1 Incompressible Navier Stokes Equation

1.1 Problem/Strong Form

We want to find

$$(u, p) : \Omega \times (0, T) \to \mathbb{R}^d \times \mathbb{R}$$

where u is unknown velocity and p is unknown pressure such that

$$\begin{cases} \dfrac{\partial u}{\partial t} + (u \cdot \nabla)u - \nu \triangle u + \nabla p = f & \text{in } \Omega \times (0, T) \\ \nabla \cdot u = 0 & \text{in } \Omega \times (0, T) \\ u = 0 & \text{on } \partial\Omega \times (0, T) \\ u = u^0 & \text{in } \Omega, \text{ at } t = 0 \end{cases} \tag{1}$$

where $f : \Omega \times (0, T) \to \mathbb{R}^d$ and $u^0 : \Omega \to \mathbb{R}^d$ are given functions, $\nu > 0$ is a viscosity.

1.2 Weak Form

The weak formulation for equation (1) is shown below. We want to find $\{(u, p)(t) \in V \times Q; t \in (0.T)\}$ such that for $t \in (0, T)$

$$\begin{cases} (\dfrac{\partial u}{\partial t} + (u \cdot \nabla)u, v) + a(u, v) + b(v, p) + b(u, q) = (f, v) & , \forall (v, q) \in V \times Q \\ u = u^0 & , t = 0 \end{cases} \tag{2}$$

where

$$\begin{aligned} a(u, v) &= \nu \int_\Omega \nabla u : \nabla v \; dx \\ b(v, q) &= - \int_\Omega (\nabla \cdot v) q \; dx \\ V &= H_0^1(\Omega, \mathbb{R}^d) = H_0^1(\Omega)^d \\ Q &= \{q \in L^2(\Omega); \int_\Omega q \; dx = 0\}. \end{aligned}$$

1.3 Discretization

Before applying to FreeFEM++, we need to discritize $\dfrac{\partial u}{\partial t} + (u \cdot \nabla)u_i$ part, where dt as time increment

- Using convect and first order time discretization, we obtain

$$\frac{\partial u_i}{\partial t} + (u \cdot \nabla)u_i = \frac{u_i^n - u_i^{n-1}(X(u^{n-1}, dt))}{dt}$$

1.4 Error estimate

To estimate the error in 2D, we use L_2-norm

$$\begin{aligned} E(h, dt) &= \|u_h^n - u^n\|_{L^\infty(L^2)} \\ &= max \; \|u_h^n - u^n\|_{L^2(\Omega)}^2 \\ &= max \; \{\|u_{h_1}^n - u_1^n\|_{L^2(\Omega)}^2 + \|u_{h_2}^n - u_2^n\|_{L^2(\Omega)}^2\}^{1/2} \\ &= max \; \{\int_\Omega (u_{h_1}^n - u_1^n)^2 \; dx + \int_\Omega (u_{h_2}^n - u_2^n)^2 \; dx\}^{1/2} \end{aligned}$$

To estimate the error in 3D, we use

$$\begin{aligned} E(h, dt) &= \|u_h^n - u^n\|_{L^\infty(L^2)} \\ &= max \; \|u_h^n - u^n\| \\ &= max \; \{\|u_{h_1}^n - u_1^n\|_{L^2(\Omega)}^2 + \|u_{h_2}^n - u_2^n\|_{L^2(\Omega)}^2 + \|u_{h_3}^n - u_3^n\|_{L^2(\Omega)}^2\}^{1/2} \\ &= max \; \{\int_\Omega (u_{h_1}^n - u_1^n)^2 \; dx + \int_\Omega (u_{h_2}^n - u_2^n)^2 \; dx + \int_\Omega (u_{h_3}^n - u_3^n)^2 \; dx\}^{1/2} \end{aligned}$$

To estimate the error, we also use H-norm

$$E(h, dt) = \|u_h^n - u^n\|_{L^\infty(H^1)} = max \sqrt{\|u_{h_1}^n - u_1^n\|_{L^2(\Omega)}^2 + \|\nabla(u_{h_1}^n - u_1^n)\|_{L^2(\Omega)}^2}$$

1.5 2D Simulation

Below, is the exact solution to check if the program for 2D is working.

$$\begin{aligned} u &= (u_1, u_2) \\ u_1 &= -cos(x_1)sin(x_2)e^{-4t} \\ u_2 &= -sin(x_1)cos(x_2)e^{-4t} \\ p &= \frac{1}{4}(cos(2x_1) + cos(2x_2))e^{-4t} \end{aligned}$$

such that equation (1) is satisfied with $f = (f_1, f_2)$.
With $f_1 = -e^{-4t}sin(2x_1)$ and $f_2 = -e^{-4t}sin(2x_2)$.
with the error estimate :

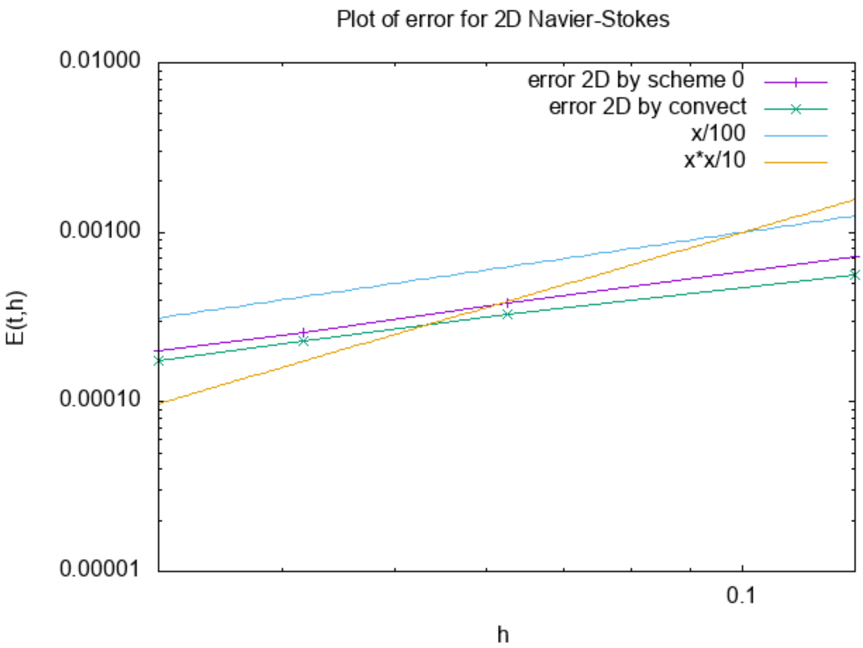

Figure 1:

As we can see, the error using convect is slightly better than scheme 0. By the graphic, we have $O(h)$.

1.6 3D Simulation

Below, is the exact solution to check if the program 3D is working.

$$
\begin{aligned}
u &= (u_1, u_2, u_3) \\
u_1 &= -\cos(x_1)\sin(x_2)\cos(x_3)e^{-2t} \\
u_2 &= -\sin(x_1)\cos(x_2)\cos(x_3)e^{-2t} \\
u_3 &= 0 \\
p &= \frac{1}{4}e^{-4t}(\cos(2x_1) + \cos(2x_2) + \cos(2x_3))
\end{aligned}
$$

such that equation (1) is satisfied with $f = (f_1, f_2, f_3)$. With $f_1 = -\cos(x_1)\sin(x_2)\cos(x_3)e^{-2t}$, $f_2 = -\sin(x_1)\cos(x_2)\cos(x_3)e^{-2t}$, and $f_3 = -(\frac{1}{4})e^{-4t}\sin(2x_3)(2\cos(2x_3)+1)$

With the error estimate 1st order using scheme 0 and 2nd order using Adam-Bashforth

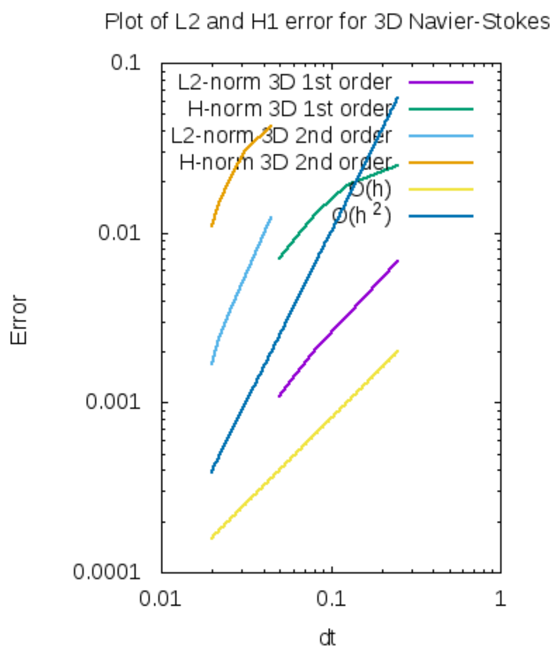

Figure 2:

Below is the FreeFEM++ code used to solve the problem above with the discritization of time using scheme 0 with convect term :

```
load "iovtk"
load "msh3"

// Variable declaration
real nu = 1.0;
real delta=1.0;
real error,Herror;
real errormax = 0, Herrormax = 0;
real t=0;
func exactu1 = -cos(x)*sin(y)*cos(z)*exp(-2*t);
func exactu2 = sin(x)*cos(y)*cos(z)*exp(-2*t);
func exactu3 = 0.;
func dx1 = sin(x)*sin(y)*cos(z)*exp(-2*t);
func dy2 = sin(x)*(-sin(y))*cos(z)*exp(-2*t);
func dz3 = 0.;
func f1 = -cos(x)*sin(y)*cos(z)*exp(-2*t);
func f2 = sin(x)*cos(y)*cos(z)*exp(-2*t);
func f3 = (-exp(-4*t)/4)*sin(2*z)*(2*cos(2*z)+1);

int[int] rup=[0,1], rdown=[0,1], rmid=[1,1,2,1,3,1,4,1];
real zmin=0,zmax=1;

ofstream ff("1error_3D.txt");
ofstream hh("1error_H.txt");

//iteration for each mesh devider
for(int n=24; n>=4; n=n-4 )
{
real dt = 1./n; //take dt=h

// Create the mesh
mesh Th2=square(n,n);
mesh3 Th=buildlayers(Th2,n,
zbound=[zmin,zmax], labelmid=rmid, reffaceup = rup, reffacelow = rdown);
plot ( Th, ps = "NS_3D_mesh_1.ps" );

fespace Uh(Th,[P1,P1,P1,P1]);
fespace Vh(Th,P13d);
macro Grad(u) [dx(u),dy(u),dz(u)]// EOM
macro div(u1,u2,u3) (dx(u1)+dy(u2)+dz(u3)) //EOM
macro L2norm(Th,u,exactu) (int3d(Th)(square(u-exactu))) //EOM

Uh [u1,u2,u3,p];
Uh [v1,v2,v3,q];
Vh u1old, u2old, u3old;

problem navierstokes ([u1,u2,u3,p],[v1,v2,v3,q]) =
int3d(Th) (u1*v1/dt) - int3d(Th) (convect([u1old,u2old,u3old],(-dt),u1old)*v1/dt)
+ int3d(Th) (u2*v2/dt) - int3d(Th) (convect([u1old,u2old,u3old],(-dt),u2old)*v2/dt)
+ int3d(Th) (u3*v3/dt) - int3d(Th) (convect([u1old,u2old,u3old],(-dt),u3old)*v3/dt)
+ int3d(Th,qforder=3)( Grad(u1)'*Grad(v1) + Grad(u2)'*Grad(v2) + Grad(u3)'*Grad(v3) //)'
- div(u1,u2,u3)*q - div(v1,v2,v3)*p)
- int3d(Th) ((f1*v1) + (f2*v2) + (f3*v3))
- int3d(Th) (delta* hTriangle * hTriangle * Grad(p)'*Grad(q))
+ on(1,u1=exactu1,u2=exactu2,u3=exactu3) ;

u1old = -cos(x)*sin(y)*cos(z) ;
u2old = sin(x)*cos(y)*cos(z) ;
u3old = 0;

for ( int it = 1; it <= n; it++ )
```

```
{
t=it*dt;
navierstokes;
error = sqrt( L2norm(Th,u1,exactu1) + L2norm(Th,u2,exactu2) + L2norm(Th,u3,exactu3));
Herror = sqrt( square(error) + L2norm(Th,dx(u1),dx1) + L2norm(Th,dy(u2),dy2) + L2norm(Th,d
if (error > errormax) errormax = error ;
if (Herror > Herrormax) Herrormax = Herror ;
cout << "L2-error at " << t << "is " << error << "max = " << errormax << "\n" ;
cout << "H1-error at " << t << "is " << Herror << "max = " << Herrormax << "\n";
if(n==24){
plot ( Th, [u1,u2,u3], nbiso = 60, fill = 0, value = 1, wait = 0);
savevtk("NS_3D_1_plot"+n+"_"+it+".vtk",Th,[u1,u2,u3],p,dataname="NavSto");
plot (p, nbiso=60, fill =0, value =1, wait =0 );
}
u1old = u1; u2old=u2; u3old=u3;
}
ff << dt << "\t" << errormax << "\n" ;
hh << dt << "\t" << Herrormax << "\n" ;
cout << ">>>>>MESH>>>>> " << n << " executed \n" ;
errormax = 0;
}
//  Terminate.
//
cout << "\n";
cout << "NAVIERSTOKES:\n";
cout << "Normal end of execution.\n";
```

We also try to use discretization of time using Adam-Bashfoth with convect term, the code is :

```
load "iovtk"
load "msh3"

// Variable declaration
real nu = 1.0;
real delta=1.0;
real error, Herror;
real errormax = 0, Herrormax = 0;
real t=0;
func exactu1 = -cos(x)*sin(y)*cos(z)*exp(-2*t);
func exactu2 = sin(x)*cos(y)*cos(z)*exp(-2*t);
func exactu3 = 0.;
func dx1 = sin(x)*sin(y)*cos(z)*exp(-2*t);
func dy2 = sin(x)*(-sin(y))*cos(z)*exp(-2*t);
func dz3 = 0.;
func f1 = -cos(x)*sin(y)*cos(z)*exp(-2*t);
func f2 = sin(x)*cos(y)*cos(z)*exp(-2*t);
func f3 = (-exp(-4*t)/4)*sin(2*z)*(2*cos(2*z)+1);

int[int] rup=[0,1], rdown=[0,1], rmid=[1,1,2,1,3,1,4,1];
real zmin=0,zmax=1;

ofstream ff("2error_3D.txt");
ofstream hh("2error_H.txt");

//iteration for each mesh devider
for(int n=24; n>=4; n=n-4 )
{
real dt = 1./n; //take dt=h

// Create the mesh
mesh Th2=square(n,n);
mesh3 Th=buildlayers(Th2,n,
zbound=[zmin,zmax], labelmid=rmid, reffaceup = rup, reffacelow = rdown);
plot ( Th, ps = "NS_3D_mesh_2.ps" );

fespace Uh(Th,[P1,P1,P1,P1]);
fespace Vh(Th,P13d);
macro Grad(u) [dx(u),dy(u),dz(u)]// EOM
macro div(u1,u2,u3) (dx(u1)+dy(u2)+dz(u3)) //EOM
macro L2norm(Th,u,exactu) (int3d(Th)(square(u-exactu))) //EOM

Uh [u1,u2,u3,p];
Uh [v1,v2,v3,q];
Vh u1old, u2old, u3old;
Vh u1oldd, u2oldd, u3oldd;
Vh u1star,u2star,u3star;

problem navierstokesinit ([u1,u2,u3,p],[v1,v2,v3,q]) =
int3d(Th) (u1*v1/dt) - int3d(Th) (convect([u1old,u2old,u3old],(-dt),u1old)*v1/dt)
+ int3d(Th) (u2*v2/dt) - int3d(Th) (convect([u1old,u2old,u3old],(-dt),u2old)*v2/dt)
+ int3d(Th) (u3*v3/dt) - int3d(Th) (convect([u1old,u2old,u3old],(-dt),u3old)*v3/dt)
+ int3d(Th,qforder=3)( Grad(u1)'*Grad(v1) + Grad(u2)'*Grad(v2) + Grad(u3)'*Grad(v3) //)'
- div(u1,u2,u3)*q - div(v1,v2,v3)*p)
- int3d(Th) ((f1*v1) + (f2*v2) + (f3*v3))
- int3d(Th) (delta* hTriangle * hTriangle * Grad(p)'*Grad(q))
+ on(1,u1=exactu1,u2=exactu2,u3=exactu3) ;

problem navierstokes ([u1,u2,u3,p],[v1,v2,v3,q]) =
int3d(Th) (3*u1*v1/dt) - int3d(Th) (convect([u1star,u2star,u3star],(-dt),u1old)*4*v1/dt)
+ int3d(Th) (convect([u1star,u2star,u3star],(-2*dt),u1oldd)*v1/dt)
+ int3d(Th) (3*u2*v2/dt) - int3d(Th) (convect([u1star,u2star,u3star],(-dt),u2old)*4*v2/dt)
```

```
+ int3d(Th) (convect([u1star,u2star,u3star],(-2*dt),u2oldd)*v2/dt)
+ int3d(Th) (3*u3*v3/dt) - int3d(Th) (convect([u1star,u2star,u3star],(-dt),u3old)*4*v3/dt)
+ int3d(Th) (convect([u1star,u2star,u3star],(-2*dt),u3oldd)*v3/dt)
+ int3d(Th,qforder=3)( Grad(u1)'*Grad(v1) +  Grad(u2)'*Grad(v2) +  Grad(u3)'*Grad(v3) //)'
- div(u1,u2,u3)*q - div(v1,v2,v3)*p)
- int3d(Th) ((f1*v1) + (f2*v2) + (f3*v3))
- int3d(Th) (delta* hTriangle * hTriangle * Grad(p)'*Grad(q))
+ on(1,u1=exactu1,u2=exactu2,u3=exactu3) ;

u1old = -cos(x)*sin(y)*cos(z) ;
u2old = sin(x)*cos(y)*cos(z) ;
u3old = 0;

for ( int it = 1; it <= n; it++ )
{
t=it*dt;
if (it==1) {navierstokesinit;}
else {
navierstokes;
error = sqrt( L2norm(Th,u1,exactu1) + L2norm(Th,u2,exactu2) + L2norm(Th,u3,exactu3));
Herror = sqrt( square(error) + L2norm(Th,dx(u1),dx1) + L2norm(Th,dy(u2),dy2) + L2norm(Th,d
if (error > errormax) errormax = error ;
if (Herror > Herrormax) Herrormax = Herror ;
cout << "L2-error at " << t << "is " << error << "max = " << errormax << "\n" ;
cout << "H1-error at " << t << "is " << Herror << "max = " << Herrormax << "\n";
}
if(n==24){
plot ( Th, [u1,u2,u3], nbiso = 60, fill = 0, value = 1, wait = 0);
savevtk("NS_3D_2_plot"+n+"_"+it+".vtk",Th,[u1,u2,u3],p,dataname="NavSto");
plot (p, nbiso=60, fill =0, value =1, wait =0 );
}
u1oldd = u1old; u2oldd = u2old; u3oldd = u3old;
u1old = u1; u2old=u2; u3old=u3;
u1star = 2*u1old-u1oldd; u2star = 2*u2old-u2oldd; u3star = 2*u3old-u3oldd;
}
ff << dt << "\t" << errormax << "\n" ;
hh << dt << "\t" << Herrormax << "\n" ;
cout << ">>>>>MESH>>>>> " << n << " executed \n" ;
errormax = 0;
}
//  Terminate.
//
cout << "\n";
cout << "NAVIERSTOKES:\n";
cout << "Normal end of execution.\n";
```