SyFi - An Element Matrix Factory, with Emphasis on the Incompressible Navier-Stokes Equations

Kent-Andre Mardal

20th June 2006

Outline - Short Description of SyFi

Purpose

- SyFi is a tool for defining polygons, polynomial spaces, degrees of freedom, and finite elements
- SyFi makes it easy to define weak forms (differentiation and integration of polynomials over polygons)
- SyFi computes matrices (element matrices or global matrices)
- SyFi generates efficient C++ code for global matrices

Dependencies

- SyFi relies heavily on GiNaC and Swiginac for the symbolic computations and code generation
- SyFi can generate matrices based on either a Dolfin or a Diffpack mesh (we plan to include other meshes soon)
- SyFi can generate either Epetra or PyCC matrices (we plan to include other matrices soon).

Short Description of GiNaC and Swiginac

GiNaC

- GiNaC is a C++ library for symbolic mathematics
- URL: www.ginac.de
- License : GPL

Swiginac

- Swiginac is a Python interface to GiNaC
- URL: http://swiginac.berlios.de/
- License: Open

Swiginac Code Example

```
from swiginac import *
x = symbol('x'); y = symbol('y')

f = sin(x*x*y)
print "f = ", f

dfdx = diff(f,x)
print "df/dx = ", dfdx

g = pow(x,9)*(1-x)
intg = integral(x,0,1,g)
```

SyFi Extends GiNaC/Swiginac

GiNaC/Swiginac supports:

- Polynomials
- Differentiation with respect to one variable
- Integration with respect to one variable

Basic Extensions in SyFi:

- Polynomial spaces (such as Legendre and Bernstein)
- Differentiations with respect to several variables
- Integration over polygons

$\rightarrow$ SyFi extends GiNaC/Swiginac with the ingredients typically needed in finite element methods

Elements currently implemented in SyFi

Finite Elements

- continuous and discontinuous Lagrangian elements (arbitrary order)
- Nedelec elements (arbitrary order)
- Raviart-Thomas elements (arbitrary order)
- Crouzeix-Raviart elements
- Hermite elements
- Bubble elements

Matrices/Weak forms

- mass matrix
- convection matrix
- diffusion matrix
- divergence matrix

Evaluation of Weak Forms in SyFi

We will now demonstrate the computation of various element matrices in SyFi

- The divergence matrix with mixed elements:

$$\boldsymbol{B}_{ij} = \int_T \nabla \cdot \boldsymbol{N}_i \, L_j \, dx$$

  where $\boldsymbol{N}_i$ and L_j are the basis functions of the velocity and pressure elements

- The computation of the Jacobian matrix

$$\boldsymbol{J}_{ij} = \frac{\partial \boldsymbol{F}_i}{\partial u_j}$$

  in the case of nonlinear convection diffusion

$$\boldsymbol{F}_i = \int_T (\boldsymbol{u} \cdot \nabla \boldsymbol{u}) \cdot \boldsymbol{N}_i + \nabla \boldsymbol{u} : \nabla \boldsymbol{N}_i) \, dx$$

SyFi Code Example : Divergence Matrix

```
from swiginac import *
from SyFi import *

polygon = ReferenceTriangle()
v_element = VectorLagrangeFE(polygon,2)
v_element.set_size(2)
v_element.compute_basis_functions()

p_element = LagrangeFE(polygon,1)
p_element.compute_basis_functions()

for i in range(0,v_element.nbf()):
    for j in range(0,p_element.nbf()):
        integrand = div(v_element.N(i))*p_element.N(j)
        Aij =  polygon.integrate(integrand)
        print "A[%d,%d]=%s "% (i,j, Aij)
```

SyFi Code Example : Convection-Diffusion Matrix

```
.. initialize element

for i in range(0,fe.nbf()):

  # compute diffusion term
  fi_diffusion = inner(grad(u), grad(fe.N(i)))

  # compute convection term
  uxgradu = (u.transpose()*grad(u)).evalm()
  fi_convection = inner(uxgradu, fe.N(i), True)

  fi = fi_diffusion + fi_convection
  Fi = polygon.integrate(fi)

  for j in range(0,fe.nbf()):
    # differentiate to get the Jacobian
    Jij = diff(Fi, uj)
    print "J[%d,%d]=%s "%(i,j,Jij)
```

SyFi Code Example : Convection and Power-law Diffusion

```
n = symbol("n")

for i in range(0,fe.nbf()):

  # nonlinear power-law diffusion term
  mu = inner(grad(u), grad(u))
  fi_diffusion = pow(mu,n)*inner(grad(u), grad(fe.N(i)))

  # nonlinear convection term
  uxgradu = (u.transpose()*grad(u)).evalm()
  fi_convection = inner(uxgradu, fe.N(i), True)

  fi = fi_diffusion + fi_convection

  Fi = polygon.integrate(fi)

  for j in range(0,fe.nbf()):
    # differentiate to get the Jacobian
    Jij = diff(Fi, uj)
    print "J[%d,%d]=%s\n"%(i,j, Jij.evalf().printc())
```

Creation of Epetra Matrices

```
class EpetraMatrixFactory
{
public:
  Epetra_Comm        *Comm;
  Epetra_Map         *Map;
  MapMatrixFactory   *mapfac;
  Mesh               *mesh;
  Dof_ptv            *idof,jdof;
  int order1,order2;

  EpetraMatrixFactory(Mesh *mesh, Epetra_Comm* Comm);

  // scalar mass matrix based on Lagrangian elements
  Epetra_CrsMatrix*  computeMassMatrix();

  // scalar stiffness matrix based on Lagrangian elements
  Epetra_CrsMatrix*  computeStiffnessMatrix();

  // scalar convection matrix based on Lagrangian elements
  Epetra_CrsMatrix*  computeConvectionMatrix(VectorFunction& func,);

  // vector div matrix based on vector Lagrangian elements
  Epetra_CrsMatrix*  computeDivMatrix();

};
```

Concluding Remarks

Present Use of SyFi

- We have implemented a set of simple test examples within SyFi/PyCC for the Poisson problem, Stokes problem, convection-diffusion problems and Navier-Stokes equations
- We have used SyFi/PyCC in rather advanced applications concerning the electrical activity of the heart

Future

- Support other meshes and other matrices
- Implement more elements
- Develop fluid-structure solvers within PyCC