*ezz*mu
```

- Define a single physics with a vector-valued displacement field $\mathbf{u}$.
- Components of $\mathbf{u}$: (ux, uy, uz).
- The weak form of the linear elastic BVP is

$$(\tilde{\mathbf{u}}, \partial_{tt}\mathbf{u} - \mathbf{b}) + ([\nabla\tilde{\mathbf{u}}]_{\mathrm{sym}}, \boldsymbol{\sigma}) - (\tilde{\mathbf{u}}, \mathbf{s})_{\Gamma_N} = 0$$

$$\boldsymbol{\sigma} = 2\mu\boldsymbol{\varepsilon} + \lambda(\operatorname{tr}\boldsymbol{\varepsilon})\mathbf{I}$$

$$\boldsymbol{\varepsilon} = (\nabla\mathbf{u})_{\mathrm{sym}}$$

  λ and μ are the Lamè elastic constants (moduli).
- Boundary conditions: $\mathbf{u} = \mathbf{u}_0$ on Γ_D and $\boldsymbol{\sigma}\mathbf{n} = \mathbf{s}$ on Γ_N, with Γ_D and Γ_N the Dirichlet and the Neumann parts of the boundary.

Input for the VMM-Stabilized Porous Flow Example

Actually, only one term . . .

Definition of the stabilization term:

```
StabilizationIWC
J*((taurr*(-br+pr*rer+(2*muB*vr)/er^2+(epsilon*muD*vr)/kappa+pz*zer-(2*
muB*(rer*vrr+vrz*zer))/er-muB*(2*rerer*vrr+rezez*vrr+2*rer^2*vrrr+rez^2*
vrrr+rerez*vzr+rer*rez*vzrr+4*rer*vrrz*zer+rez*vzrz*zer+2*vrzz*zer^2+2*
vrz*zerer+vzz*zerez+2*rez*vrrz*zez+rer*vzrz*zez+vzzz*zer*zez+vrzz*zez^2+
vrz*zezez))+taurz*(-bz+pr*rez+(epsilon*muD*(c+vz))/kappa+pz*zez-(muB*(
rez*vrr+rer*vzr+vzz*zer+vrz*zez))/er-muB*(rerez*vrr+rerer*vzr+rer^2*vzrr
+rez*vrrz*zer+vzzz*zer^2+vzz*zerer+vrz*zerez+vrzz*zer*zez+rer*(rez*vrrr+
2*vzrz*zer+vrrz*zez))-2*muB*(rezez*vzr+rez^2*vzrr+2*rez*vzrz*zez+vzzz*
zez^2+vzz*zezez)))*(rer*test(pr)+zer*test(pz)-(epsilon*muD*test(vr))/
kappa+(2*muB*(-test(vr)+er*rer*test(vrr)+er*zer*test(vrz)))/er^2+2*muB*(
rerer*test(vrr)+rer^2*test(vrrr)+2*rer*zer*test(vrrz)+zerer*test(vrz)+
zer^2*test(vrzz))+muB*(rezez*test(vrr)+rez^2*test(vrrr)+2*rez*zez*test(
vrrz)+zezez*test(vrz)+zez^2*test(vrzz)+rerez*test(vzr)+rer*rez*test(vzrr
)+rez*zer*test(vzrz)+rer*zez*test(vzrz)+zerez*test(vzz)+zer*zez*test(
vzzz)))+(tauzr*(-br+pr*rer+(2*muB*vr)/er^2+(epsilon*muD*vr)/kappa+pz*zer
-(2*muB*(rer*vrr+vrz*zer))/er-muB*(2*rerer*vrr+rezez*vrr+2*rer^2*vrrr+
rez^2*vrrr+rerez*vzr+rer*rez*vzrr+4*rer*vrrz*zer+rez*vzrz*zer+2*vrzz*zer
^2+2*vrz*zerer+vzz*zerez+2*rez*vrrz*zez+rer*vzrz*zez+vzzz*zer*zez+vrzz*
zez^2+vrz*zezez))+tauzz*(-bz+pr*rez+(epsilon*muD*(c+vz))/kappa+pz*zez-(
muB*(rez*vrr+rer*vzr+vzz*zer+vrz*zez))/er-muB*(rerez*vrr+rerer*vzr+rer^2
*vzrr+rez*vrrz*zer+vzzz*zer^2+vzz*zerer+vrz*zerez+vrzz*zer*zez+rer*(rez*
vrrr+2*vzrz*zer+vrrz*zez))-2*muB*(rezez*vzr+rez^2*vzrr+2*rez*vzrz*zez+
vzzz*zez^2+vzz*zezez)))*(rez*test(pr)+zez*test(pz)-(epsilon*muD*test(vz)
)/kappa+(muB*(rez*test(vrr)+zez*test(vrz)+rer*test(vzr)+zer*test(vzz)))/
er+muB*(rerez*test(vrr)+rer*rez*test(vrrr)+rez*zer*test(vrrz)+rer*zez*
test(vrrz)+zerez*test(vrz)+zer*zez*test(vrzz)+rerer*test(vzr)+rer^2*test
(vzrr)+2*rer*zer*test(vzrz)+zerer*test(vzz)+zer^2*test(vzzz))+2*muB*(
rezez*test(vzr)+rez^2*test(vzrr)+2*rez*zez*test(vzrz)+zezez*test(vzz)+
zez^2*test(vzzz))))
```

I did not type this, Mathematica did!

- The whole formulation is contained in a single text file with 4776 characters, forming 1045 “words”, over 30 lines — the line to the left has 2007 characters
- This is not the most complex input we asked COMSOL to read
- The COMSOL parser is very resilient
- The true limitation is on the part of the user when he/she might need to debug errors

Let’s see how this is created with Mathematica . . .

Reduced Coordinates

Weak Forms

$$\left(\tilde{\mathbf{v}}, \frac{\epsilon\mu}{\kappa}(\mathbf{v}-\mathbf{v}_m)-\mathbf{b}\right)+(\nabla\tilde{\mathbf{v}},\boldsymbol{\sigma})+(\tilde{p},\nabla\cdot\mathbf{v}-q)+(\boldsymbol{\phi},\boldsymbol{\tau}\mathbf{r})=0, \quad \boldsymbol{\sigma}=-p\mathsf{I}+2\mu(\nabla\mathbf{v})_{\mathrm{sym}},$$

$$\boldsymbol{\phi}=-\frac{\epsilon\mu}{\kappa}\tilde{\mathbf{v}}+\nabla\cdot[\tilde{p}\mathsf{I}+2\mu(\nabla\tilde{\mathbf{v}})_{\mathrm{sym}}], \quad \left(\tilde{\boldsymbol{\tau}}\mathbf{w}_i, \frac{\epsilon\mu}{\kappa}\mathbf{w}_i-\mathbf{w}_i\right)+([\nabla(\tilde{\boldsymbol{\tau}}\mathbf{w}_i)]_{\mathrm{sym}}, 2\mu[\nabla(\boldsymbol{\tau}\mathbf{w}_i)]_{\mathrm{sym}})=0,$$

Balance of Momentum and Continuity Equation: Core without VMM stabilization

Expression of the residual

```
In[34]:= Residual = (epsilon muD / kappa) (vF - vm) - bF - Div[σ[pF, vF], ECoordinates, "Cylindrical"];
```

Stabilization test function

```
In[35]:= AdjointTestStress = test[pF] × IdentityMatrix[3] +
           2 muB Sym[Grad[test[vF], ECoordinates, "Cylindrical"]];

In[36]:= StabilizationTestFunction =
          - (epsilon muD / kappa) test[vF] + Div[AdjointTestStress, ECoordinates, "Cylindrical"];
```

Stabilization contribution to weak form

```
In[37]:= StabilizationIWCLong = J IP[StabilizationTestFunction, tauF.Residual];

In[38]:= StabilizationIWC =
          Simplify[Compactify[Compactify[StabilizationIWCLong, RLECoordinates], RECoordinates]]
```

no other

Mathematica COMSOL Support Package

Key features

- Our Mathematica package allows one to mimic the "paper and pencil" equations
- We stuck to using Mathematica's native differential operators:
  - **ALE formulations are automatically built via the chain rule!**
  - No limit to the order of differentiation: **determination of residuals for in any coordinate system**
- To use Mathematica's native differential operators, the relevant fields must be written as functions of the coordinates: **This is not allowed in COMSOL**. **Therefore, we created a function called `Compactify[]` to convert Mathematica expressions into COMSOL-compatible equivalents.**
- When solving for the gradients of an inverse ALE Map we rely on the Mathematica's native `Solve[]` function.
- We created a `test[]` operator to imitate the required syntax in COMSOL.
- We created a function called `COMSOLForm[]` as an extension of Mathematica's native `FortranForm[]` to translate Mathematica's expressions into compatible COMSOL definitions.

Package Function List

Here is the full list of the functions in our COMSOLSupport Package

- `IP[*,*]`: generalized inner product
- `Sym[*]`: symmetrization of 2nd order tensors
- `COMSOLProblemSettings[---]`: Initialization of problem type, dimension, coordinate system, naming convention
- `test[*]`: a linear operator obeying the product rule, commuting with differential operators
- `DInt[*]`: a mere wrapper standing for "domain integration"; **must be defined in COMSOL**
- `Tensorify[*,*,*]`: creates an expression representing a tensor of specified order
- `Functionify[*,*]`: turns a symbol into a function of specified arguments
- `Fieldfy[*,*]`: an alias for `Functionify[*,*]`
- `Testify[*]`: alternative to `test[*]` treating test functions as functions
- `Compactify[*,*]`: transform expression functions of specified coordinates into COMSOL-compatible expressions.
- `ClearOutputFiles[*]`: simple file initialization
- `COMSOLExport[*,*]`: writes COMSOL-compatible expressions to a textfile
- `COMSOLForm[*]`: turns a Mathematica expression into one interpretable by the COMSOL parser
- `COMSOLRule[*,*]`: defines the pairs to appear in a COMSOL definition table.

Summary

- Research support mission: rapid deployment of schemes not normally available in ready-made computational packages
- COMSOL Multiphysics is an ideal tool in creating new FE formulations
- Main stumbling block: the expressions in multiphysics simulations are often very hard to hand-type correctly
- We were able to overcome our difficulties by manipulating complex formulations using Mathematica
- We formalized our workflow into a Mathematica Package that has proven to be very versatile and (relatively) easy to use even in the most complex formulations we have worked on so far.
- We will make our Mathematica package and accompanying examples available as a Git repository hosted on GitHub
- Please contact me (Francesco Costanzo) at **fxc8@psu.edu** is you are interested

Any questions?