AUTOMATIC DIFFERENTIATION FOR NONLINEAR, MULTIPHYSICS SIMULATION

Derek Gaston

October 24 2016
MOOSE is a finite-element, multiphysics framework that simplifies the development of numerical applications.

It provides a high-level interface to sophisticated nonlinear solvers and massively parallel computational capability.

Used to model thermomechanics, neutronics, geomechanics, reactive transport, microstructure, computational fluid dynamics...

Open source and freely available at mooseframework.org

High honors:
• Early career award from President Obama
• R&D 100 from R&D Magazine
• Hundreds of publications, thousands of citations
MULTIPHYSICS IS TOUGH!

- Multiphysics: simultaneously solving multiple PDEs representing multiple, coupled, physical phenomena
  - Heat conduction, solid mechanics, neutronics, etc.
- Many different solution schemes
  - Loose, Picard, Fully Coupled, etc.
- Fully Coupled
  - Fast, Robust
  - Solves all equations simultaneously
  - Typically uses a Newton-like solver
- Newton solvers require **Jacobians**

Loose, Picard and Full Coupling
FINITE-ELEMENT CONSTRUCTION

Strong Form: \(-\nabla \cdot D(u) \nabla u = 0\)

Weak Form: \(\int_V D(u) \nabla u \cdot \nabla \psi dV - \int_S (D(u) \nabla u \cdot n) \psi dS = 0\)

Discretized Variable: \(u_h = \sum_k u_k \phi_k\)

Discretized Test Space: \(\psi = \{\phi_i\}\)

\(\nabla u_h = \sum_k u_k \nabla \phi_k\)

Break domain into “elements”. Evaluate integrals using Quadrature.
SOLVING NONLINEAR FE

Nonlinear Vector Equation:

\[ R_i(u_h) = 0 \]

Vector Newton's Method:

\[ \mathbf{J}(u_h^n) \delta u_h^{n+1} = -R_i(u_h^n) \]

\[ u_h^{n+1} = u_h^n + \delta u_h^{n+1} \]

\[ \mathbf{J}_{i,j}(u_h^n) = \frac{\partial R_i(u_h^n)}{\partial u_j} \]

\[ \frac{\partial u_h}{\partial u_j} = \sum_k \frac{\partial}{\partial u_j} (u_k \phi_k) = \phi_j \]

\[ \frac{\partial (\nabla u_h)}{\partial u_j} = \sum_k \frac{\partial}{\partial u_j} (u_k \nabla \phi_k) = \nabla \phi_j \]
The Jacobian gets out of control quickly:

- NxN matrix to store
- NxN matrix to compute
- Differentiation of non-smooth properties
- Tons of analytic derivatives
- Grows as number of equations squared

Could use “Jacobian-Free” methods: JFNK

Could use Automatic Differentiation (AD)…
AUTOMATIC DIFFERENTIATION

• Automatically compute derivative of code

• Many options:
  • Code transformation
  • Reverse Mode
  • Template Metaprogramming
  • Forward Mode via operator overloading
  • …

function residual(args)
    return D*grad_u*grad_psi
end

function jacobian(args)
    return (dD_du*grad_u + D*grad_phi)
    * grad_psi
end
FORWARDDIFF.JL

- Developed by Jarrett Revels (MIT)
  - [https://github.com/JuliaDiff/ForwardDiff.jl](https://github.com/JuliaDiff/ForwardDiff.jl)
- Implements AD via “Dual” number Type and function overloading (perfect for Julia!)
- “Dual” holds both the value and partial derivatives
- As a Dual is operated on the partials are automatically accumulated
- By seeding the partials with orthogonal components, the derivative with respect to multiple variables can be computed simultaneously
- One evaluation of “f” can also evaluate the entire gradient

2016, Revels, J. and Lubin, M. and Papamarkou, T. (view online)
Forward-Mode Automatic Differentiation in Julia
Solution Coefficients: \((u_1, u_2, u_3, u_4)\)

Dual Numbers:
\[
\begin{align*}
u_1 &= [u_1, 1, 0, 0, 0] \\
u_2 &= [u_2, 0, 1, 0, 0] \\
u_3 &= [u_3, 0, 0, 1, 0] \\
u_4 &= [u_4, 0, 0, 0, 1]
\end{align*}
\]

Field Value: \(u_h = \sum_k u_k \phi_k\)

Dual Field Value: \(u_h = [u_h, \phi_1, \phi_2, \phi_3, \phi_4]\)

Residual Entry: \(R_i = R(u_h, \psi_i)\)

Dual Residual:
\[
R_i = R(u_h, \psi_i) = [R_i, \frac{\partial R_i}{\partial u_1}, \frac{\partial R_i}{\partial u_2}, \frac{\partial R_i}{\partial u_3}, \frac{\partial R_i}{\partial u_4}]
\]

\(J_{i,j} = \)
MAX_CHUNK_SIZE

- ForwardDiff.jl defines: MAX_CHUNK_SIZE = 10
- Prohibitive!
  - Can only solve for 2 variables on a Quad4 grid!
- I modified ForwardDiff.jl to set MAX_CHUNK_SIZE = 256
- WARNING: This parameter controls procedurally generated type definitions!
  - Setting this too high (I tried 1000) will cause Julia to take forever to compile ForwardDiff.jl!
MOOSE.JL

- Reimplementation of MOOSE in Julia
  - https://github.com/friedmud/moose.jl
  - Pkg.clone("https://github.com/friedmud/MOOSE.jl.git")
  - Full test suite with CI: https://travis-ci.org/friedmud/MOOSE.jl

- “plug-and-play” equation system construction

- Automatic differentiation for Jacobian calculation

- Built on:
  - ForwardDiff.jl (Jarrett Revels)
  - JuAFEM.jl (Kristoffer Carlsson)
  - ContMechTensors.jl (Kristoffer Carlsson)
  - FastGaussQuadrature.jl (Alex Townsend)

- Much more left to do! (Final Project)
using MOOSE

include("CoupledConvection.jl")

mesh = buildSquare(0, 1, 0, 1, 20, 20)
diffusion_system = System{Float64}(mesh)

u = addVariable!(diffusion_system, "u")
v = addVariable!(diffusion_system, "v")

addKernel!(diffusion_system, Diffusion(u))
addKernel!(diffusion_system, CoupledConvection(u, v))
addKernel!(diffusion_system, Diffusion(v))

addBC!(diffusion_system, DirichletBC(u, [4], 0.0))
addBC!(diffusion_system, DirichletBC(u, [2], 1.0))

addBC!(diffusion_system, DirichletBC(v, [4], 0.0))
addBC!(diffusion_system, DirichletBC(v, [2], 1.0))

initialize!(diffusion_system)
solver = JuliaDenseNonlinearImplicitSolver(diffusion_system)
solve!(solver, nl_max_its=5)

out = VTKOutput()
output(out, solver, "coupled_convection_out")

Solves:
\[-\nabla \cdot \nabla u + \nabla v \cdot \nabla u = 0\]
\[u = 0, u \in S_{left}\]
\[u = 1, u \in S_{right}\]

\[-\nabla \cdot \nabla v = 0\]
\[v = 0, v \in S_{left}\]
\[v = 1, v \in S_{right}\]
“NO COUPLING” PERFORMANCE

Equations:

\[-\nabla \cdot \nabla u_i + u_i^2 = 0\]

\[u_i = 0, u_i \in S_{left}\]

\[u_i = 1, u_i \in S_{right}\]

NonlinearForce Assembly Time (20x20, 5x)

20x20 Mesh, 5 Assembly Calculations
"FULLY COUPLED" PERFORMANCE

Equations:
\[-\nabla \cdot \nabla u_i + \sum_{k,k \neq i} \nabla u_k \nabla u_i = 0\]
\[u_i = 0, u_i \in S_{\text{left}}\]
\[u_i = 1, u_i \in S_{\text{right}}\]

20x20 Mesh, 5 Assembly Calculations
CONCLUSIONS

• Julia is working well for Multiphysics!

• Open source packages accelerate development

• ForwardDiff.jl provides effective AD for FE

• MOOSE.jl is now available (and will continue to improve)