

Center of Excellence

Software Components for Biomedical Flows

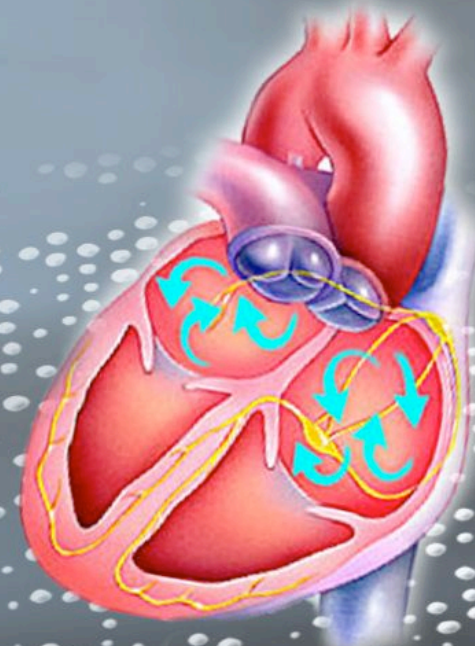
2007 - 2017

```
M = G.compute_mass_matrix(velocity_element)
A = G.compute_stiffness_matrix(velocity_element)
B = G.compute_div_matrix(velocity_element, pressure_element)
D = G.compute_stiffness_matrix(pressure_element)

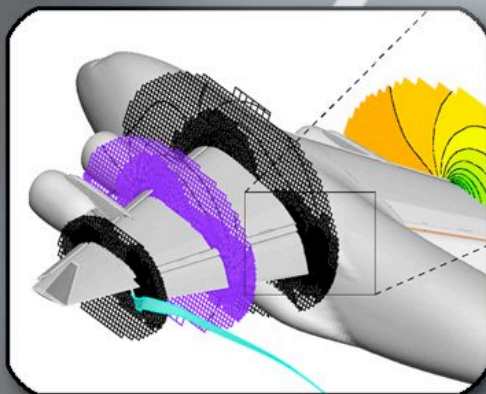
T = 1; dt = 0.1
while t < T:
    t = t + dt;
    f = G.compute_source_vector(rhs)
    C = G.compute_convection_matrix(velocity_element, v_prev)

    A1 = M + dt*A + dt*C
    prec1 = MLPrec(A1)
    v, iter = preconditionBiCGstab(prec1, A1, v, f, 1.0e-9)

    g = (1.0/dt)*B.t*v
    phi, iter = preconditionConjGrad(MLPrec(D), D, phi, g, 1.0e-9)
    v = v - dt*B.t*phi
    p = (Mp + dt*Ap)phi
```



$$\rho \left(\frac{\partial u}{\partial t} + V \cdot \nabla u \right) = - \frac{\partial \tau_{yx}}{\partial x} - \frac{\partial \tau_{zx}}{\partial y} - \frac{\partial \tau_{yz}}{\partial z} + \rho f_x$$

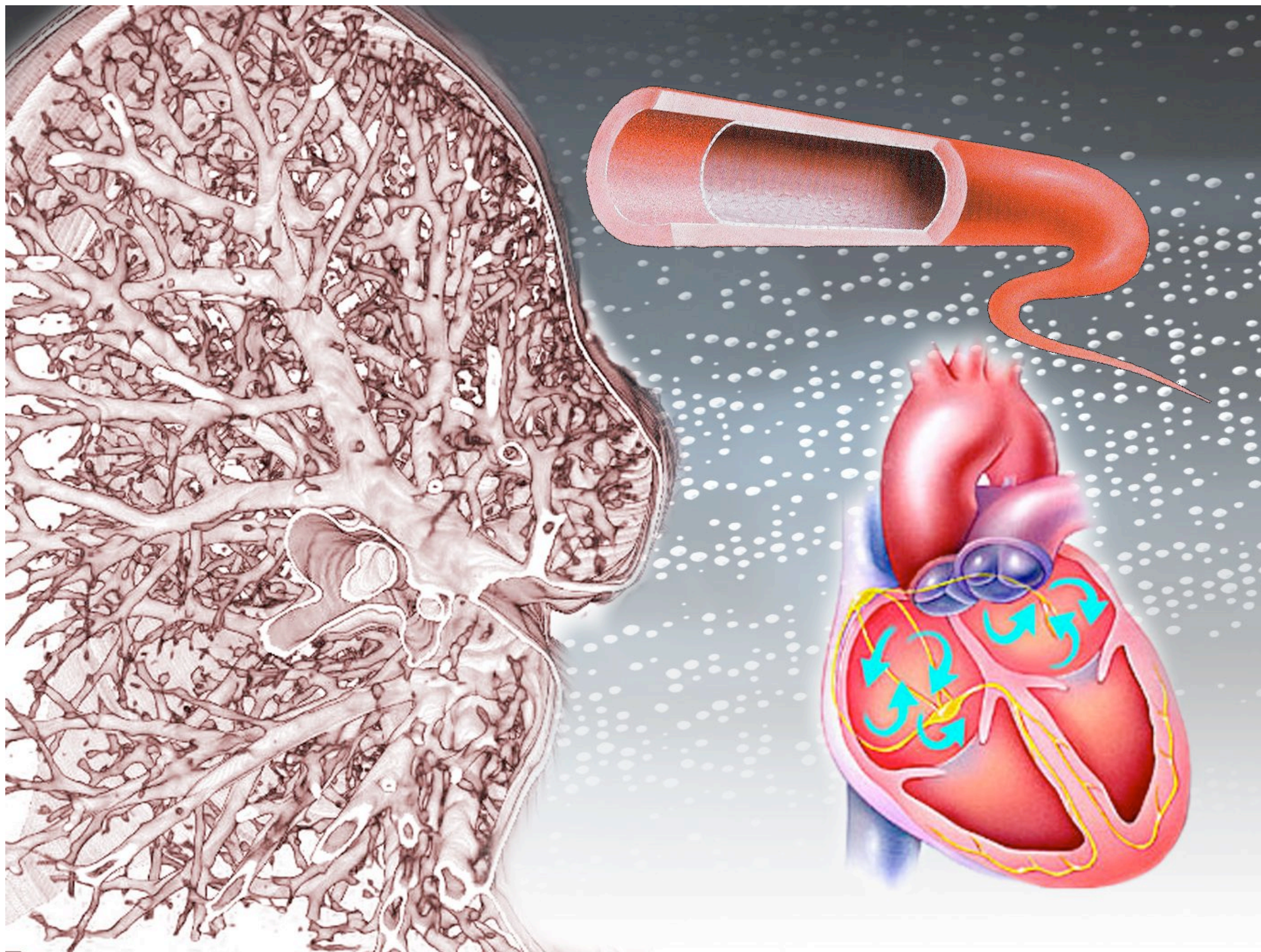


$$\begin{aligned} & \frac{\partial(\rho u)}{\partial t} + \frac{\partial(\rho u^2)}{\partial x} + \frac{\partial(\rho u v)}{\partial y} + \frac{\partial(\rho u w)}{\partial z} \\ &= -\frac{\partial p}{\partial x} + \frac{\partial}{\partial x} \left(\lambda \nabla \cdot \vec{u} + 2\mu \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \mu \left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right) + \rho f_x \\ & \frac{\partial(\rho v)}{\partial t} + \frac{\partial(\rho u v)}{\partial x} + \frac{\partial(\rho v^2)}{\partial y} + \frac{\partial(\rho v w)}{\partial z} \\ &= -\frac{\partial p}{\partial y} + \frac{\partial}{\partial x} \mu \left(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y} \right) + \rho f_y \end{aligned}$$



```
void PressureIntg::integrands
(ElmMatVec& elmat, const FiniteElement)
{
    const int nsd = fe.getNoSpaceDim();
    real div_v=0;
    for (int k = 1; k <= nsd; k++) {
        data->u()(k).derivativeFEM (data->gradu_pt(k)(k);
        div_v += data->gradu_pt(k)(k);
        // (gradu_pt is a VecSimple(Ptv(real)))
    }
}
```

$$\begin{aligned} F_i^r &\equiv \int_{\Omega} \alpha (c_{ij} N_j^{r,l} - c_{ij} N_j^{r,l-1}) + \theta \Delta t d_{ij}^l N_j^{r,l} \\ &\quad + (1-\theta) \Delta t d_{ij}^{l-1} N_j^{r,l-1} d\Omega + \\ &\quad \int_{\Omega} (\theta \lambda \Delta t q_i^{r,l} + (1-\theta) \lambda \Delta t q_i^{r,l-1}) d\Omega - \\ &\quad \int_{\partial\Omega_N^r} (\theta \Delta t F_i^{r,l} + (1-\theta) \Delta t F_i^{r,l-1}) d\Gamma \end{aligned}$$



$$\frac{\partial N}{\partial t} + N \cdot \nabla N = -\frac{1}{\rho} \nabla p + \nu \nabla^2 N + g$$

$$\nabla \cdot N = 0$$

$$\begin{bmatrix} N & Q \\ Q^T & -\varepsilon D \end{bmatrix} \begin{bmatrix} u \\ p \end{bmatrix} = \begin{bmatrix} F \\ -\varepsilon d \end{bmatrix}$$

$$\nabla^2 N^* + \Delta t g^{l+1} = N^{l+1}$$

$$N^l \cdot \nabla N^* + \Delta t$$

$$N^* = N^l -$$

$$\beta \nabla p^l$$

$$= \frac{\rho}{\Delta t} \nabla \cdot$$

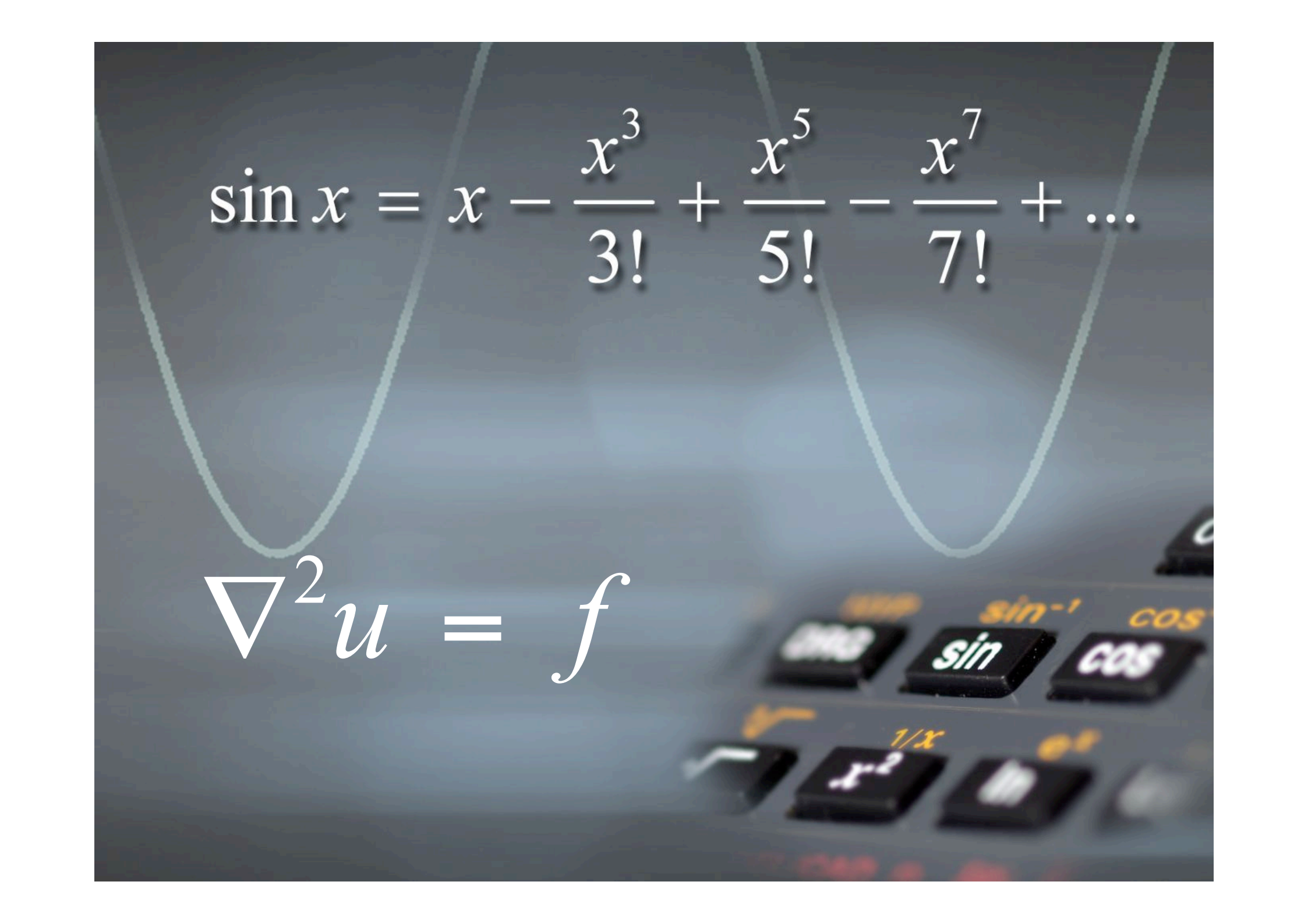
$$\nabla^2 \varphi$$

$$N^{l+1} = N$$

$$p^{l+1} = \beta p^l + \varphi$$

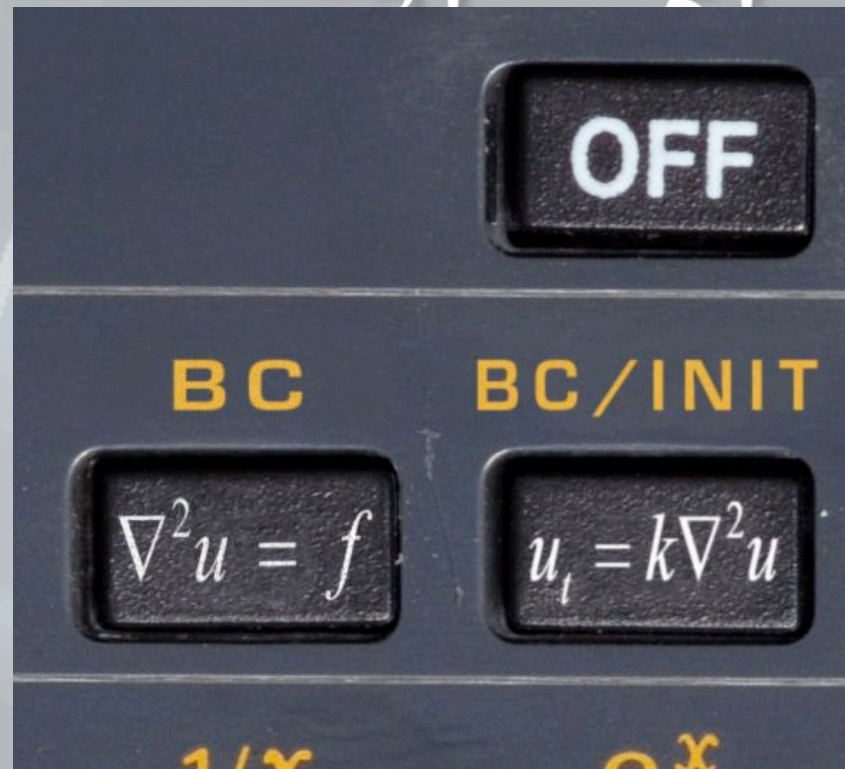
$$p^{l+1} = \beta p^l + \varphi$$

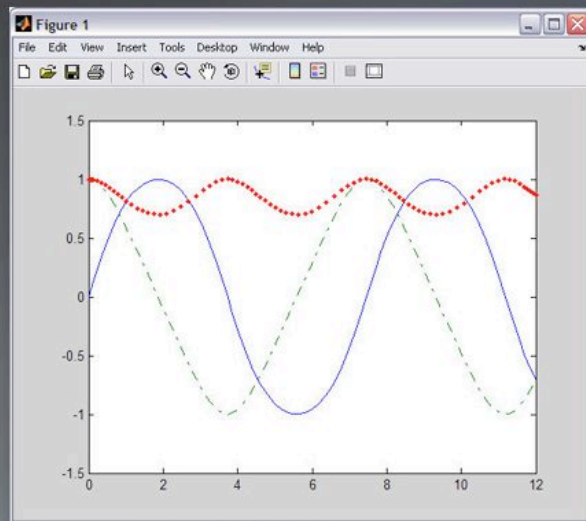
$$N^{l+1} = N^* - \frac{\Delta t}{\rho} \nabla \varphi$$

The background of the image is a close-up, slightly blurred photograph of a scientific calculator. A light blue sine wave is superimposed over the calculator's surface, arching from the left side, peaking in the center, and arching again towards the right. The wave's peaks are positioned above the two main equations. The calculator's buttons are visible in the lower right, with labels like 'sin', 'cos', 'sin⁻¹', 'cos⁻¹', 'x²', and '1/x' in white and yellow text.
$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

$$\nabla^2 u = f$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$





```

C:\Documents and Settings\jol\Desktop\div\hpl\div\matlab

Command Window

To get started, select MATLAB Help or Demos from the Help menu.

>> options = odeset('RelTol',1e-4,'AbsTol',[1e-4 1e-4 1e-5]);
>> [T,Y] = ode45(@rigid,[0 12],[0 1 1],options);
>> plot(T,Y(:,1),'-',T,Y(:,2),'--',T,Y(:,3),'-.');
>>

```

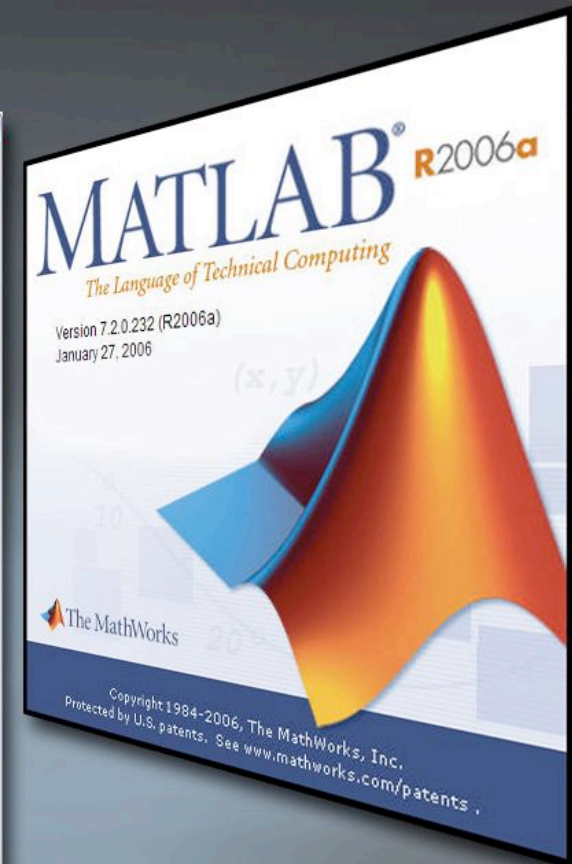
```

Editor - C:\Documents and Settings\jol\Desktop\div\hpl\div\matlab\rigid.m

File Edit Text Go Cell Tools Debug Desktop Window Help

function dy = rigid(t,y)
% An example of a nonstiff system is the system of equations describing the
% motion of a rigid body without external forces.
%
dy = zeros(3,1);
dy(1) = y(2) * y(3);
dy(2) = -y(1) * y(3);
dy(3) = -0.51 * y(1) * y(2);

```



$$\frac{dy}{dt} = f(y,t), \quad y(t_0) = y_0$$

$$Ax = b, \quad Ax = \lambda x$$

```

if (efix .and. qr(i-1,1).lt.0.d0
& .and. ql(i,1).gt.0.d0) then
    amdq(i,1) = -qr(i-1,1)
    apdq(i,1) = ql(i,1)**2

```

```

M = G.compute_mass_matrix
A = G.compute_stiffness_matrix
B = G.compute_div_matrix(velo)
D = G.compute_stiffness_matrix

```

```

T = 1; dt = 0.1
while t < T:
    t = t + dt
    f = G.compute_source_vector(rhs)
    C = G.compute_convection_matrix(velo)

```

```

A1 = M + dt*A + dt*C
prec1 = MLPrec(A1)
v_iter = precendRiCGStab(prec1, A1, v, f, 1.0e-6)

```

```

template <class _Tp, class _Allocator, bool _IsStatic> class
public:
    typedef typename _Alloc_traits<_Tp, _Allocator>::allocator_type;
    allocator_type get_allocator() const { return _Node_allocator; }
    _List_alloc_base(const allocator_type& __a) : _Node_allocator(__a) {}
#ifdef __NO_ARROW_OPERATOR
    pointer operator->() const { return &(operator*()); } #endif

```



FLUENT

The Right Answer in CFD



```

import NSfast, ConvDiffReact_john
t = 0
while t <= T:
    report.write('solve at T=%g' % t)
    t += dt
    v, p = NSfast.advance(t, t_prev, v, p, v_prev, p_prev)
    v2 = ConvDiffReact_john.memlayout(v, from_file)
    c = ConvDiffReact_john.advance(t, t_prev, v2,
                                   param['ConvDiffReact_john', 10])
    diskdata.write(v, p, c, label='t=%g' % t)
    updateGUI()
    v_prev, v2_prev, p_prev, c_prev = v, v2, p, c
    report.end()
diskdata.close()

```

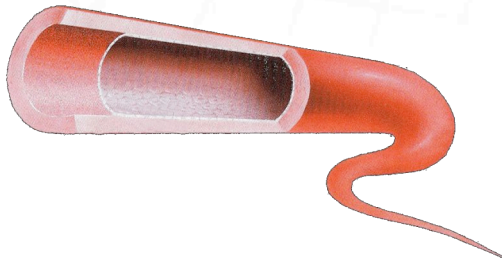
```

GOTO 30
20 CONTINUE
   NHKLG = 1
   WK1(3) = NHKLG + YU(P,4)
   WK1(4) = QW3P + YU(P+1,4)
   WK1(5) = HZ0 + YU(P+1BVZ,1)
30 CONTINUE
   CALL SLV16(A1,K,NZ,P,NP,U,NU,V
>   GFTERA,MQ,HJOPAS,MB
>   NHKLG,IERR1,IERR2,IERR
>   LHJFGD,KKH1,KKH4,KKH

```



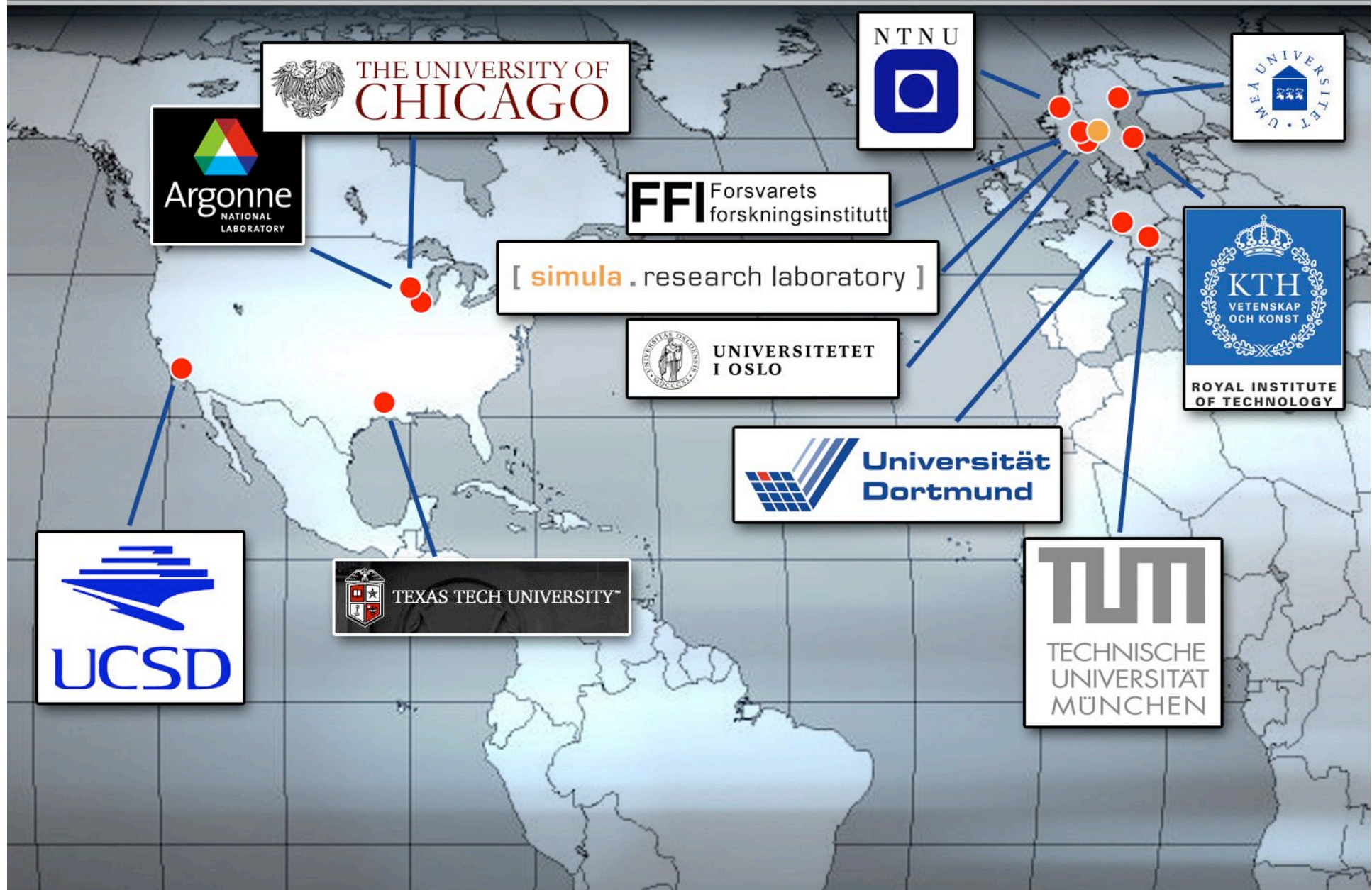
$$\begin{bmatrix} N & Q \\ Q^T & -\varepsilon D \end{bmatrix} \begin{bmatrix} u \\ P \end{bmatrix} = \begin{bmatrix} 0 \\ -\varepsilon d \end{bmatrix}$$



Three main parts

- Computational middleware
- Robust flow solvers
- Biomedical flow applications

SOFTWARE COMPONENTS FOR BIOMEDICAL FLOWS



Computational middleware

```
import NSfast, ConvDiffReact_john
t = 0
while t <= T:
    report.write('solve at T=%g' % t)
    t += dt
    v, p = NSfast.advance(t, t_prev, v, p, v_prev, p_prev)
    v2 = ConvDiffReact_john.memlayout(v, from, to)
    c = ConvDiffReact_john.advance(t, t_prev, v2, p, p_prev)
    param['ConvDiffReact_john'] = 10
    diskdata.write(v, p, c, label='t=%g' % t)
    updateGUI()
    v_prev, v2_prev, p_prev, c_prev = v, v2, p, c
    report.end()
diskdata.close()
```

```
template <class T>
public:
    typedef T type;
    allocator<T> alloc;
    _List_alloc<T> _List_alloc;

#ifdef __NO_POINTER_OPERATOR__
    pointer operator~() const { return &(operator~)(); }
```

```
M = G.compute_mass_matrix(velocity_element)
A = G.compute_stiffness_matrix(velocity_element)
B = G.compute_div_matrix(velocity_element, pressure_element)
D = G.compute_stiffness_matrix(pressure_element)

T = 1; dt = 0.1
while t < T:
    t = t + dt
    f = G.compute_source_vector(rhs)
    C = G.compute_convection_matrix(velocity_element)

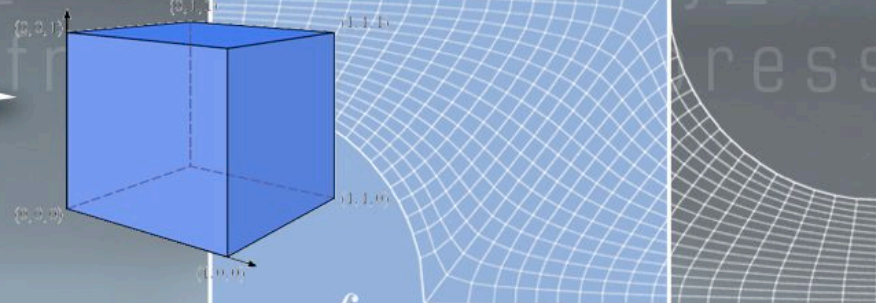
    A1 = M + dt*A + dt*C
    prec1 = MLPrec(A1)
    v, iter = preconditionBiCGStab(prec1, A1, v, f, 1.0e-
```

- Problem solving environments for PDEs
- Old and modern libraries
- Stand-alone black-box codes
- New code (PyCC, FEniCS, automatically generated)
- Parallel computing
- Main focus: fluid flows, but the tools aim at "any" PDE system



PyCC

FENICS



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

PETSc

$$Ax = b$$

$$F(x) = 0$$

Trilinos

$$M^{-1}Ax = M^{-1}b$$

$$U^{n+1} = U^n + \Delta t f(U^n)$$



$$\frac{\partial v}{\partial t} + V \cdot \nabla v =$$

$$\nabla \cdot (k \nabla v) + g(v)$$

$$(\lambda + \mu) \nabla (\nabla \cdot u) + \mu \nabla^2 u =$$

$$\alpha (3\lambda + 2\mu) \nabla T - \rho b$$

$$\rho \left(\frac{\partial u}{\partial t} + V \cdot \nabla u \right) =$$

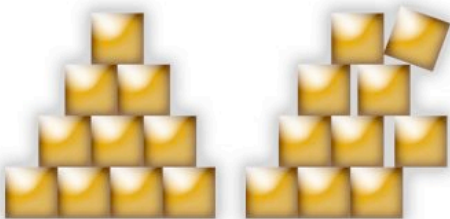
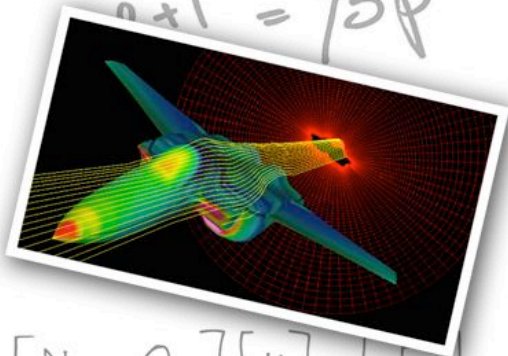
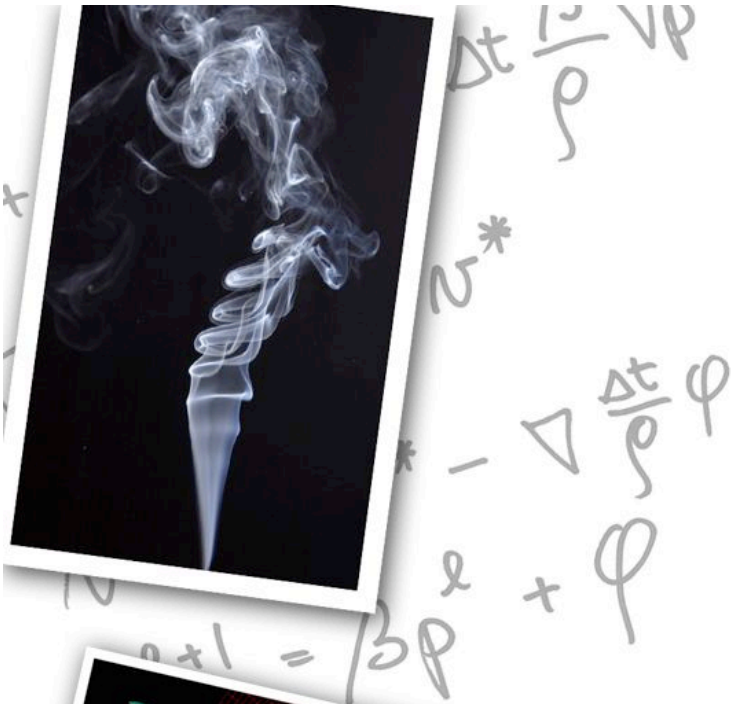
$$-\frac{\partial p}{\partial x} + \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z} + \rho f_x$$

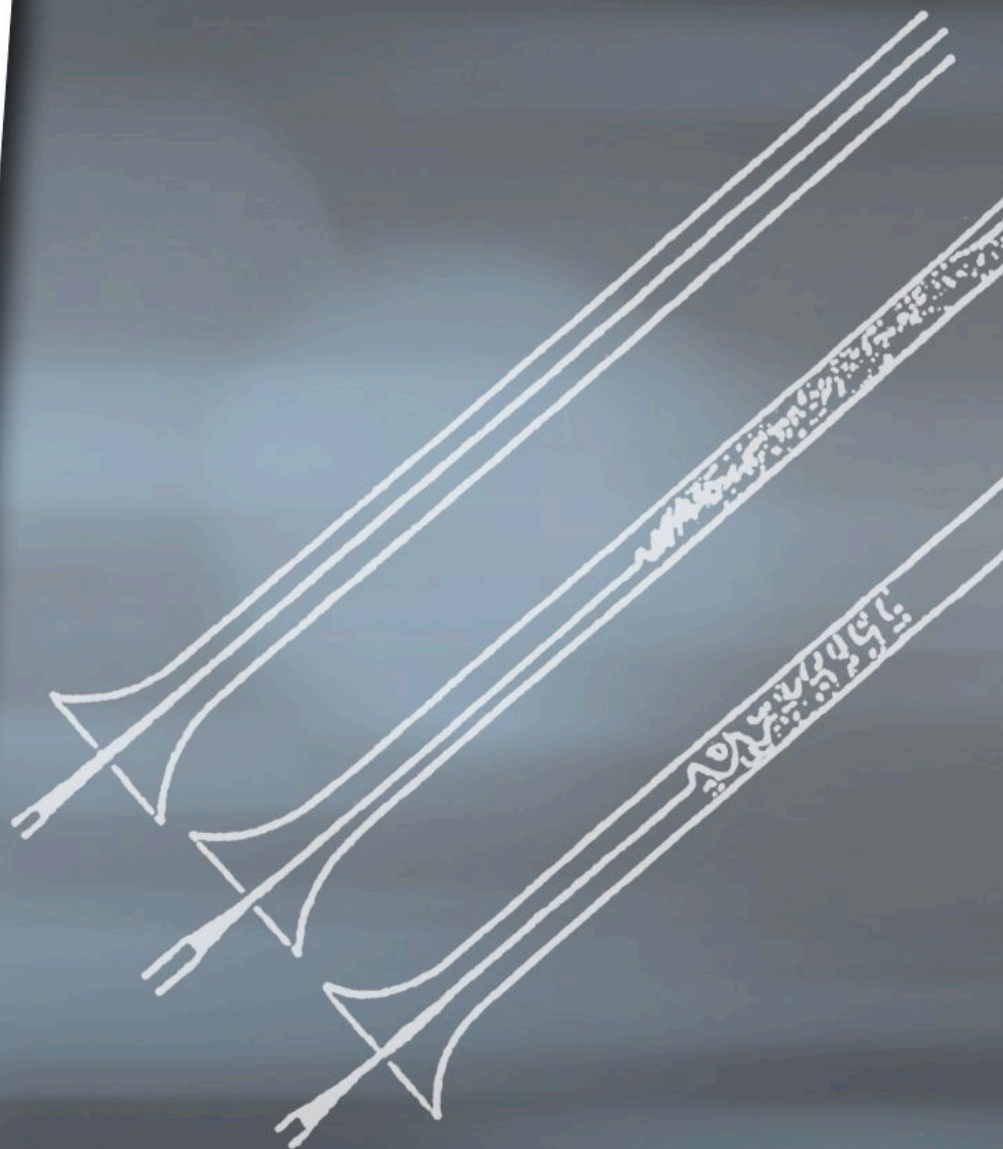
$$\nabla^2 u = f$$



Robust flow solvers

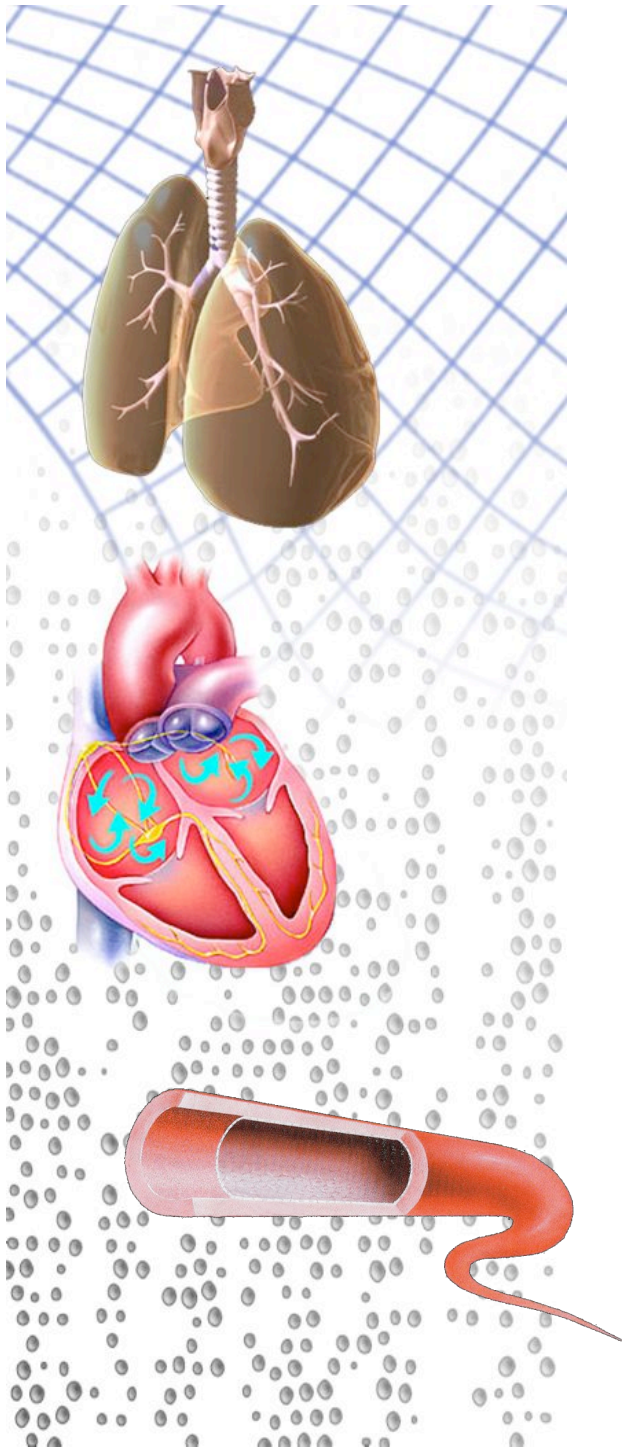
- Implicit methods
- Block preconditioning
- Fluid-structure interaction
- Error estimation
- Adaptivity
- Turbulence modeling & DNS
- Validation by lab. experiments



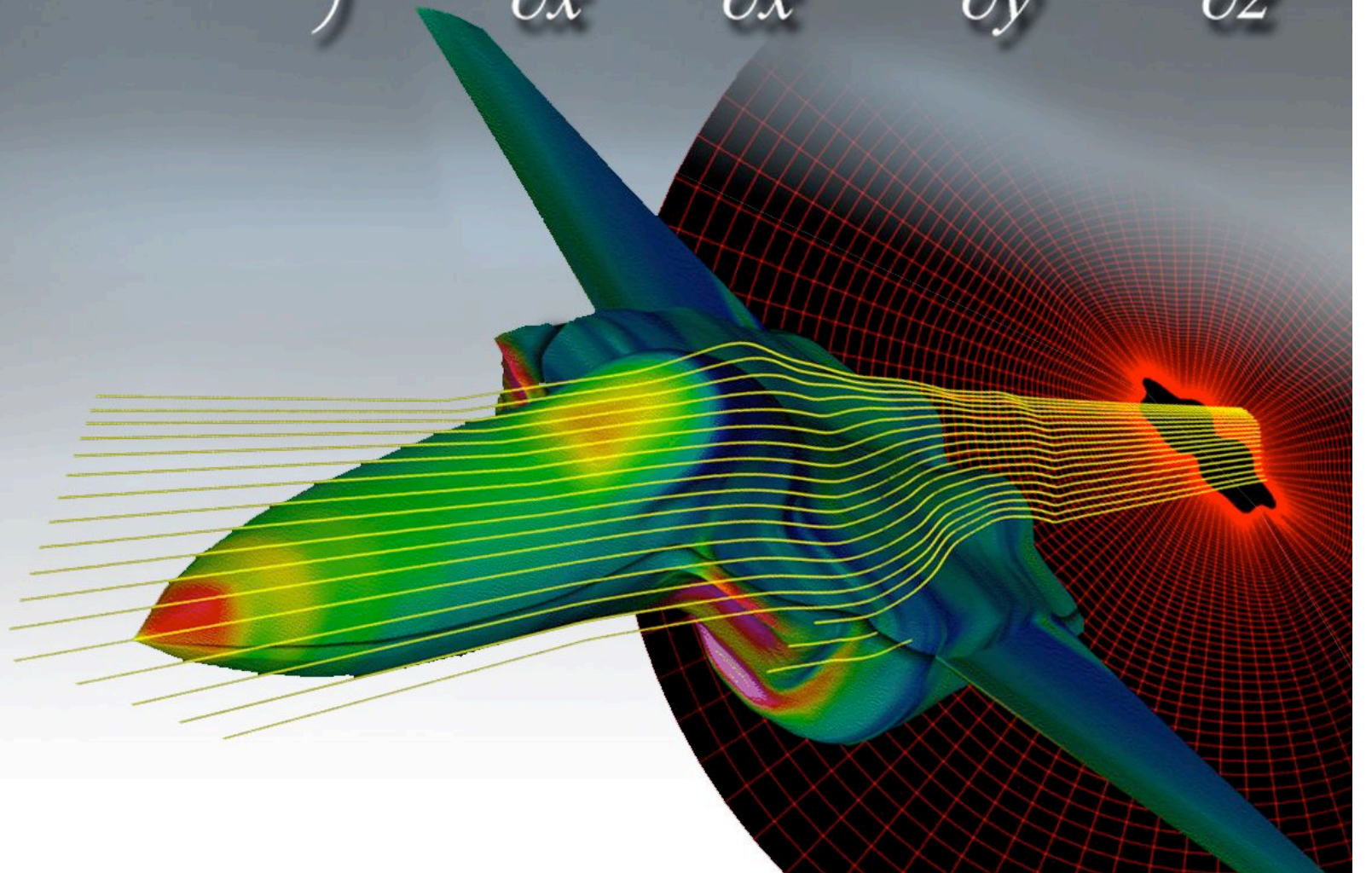


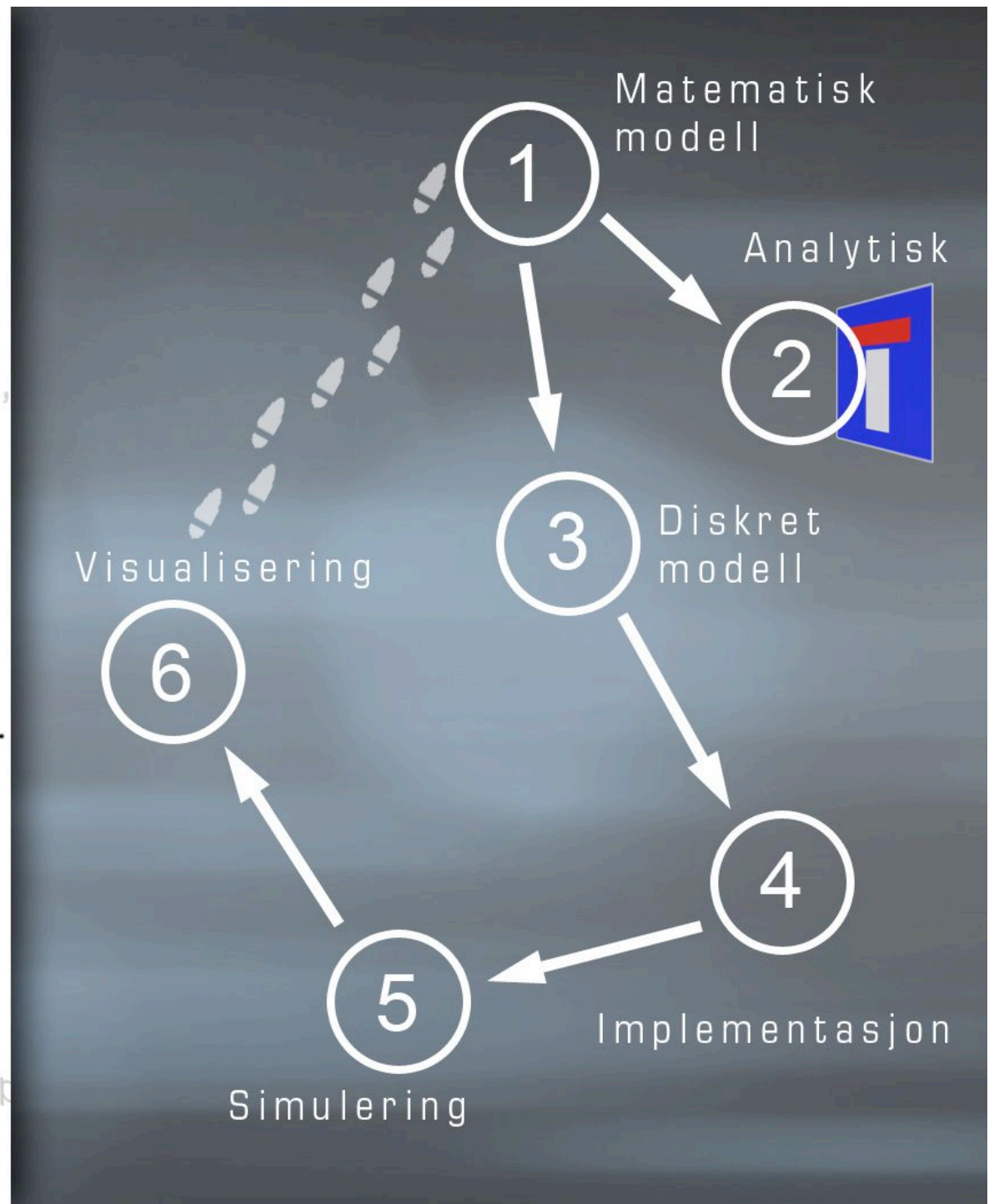
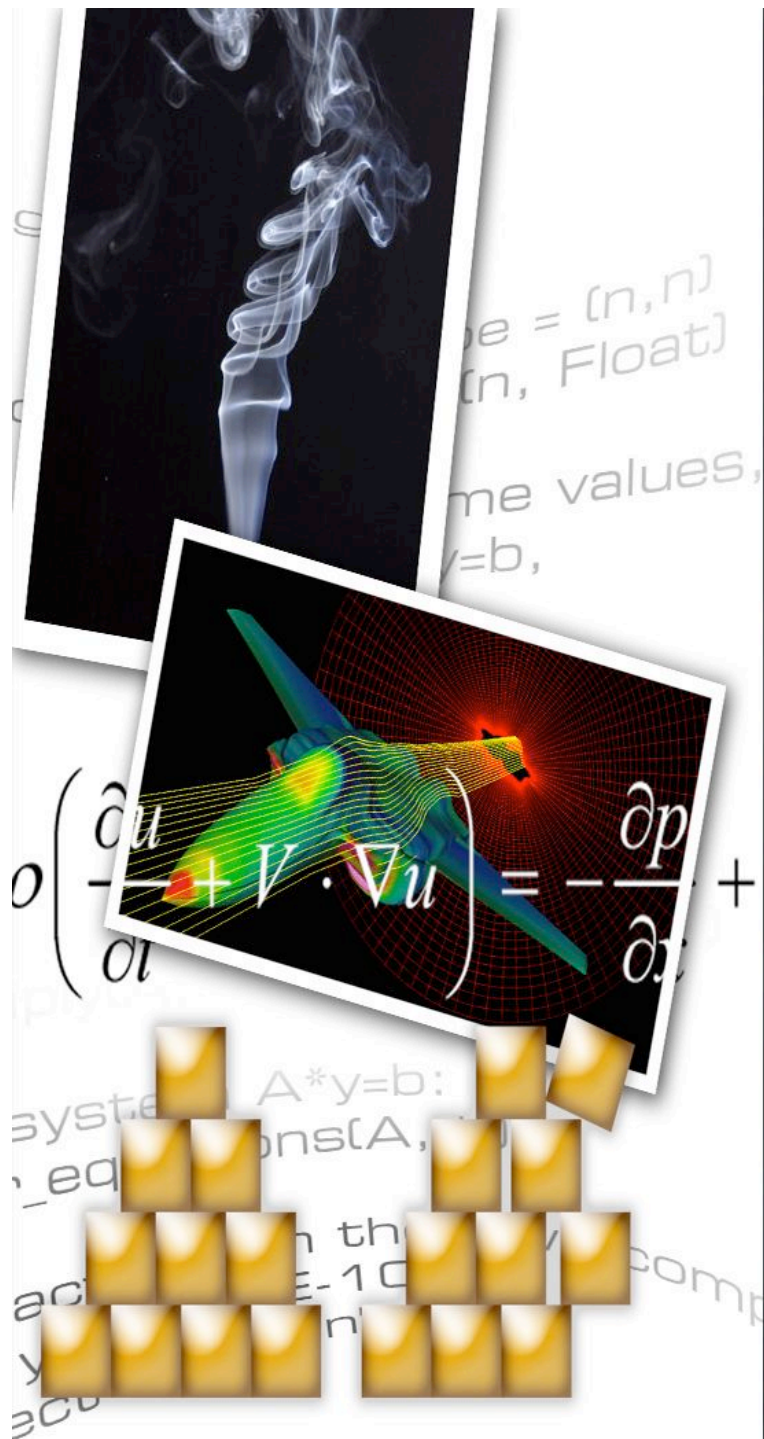
Biomedical flows

- Drug inhalation
- Aerosol flow (bacteria)
- Circle of Willis
- Fluid-structure interactions
- 1D arterial network
- Multiscale 1D-3D blood flow
- Tissue deformation
- Flow around the mitral valve in the heart
- Sound waves in the lungs
- Flow, deformation and electrophysiology in the heart
- Fundamental fluid mechanics research

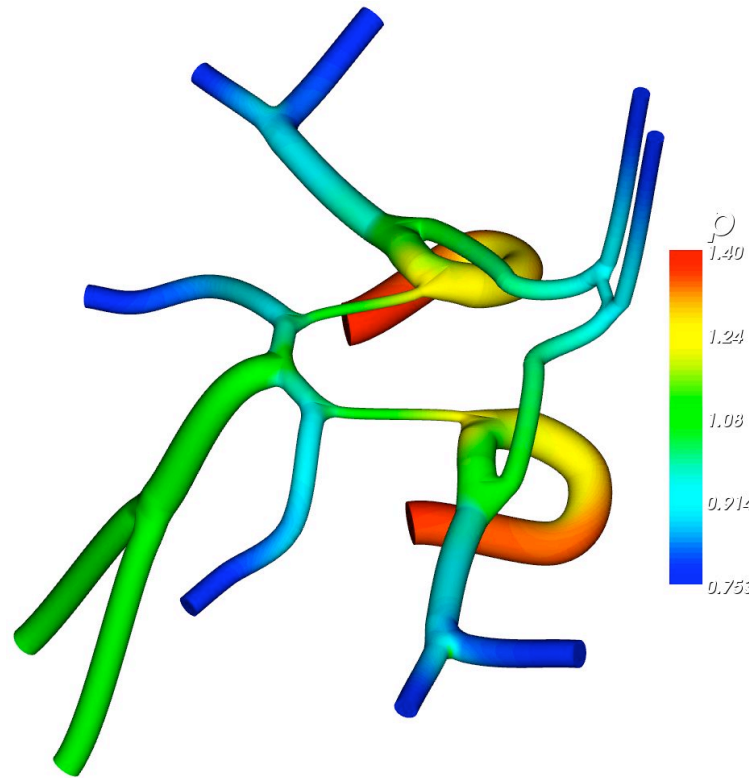


$$\rho \left(\frac{\partial u}{\partial t} + V \cdot \nabla u \right) = - \frac{\partial p}{\partial x} + \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z} -$$



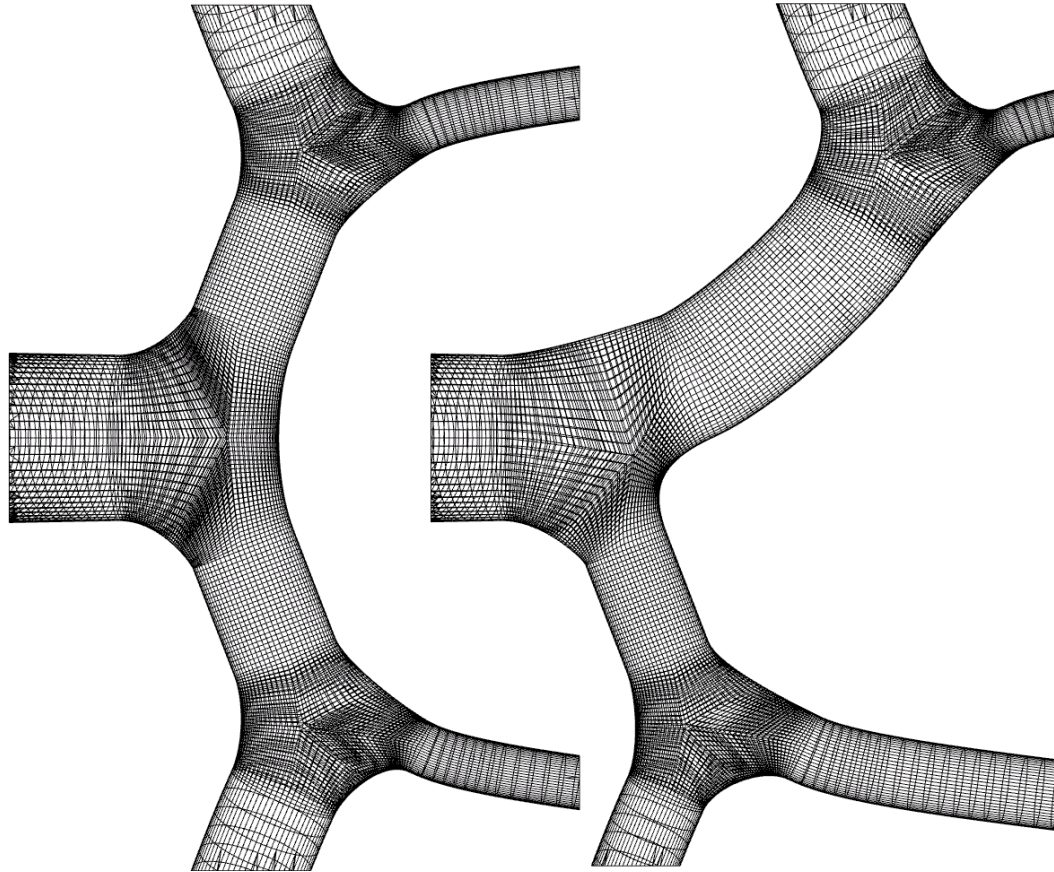


Blood flow is incompressible, with Newtonian behaviour in large vessels



Using prescribed time varying flow rates on inlets, equal pressures on all outlets, and rigid vessel walls

Navier-Stokes equations are solved with the Finite Element Method, using Featflow



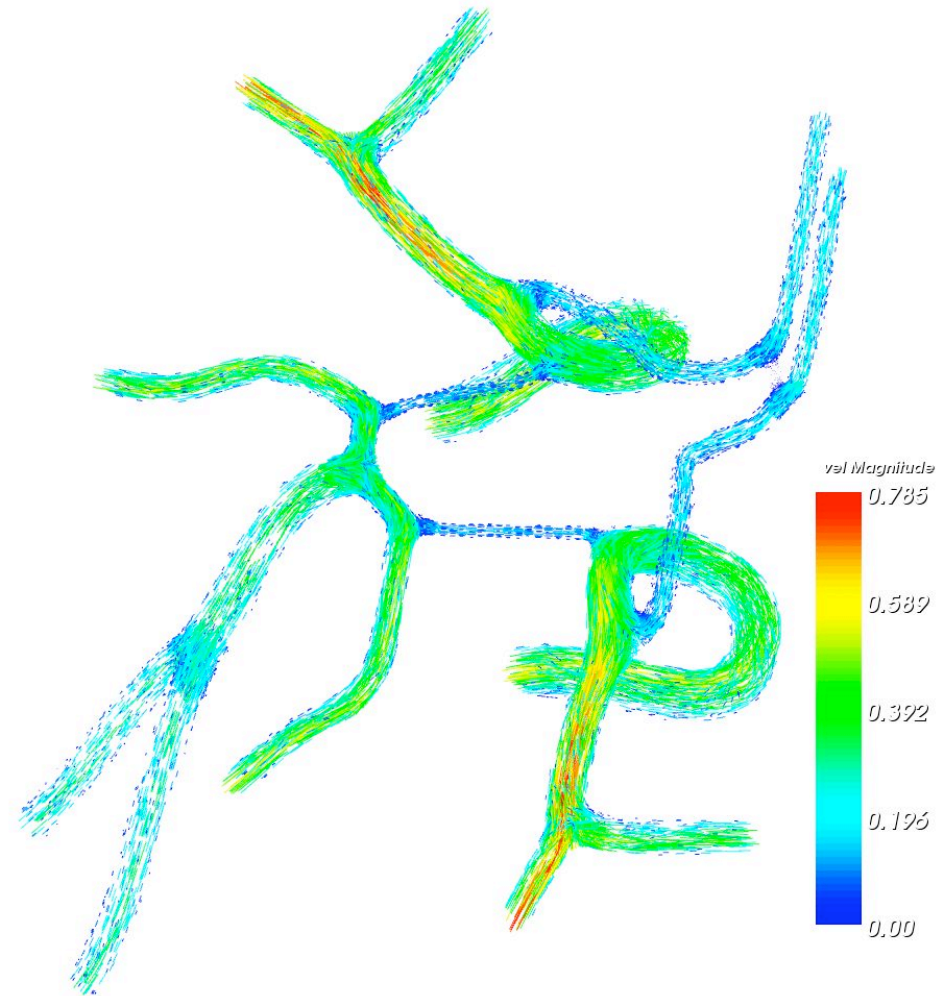
**Grids are created from a parameterization,
using custom written software**

Simulation results on the full CoW geometry are within normal values found in medical studies

Only limited boundary data are available

The geometry is a simplification

Even so, the results seem realistic



Other biomedical applications; cardiac electrophysiology

1. Complete simulator for heart electrophysiology and mechanics
2. Automatic detection of ischemic heart disease based on ECG recordings



Center for Biomedical Computing

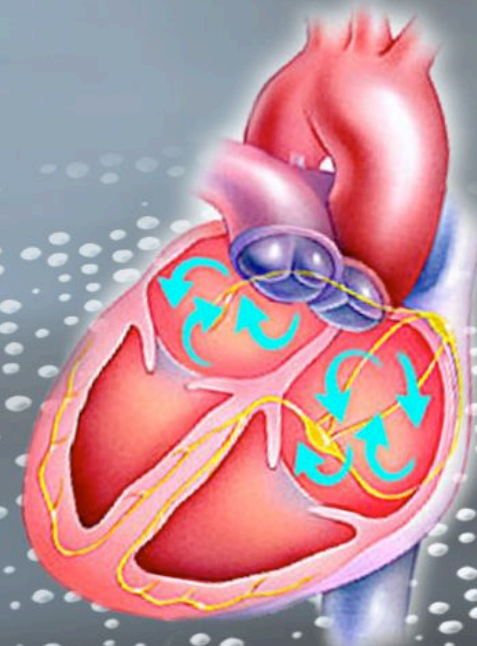
2007 - 2017

```
M = G.compute_mass_matrix(velocity_element)
A = G.compute_stiffness_matrix(velocity_element)
B = G.compute_div_matrix(velocity_element, pressure_element)
D = G.compute_stiffness_matrix(pressure_element)

T = 1; dt = 0.1
while t < T:
    t = t + dt;
    f = G.compute_source_vector(rhs)
    C = G.compute_convection_matrix(velocity_element, v_prev)

    A1 = M + dt*A + dt*C
    prec1 = MLPrec(A1)
    v, iter = preconditionBiCGstab(prec1, A1, v, f, 1.0e-9)

    g = (1.0/dt)*B.t*v
    phi, iter = preconditionConjGrad(MLPrec(D), D, phi, g, 1.0e-9)
    v = v - dt*B.t*phi
    p = (Mp + dt*Ap)phi
```



$$\rho \left(\frac{\partial u}{\partial t} + V \cdot \nabla u \right) = - \frac{\partial \tau_{yx}}{\partial x} - \frac{\partial \tau_{zx}}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z} + \rho f_x$$