Solving differential equations

- Build a calculator
- One button for each equation?
- Too many buttons!
Automate the solution of differential equations

- Input: Differential equation
- Output: Solution

- Generate computer code
- Compute solution

- Build a calculator for each equation!
- Automate the generation of calculators!
Automate the solution of differential equations

\[-\Delta u = f\]
Simplicity

- Simple and intuitive user interface
  - Close to mathematical abstractions
  - Close to mathematical notation
  - But give users what they expect: Matrix, Vector, Mesh, Form (?)

- Simplicity of implementation
  - Simple components: each component does one thing and does it well
  - Simple components: independent development
  - Simplicity by (new) mathematical ideas
  - Simplicity by (new) programming techniques
Generality and efficiency

- Any equation
- Any (finite element) method
- Maximum efficiency

Possible to combine generality with efficiency by generating code
Generality

- Any (multilinear) form
- Any element:
  - $P_k$: Arbitrary degree Lagrange elements
  - $DG_k$: Arbitrary degree discontinuous elements
  - $BDM_k$: Arbitrary degree Brezzi–Douglas–Marini
  - $RT_k$: Arbitrary degree Raviart-Thomas
  - $CR_1$: Crouzeix–Raviart
  - ... (more in preparation)
- 2D (triangles) and 3D (tetrahedra)
Efficiency

- CPU time for computing the “element stiffness matrix”
- Straight-line C++ code generated by the FEniCS Form Compiler (FFC)
- Speedup vs a standard quadrature-based C++ code with loops over quadrature points

<table>
<thead>
<tr>
<th>Form</th>
<th>$q = 1$</th>
<th>$q = 2$</th>
<th>$q = 3$</th>
<th>$q = 4$</th>
<th>$q = 5$</th>
<th>$q = 6$</th>
<th>$q = 7$</th>
<th>$q = 8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mass 2D</td>
<td>12</td>
<td>31</td>
<td>50</td>
<td>78</td>
<td>108</td>
<td>147</td>
<td>183</td>
<td>232</td>
</tr>
<tr>
<td>Mass 3D</td>
<td>21</td>
<td>81</td>
<td>189</td>
<td>355</td>
<td>616</td>
<td>881</td>
<td>1442</td>
<td>1475</td>
</tr>
<tr>
<td>Poisson 2D</td>
<td>8</td>
<td>29</td>
<td>56</td>
<td>86</td>
<td>129</td>
<td>144</td>
<td>189</td>
<td>236</td>
</tr>
<tr>
<td>Poisson 3D</td>
<td>9</td>
<td>56</td>
<td>143</td>
<td>259</td>
<td>427</td>
<td>341</td>
<td>285</td>
<td>356</td>
</tr>
<tr>
<td>Navier–Stokes 2D</td>
<td>32</td>
<td>33</td>
<td>53</td>
<td>37</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Navier–Stokes 3D</td>
<td>77</td>
<td>100</td>
<td>61</td>
<td>42</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Elasticity 2D</td>
<td>10</td>
<td>43</td>
<td>67</td>
<td>97</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Elasticity 3D</td>
<td>14</td>
<td>87</td>
<td>103</td>
<td>134</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>
The FEniCS project

- Initiated in 2003
- Collaborators (in order of appearance):
  - (Chalmers University of Technology)
  - University of Chicago
  - Argonne National Laboratory
  - (Toyota Technological Institute)
  - Delft University of Technology
  - Royal Institute of Technology (KTH)
  - Simula Research Laboratory
  - Texas Tech

- Automation of Computational Mathematical Modeling (ACMM)

- www.fenics.org
Poisson’s equation

Differential equation

Differential equation:

\[-\Delta u = f\]

- Heat transfer
- Electrostatics
- Magnetostatics
- Fluid flow
- etc.
Poisson’s equation

Variational formulation

Find $u \in V$ such that

$$a(v, u) = L(v) \quad \forall v \in \hat{V}$$

where

$$a(v, u) = \int_{\Omega} \nabla v \cdot \nabla u \, dx$$

$$L(v) = \int_{\Omega} vf \, dx$$
Poisson’s equation

Implementation

```
element = FiniteElement("Lagrange", "triangle", 1)

v = TestFunction(element)
u = TrialFunction(element)
f = Function(element)

a = dot(grad(v), grad(u))*dx
L = v*f*dx
```
Linear elasticity

Differential equation:

\[-\nabla \cdot \sigma(u) = f\]

where

\[\sigma(v) = 2\mu \varepsilon(v) + \lambda \text{tr} \varepsilon(v) I\]
\[\varepsilon(v) = \frac{1}{2} (\nabla v + (\nabla v)^\top)\]

- Displacement \( u = u(x) \)
- Stress \( \sigma = \sigma(x) \)
Find $u \in V$ such that

$$a(v, u) = L(v) \quad \forall v \in \hat{V}$$

where

$$a(v, u) = \int_{\Omega} \nabla v : \sigma(u) \, dx$$

$$L(v) = \int_{\Omega} v \cdot f \, dx$$
Linear elasticity

Implementation

```
element = VectorElement("Lagrange", "tetrahedron", 1)

v = TestFunction(element)
u = TrialFunction(element)
f = Function(element)

def epsilon(v):
    return 0.5*(grad(v) + transp(grad(v)))

def sigma(v):
    return 2*mu*epsilon(v) + lmbda*mult(trace(epsilon(v)), I)

a = dot(grad(v), sigma(u))*dx
L = dot(v, f)*dx
```
The Stokes equations

Differential equation:

\[-\Delta u + \nabla p = f\]
\[\nabla \cdot u = 0\]

- Fluid velocity $u = u(x)$
- Pressure $p = p(x)$
The Stokes equations

Variational formulation

Find \((u, p) \in V\) such that

\[
a((v, q), (u, p)) = L((v, q)) \quad \forall (v, q) \in \hat{V}
\]

where

\[
a((v, q), (u, p)) = \int_{\Omega} \nabla v \cdot \nabla u - \nabla \cdot v p + q \nabla \cdot u \, dx
\]

\[
L((v, q)) = \int_{\Omega} v \cdot f \, dx
\]
**The Stokes equations**

**Implementation**

\[
P2 = \text{VectorElement}("\text{Lagrange}", "\text{triangle}", 2)\\
P1 = \text{FiniteElement}("\text{Lagrange}", "\text{triangle}", 1)\\
TH = P2 + P1\\
(v, q) = \text{TestFunctions}(TH)\\
(u, p) = \text{TrialFunctions}(TH)\\
f = \text{Function}(P2)\\
\]

\[
a = (\text{dot(grad}(v), \text{grad}(u)) - \text{div}(v)\cdot p + q\cdot\text{div}(u))*dx\\
L = \text{dot}(v, f)*dx
\]
Mixed Poisson with $H(\text{div})$ elements

Differential equation:

\[
\sigma + \nabla u = 0 \\
\nabla \cdot \sigma = f
\]

- $u \in L_2$
- $\sigma \in H(\text{div})$
Mixed Poisson with $H(\text{div})$ elements

Variational formulation

Find $(\sigma, u) \in V$ such that

$$a((\tau, w), (\sigma, u)) = L((\tau, w)) \quad \forall (\tau, w) \in \hat{V}$$

where

$$a((\tau, w), (\sigma, u)) = \int_{\Omega} \tau \cdot \sigma - \nabla \cdot \tau u + w \nabla \cdot \sigma \, dx$$

$$L((\tau, w)) = \int_{\Omega} w f \, dx$$
Mixed Poisson with $H(\text{div})$ elements

Implementation

```python
BDM1 = FiniteElement("Brezzi-Douglas-Marini", "triangle", 1)
DG0 = FiniteElement("Discontinuous Lagrange", "triangle", 0)

element = BDM1 + DG0

(tau, w) = TestFunctions(element)
(sigma, u) = TrialFunctions(element)

f = Function(DG0)

a = (dot(tau, sigma) - div(tau)*u + w*div(sigma))*dx
L = w*f*dx
```
Poisson’s equation with DG elements

Differential equation:

\[-\Delta u = f\]

- $u \in L_2$
- $u$ discontinuous across element boundaries
Poisson’s equation with DG elements

Variational formulation (interior penalty method)

Find $u \in V$ such that

$$a(v, u) = L(v) \quad \forall v \in \hat{V}$$

where

$$a(v, u) = \int_{\Omega} \nabla v \cdot \nabla u \, dx$$

$$+ \sum_{S} \int_{S} -\langle \nabla v \rangle \cdot [u]_{n} - [v]_{n} \cdot \langle \nabla u \rangle + (\alpha/h)[v]_{n} \cdot [u]_{n} \, dS$$

$$+ \int_{\partial\Omega} -\nabla v \cdot [u]_{n} - [v]_{n} \cdot \nabla u + (\gamma/h)v_{u} \, ds$$

$$L(v) = \int_{\Omega} vf \, dx + \int_{\partial\Omega} vg \, ds$$
Poisson’s equation with DG elements

Implementation

\[
\begin{align*}
\text{DG1} &= \text{FiniteElement}("\text{Discontinuous Lagrange}", \"triangle\", 1) \\
\text{v} &= \text{TestFunction}(\text{DG1}) \\
\text{u} &= \text{TrialFunction}(\text{DG1}) \\
\text{f} &= \text{Function}(\text{DG1}) \\
\text{g} &= \text{Function}(\text{DG1}) \\
\text{n} &= \text{FacetNormal}(\"triangle\") \\
\text{h} &= \text{MeshSize}(\"triangle\") \\
\text{a} &= \text{dot}(\text{grad}(\text{v}), \text{grad}(\text{u}))*\text{dx} - \text{dot}(\text{avg}(\text{grad}(\text{v})), \text{jump}(\text{u}, \text{n}))*\text{dS} \\
&\quad - \text{dot}((\text{jump}(\text{v}, \text{n}), \text{avg}(\text{grad}(\text{u}))))*\text{dS} \\
&\quad + \alpha/\text{h}(\text{\(\text{\texttt{+}}\)})*\text{dot}(\text{jump}(\text{v}, \text{n}), \text{jump}(\text{u}, \text{n}))*\text{dS} \\
&\quad - \text{dot}(\text{grad}(\text{v}), \text{jump}(\text{u}, \text{n}))*\text{ds} - \text{dot}(\text{jump}(\text{v}, \text{n}), \text{grad}(\text{u}))*\text{ds} \\
&\quad + \gamma/\text{h}*\text{v}*\text{u})*\text{ds} \\
\text{L} &= \text{v*}\text{f})*\text{dx} + \text{v*g})*\text{ds}
\end{align*}
\]
Code generation system

\[ -\Delta u = f \]

\[ \Omega \]

\[ u = u(x) \]

#include <ufc.h>
class Poisson: public ufc::form {
    public:
        unsigned int rank() const {
            return 2;
        }
    ...
Software map

- DOLFIN
- FFC
- UF
- Viper
- FIAT
- Instant
- FErari
- PETSc
- uBLAS
- UMFPACK
- SCOTCH
- NumPy
- SyFi
- Ko
- Puffin
- PyCC
Recent updates

- UFC, a framework for finite element assembly
- DG, BDM and RT elements
- A new improved mesh library, adaptive refinement
- Mesh and graph partitioning
- Improved linear algebra supporting PETSc and uBLAS
- Optimized code generation (FErari)
- Improved ODE solvers
- Python bindings
- Bugzilla database, pkg-config, improved manual, compiler support, demos, file formats, built-in plotting, ...
Future plans (highlights)

- UFL/UFC
- Automation of error control
  - Automatic generation of dual problems
  - Automatic generation of a posteriori error estimates
- Improved geometry support (MeshBuilder)
- Debian/Ubuntu packages
- Finite element exterior calculus

→ www.fenics.org ←
Additional slides
Software components

- DOLFIN
- PETSc
- uBLAS
- UMFPACK
- SCOTCH

FFC
- NumPy
- FIAT
- Instant
- FErari

UFC

Viper
- VTK

SyFi
- Ko
- Puffin
- PyCC
\[ Au = f \]

\[ \tilde{A}\tilde{u} = \tilde{f} \]

\[ F(x) = 0 \]

\[ (V, \hat{V}) \]

\[ \text{tol} > 0 \]

\[ U \approx u \]

\[ U \approx u \]

\[ X \approx x \]

\[ \tilde{u} \]

\[ \tilde{f} \]
A common framework: UFL/UFC

- UFL - Unified Form Language
- UFC - Unified Form-assembly Code
- Unify, standardize, extend
- Working prototypes: FFC (Logg), SyFi (Mardal)
void eval(real block[], const AffineMap& map) const
{
    [...]

    block[0] = 0.5*G0_0_0 + 0.5*G0_0_1 +
               0.5*G0_1_0 + 0.5*G0_1_1;
    block[1] = -0.5*G0_0_0 - 0.5*G0_1_0;
    block[2] = -0.5*G0_0_1 - 0.5*G0_1_1;
    block[3] = -0.5*G0_0_0 - 0.5*G0_0_1;
    block[4] = 0.5*G0_0_0;
    block[5] = 0.5*G0_0_1;
    block[6] = -0.5*G0_1_0 - 0.5*G0_1_1;
    block[7] = 0.5*G0_1_0;
    block[8] = 0.5*G0_1_1;
}
void eval(real block[], const AffineMap& map) const
{
    [...]

    block[1] = -0.5*G0_0_0 + -0.5*G0_1_0;
    block[0] = -block[1] + 0.5*G0_0_1 + 0.5*G0_1_1;
    block[7] = -block[1] + -0.5*G0_0_0;
    block[6] = -block[7] + -0.5*G0_1_1;
    block[8] = -block[6] + -0.5*G0_1_0;
    block[2] = -block[8] + -0.5*G0_0_1;
    block[5] = -block[2] + -0.5*G0_1_1;
    block[3] = -block[5] + -0.5*G0_0_0;
    block[4] = -block[1] + -0.5*G0_1_0;
}
Compiling Poisson’s equation: `ffc -f blas`, 36 ops

```c
void eval(real block[], const AffineMap& map) const
{
    [...]  

    cblas_dgemv(CblasRowMajor, CblasNoTrans,
               blas.mi, blas.ni, 1.0,
               blas.Ai, blas.ni, blas.Gi,
               1, 0.0, block, 1);
}
```
Key Features

- Simple and intuitive object-oriented API, C++ or Python
- Automatic and efficient evaluation of variational forms
- Automatic and efficient assembly of linear systems
- General families of finite elements, including arbitrary order continuous and discontinuous Lagrange elements, BDM, RT
- Arbitrary mixed elements
- High-performance parallel linear algebra
- General meshes, adaptive mesh refinement
- mcG(q)/mdG(q) and cG(q)/dG(q) ODE solvers
- Support for a range of output formats for post-processing, including DOLFIN XML, ParaView/Mayavi/VTK, OpenDX, Octave, MATLAB, GiD
- Built-in plotting
Linear algebra

- Complete support for PETSc
  - High-performance parallel linear algebra
  - Krylov solvers, preconditioners
- Complete support for uBLAS
  - BLAS level 1, 2 and 3
  - Dense, packed and sparse matrices
  - C++ operator overloading and expression templates
  - Krylov solvers, preconditioners added by DOLFIN
- Uniform interface to both linear algebra backends
- LU factorization by UMFPACK for uBLAS matrix types
- Eigenvalue problems solved by SLEPc for PETSc matrix types
- Matrix-free solvers ("virtual matrices")