Dan++: A Quick Look

Daniel Driver

April 16, 2015

What now

1. What is Dan++
2. Work Flow
3. Examples and Demo

What is Dan++

What is Dan++

Bread And Butter

We Need

$$\mathbf{F}(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}^m$$

Particle Dynamics Example:

$$\frac{d^2\mathbf{x}}{dt^2} = \frac{F_{orce}(\mathbf{x})}{\mathbf{M}}$$

$$\mathbf{F}(\mathbf{x}) = \frac{F_{orce}(\mathbf{x})}{\mathbf{M}}$$

FEM Example:

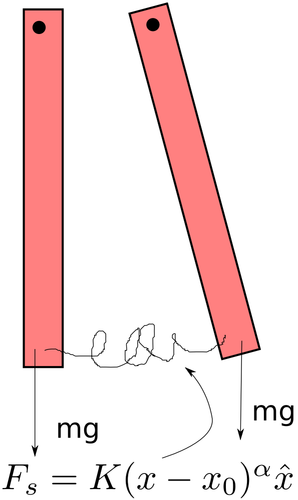
$$\nabla \cdot \mathbf{K}(u(\mathbf{u}))\nabla u(\mathbf{u}) = 0$$

$$\mathbf{F}(\mathbf{u}) = \int_\Omega \mathbf{K}(u(\mathbf{u}))\nabla_\mathbf{x} u(\mathbf{u}) \cdot \nabla_\mathbf{x} \phi d\mathbf{x}$$

u is the field u

$\mathbf{u}$ is a vector of the nodal values

ϕ is a vector of shape functions



Bread And Butter

Often Want:

$$\nabla_{\mathbf{x}}\mathbf{F}(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}^m \times \mathbb{R}^n$$

FEM Revisited:
$\mathbf{F}(\mathbf{u}) = \int_\Omega \mathbf{K}(u(\mathbf{u}))\nabla_{\mathbf{x}} u(\mathbf{u}) \cdot \nabla_{\mathbf{x}}\phi dx$

Linear :
$K_{global} = \nabla_{\mathbf{u}}\mathbf{F}(\mathbf{u})$
$K_{global}\mathbf{u} = R_{global}$
K_{global} no longer function of $\mathbf{u}$

Nonlinear(Newtons Method):
$\mathbf{F}(\mathbf{u}) = 0$
$\mathbf{u}^{n+1} = \mathbf{u}^n - \left(\nabla_{\mathbf{u}}\mathbf{F}(\mathbf{u})|_{\mathbf{u}_n}\right) \mathbf{F}(\mathbf{u})|_{\mathbf{u}_n}$

Time Stepping: $\frac{d\mathbf{x}}{dt} = \mathbf{F}(\mathbf{x})$

Explicit Euler :
$\mathbf{x}^{n+1} = \mathbf{x}^n + h\mathbf{F}(\mathbf{x}^n)$

Implicit Midpoint:
$\mathbf{x}^{n+1} = \mathbf{x}^n + \frac{h}{2}(\mathbf{F}(\mathbf{x}^n) + \mathbf{F}(\mathbf{x}^{n+1}))$
$\mathbf{F}'(\mathbf{x}^{n+1}) =$
$\mathbf{x}^{n+1} - \mathbf{x}^n - \frac{h}{2}(\mathbf{F}(\mathbf{x}^n) + \mathbf{F}(\mathbf{x}^{n+1})) = \mathbf{0}$
$\mathbf{x}^{n+1,i+1} = \mathbf{x}^{n+1,i} -$
$\left(\nabla_{\mathbf{x}}\mathbf{F}'(\mathbf{x})|_{\mathbf{x}^{n+1,i}}\right) \mathbf{F}'(\mathbf{x})|_{\mathbf{x}^{n+1,i}}$

Dan++ Philosophy

- Simulation Should be Described
  - Not “Coded”
- User Sets up $\mathbf{F}(\mathbf{x})$
  - Combine $\mathbf{F}_{elem}(\mathbf{x}) \rightarrow \mathbf{F}(\mathbf{x})$
  - Describes $\mathbf{F}_{elem}(\mathbf{x})$ with math expression not operations on data
- Expressive Assembly Syntax
  - What goes Where
  - No specifying indexes
- “Free” Gradient $\nabla_{\mathbf{x}}\mathbf{F}(\mathbf{x})$
  - No more work from user
  - Linear and Non-Linear
  - Fully Implicit
- No one should Take Derivatives
  - EVER! (ok maybe for theory)
  - Mostly a waste of time
  - Usually wrong
- Still Want Performance
  - Hidden Low Level for Number Crunching

Current Implementation

Dan++

- Defines Assembly Syntax
- Takes Sympy Expressions
- Writes a C Function
  - F and ∇F
- Returns Python Functions that call C-code

- SymPy(Symbolic Python)
  - to Describe Equations
  - Symbolic Derivative For Jacobian
  - Translates Basic expression to C code
- SciPy.Weave
  - Wraps C code
  - Makes Python Module
  - Uses Numpy arrays as template
    - Uses to Automatically Builds Wrapper

Work Flow

Work Flow

DEM

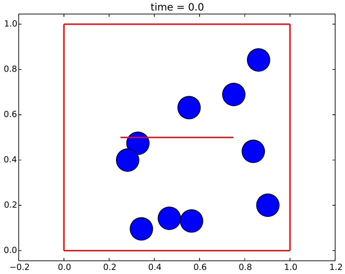


Figure : Discrete Element(Blue) simulation with walls(Red).

Work Flow with Dan++ I

1. Allocate Required Data Arrays as DanArrays
    1. State Array - Doubles to be solved for
    2. Parameter Array - Doubles need by not solved
    3. Connectivity - Integers
2. Assign meaning to Data using DanArray methods
    1. This is mostly done by forming Sub-Arrays and reshaping
    2. Helper Functions to Make Allocation and Reshaping Easier

Work Flow with Dan++ II

3. Make Mesh Class
    1. Connectivity Class to Designate Associations
        1. Static Connectivity
    2. Combine Connectivities To Define Mesh
        1. Determines how Loops in C are written
        2. Nested loops over Connectivities
4. Build Physics
    1. Use Sympy to create SymExpression Classes
        1. Store Expression for $\mathbf{F}_{elem}(\mathbf{x})$ and symbols are variables and parameters
    2. Make physics classes
        1. Takes SymExpression Class and a mapping from data to symbols
5. Add Physics to Mesh to contribute to $\mathbf{F}(\mathbf{x})$

Work Flow with Dan++ III

6. Make an Assembler object
   1. Inputs: State, Parameter, Connectivity Array
   2. Use AddMesh Method to add Assembly routines over mesh
7. Call MakePythonModule method of Assembly Object
   1. Inputs- Name and Matrix type:[Dense or COO]
   2. Automatically writes C code for Expressions
   3. Automatically take gradients and writes C-code
   4. Parses meshes and writes Assembler in C
   5. Wraps Assembly routine to call in Python using Scipy's Weave

Work Flow with Dan++ IV

8. Use wrapped assembly routine to calculate $\mathbf{F}(\mathbf{x})$ and $\nabla_{\mathbf{x}}\mathbf{F}(\mathbf{x})$
   1. f_func calculates **just** $\mathbf{F}(\mathbf{x})$
   2. gradf calculates **just** $\nabla_{\mathbf{x}}\mathbf{F}(\mathbf{x})$
   3. f_gradf calculates **Both** $\nabla_{\mathbf{x}}\mathbf{F}(\mathbf{x})$
9. Solve Matrix system using NumPy or SciPy(sparse) routines
   1. Newtons method to solve $\mathbf{F}(\mathbf{x}) = \mathbf{0}$
   2. RK Methods to solve $\frac{dx}{dt} = \mathbf{F}(\mathbf{x})$

Examples and Demo

Examples and Demo

Consistent(automatically) Linearized DEM

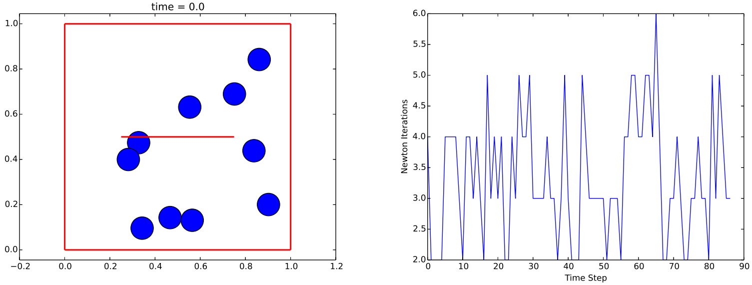
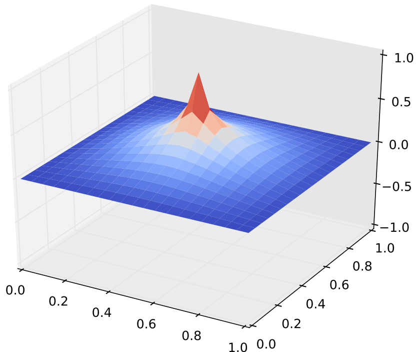


Figure : Left:DEM simulation with walls. Right:Newton Iterations per time step for the 2-stage Fully implicit Gauss Legendre RK method (4th order).

Finite Difference:Linear Laplace Eqn

$$\nabla \cdot \mathbf{K}(u)\nabla u(x,y) = \mathbf{0}$$

$$\mathbf{F}(u) = \begin{cases} \nabla \cdot \mathbf{K}(u)\nabla u(x,y) & \Omega \backslash \Omega_D \\ u - u_b & \Omega_D \end{cases}$$



Finite Elements:Linear Laplace Eqn

$$\mathbf{F}(u) = \mathbf{F}_{weak}(u) + \mathbf{F}_{pen}(u) = \int_{\Omega} \mathbf{K}(u) \nabla u(x,y) \cdot \nabla_{\mathbf{x}} \phi_i d\mathbf{x} + K_s (u - u_b)|_{\Omega_D}$$

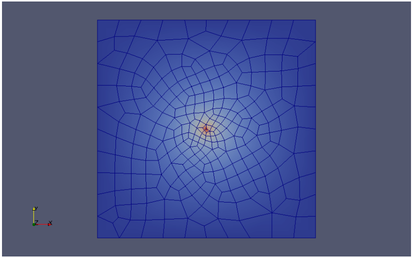


Figure : Solution of the Lapace equation with $\mathbf{K}(u) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

Finite Elements:Non-Linear Laplace Eqn

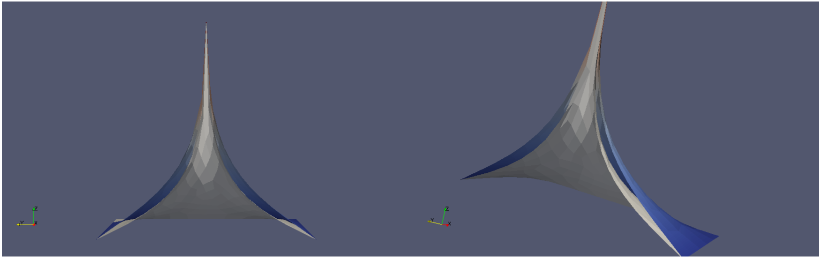


Figure : Solution of the Lapace equation with$\mathbf{K}(u) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$in solid gray. Solution to$\mathbf{K}(u) = \begin{bmatrix} u+1 & 0 \\ 0 & u+1 \end{bmatrix}$ in color map corresponding to the the height,u. Notice the gap between the two solutions.

Thank You