Inexact and Jacobian Free Newton Krylov method applied to incompressible flow solver

Trung Le

June 29, 2020

Abstract

It has been long known that implicit momentum solver is important for fractional step method of solving incompressible Navier-Stokes equation. In this method, attaining convergence solution with large time step requires the estimation of intermediate flux V^* drops down at least five order of magnitude. There were several methods proposed to solve this problem. However performance of those methods deteriorates as the mesh size decreases.
In this paper, we seek a good a preconditioner for the implicit momentum solver for Jacobian Free Newton Krylov method. This method does not requires explicit evaluation of the Jacobian thus it is suitable for fine mesh simulation. Result shows that good preconditioner is critical for attaining good convergence properties.
[1]

1 Introduction

In many applications in civil engineering, mechanical or aeronautical applications incompressibility of fluid flow is important phenomenon that needs to be respected in low speed regime. This characteristic plays a key role in controlling the governing equations which subsequently imposes difficulty in numerical solvers as it requires the speed of sound in the media goes to infinity.
In order to calculate the flow field several methods to solve Navier-Stokes equations have been proposed. They are different in details but all share the same principle that to find approximation for the momentum equation and solve the Poisson equation in some ways. In this project, one variant of fractional step method originally proposed by Chorin (1965) and extended by Kim and Moin (1986) [] is examined.

2 Problem statement

In this session we will use heavily tensor notation to faciliate the formulation of the problem in generalized coordinate. The reason for using this type of equation is to solve the problem in meshes that follow the body-shape or body-fitted grid. This type of grid gets an excellent quality near the boundary at the expense of using excessive abstract notation. However, despite of its complexity it will be pointed out that it is equivalent to Cartesian grid in any formulations.

In order to start let us consider a curvilinear coordinate where the unit vector changes its direction and magnitude at every point in the space, the partial transformation of Navier Stokes equation reads:

$$J\frac{\partial}{\partial\xi^i}(\frac{U^i}{J}) = 0 \quad (1)$$

$$(\frac{\partial u_i}{\partial t})_r + C(u_i) + G_i(p) - \frac{1}{Re}D(u_i) = 0 \quad (2)$$

In this formulation, the contravariant flux U^i is used in conjunction with Cartesian component u_i. The main idea is to utilize the superior property for mass conservation of contravariant mass flux while minimizing the effort to solve the momentum equation without it. C,G and D are convective, gradient and viscous terms respectively. They are defined:

$$C(.) = J\frac{\partial}{\partial\xi^j}(\frac{U^j}{J} - \frac{V^j}{J}).+\frac{U^j}{J}w_i \quad (3)$$

$$G_i(.) = J\frac{\partial}{\partial\xi^j}(\frac{\xi^j_{x_i}}{J}.) \quad (4)$$

$$D(.) = J\frac{\partial}{\xi^j}(\frac{g^{ik}}{J}\frac{\partial}{\partial\xi^k}.) \quad (5)$$

Note that J is the Jacobian of the transformation from Cartesian coordinate to the curvilinear system. g^{ij} is the contravariant metric tensor which define the inner product i.e length in curvilinear system ξ.

$$J = \partial(\xi^1,\xi^2,\xi^3)/\partial(x_1,x_2,x_3) \quad (6)$$

$$g^{jk} = \xi^j_{x_i}\xi^k_{x_i}. \quad (7)$$

$$U^i = u_j\xi^i_{xj} \quad (8)$$

$$u_i = U^j\frac{\partial x_i}{\partial\xi^j} \quad (9)$$

$$V^i = v_j\xi^i_{xj} \quad (10)$$

$$v_i = V^j\frac{\partial x_i}{\partial\xi^j} \quad (11)$$

$$(12)$$

We will discuss the equivalent version of contravariant flux formulation discussed above in Cartesian coordinate context. Furthermore we will explore the appropriate numerical schemes used for this equation.

3 Numerical schemes

In this paper, for the sake of simplicity of algorithm discussion we will use Cartersian coordinates. Thus Navier-Stokes equation is simply:

$$\frac{\partial u_i}{\partial x_i} = 0 \tag{13}$$

$$\frac{\partial u_i}{\partial t} + \frac{\partial (u_i u_j)}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{1}{Re}\frac{\partial u_i}{\partial x_j \partial x_j} \tag{14}$$

The first equation is the result of incompressibility constraint $\nabla \cdot \overrightarrow{u} = 0$. The second equation is Newton second law applied for a control volume of the continua.
Now let us consider the numerical schemes that are used to solved this system. We will consider two problems: variables arrangements and numerical methods.

At first, the variables arrangement should be considered. It is natural to allocate all the variables (i.e velocity components u_i and pressure p) at one location - at grid node, for example. This approach is call non-stagger approach. This approach has been shown that satisfying the incompressibility is difficult. The reason for this fact is because of the decoupling of the pressure of the momentum equation from the continuity.
The other approach is to allocate the velocity component at the surface of the control volume (i.e one grid cell) and the pressure at the center of the control volume. This approach easily reaches the mass conservation because it is easy to verify that summation of all fluxes over the discrete domain equals to the summation over the boundary of that domain. It means that mass conservation is automatically satisfied. This approach is equivalent to finite volume method, in fact.

Second, let us consider the numerical method. One numerical procedure to solve the above system is to find the solution as projection method or Fractional Step Method. The main idea is first to seek an approximated solution for the momentum equation independently of mass conservation constraint. The subsequent step is to project that approximated solution in the space of divergence free functional. If the second order time accurate scheme is used, the first step (or momentum step) can be described as:

$$\frac{\partial u_i}{\partial t} = -\frac{\partial (u_i u_j)}{\partial x_j} - \frac{\partial p}{\partial x_i} + \frac{1}{Re}\frac{\partial u_i}{\partial x_j \partial x_j} = RHS(u_i) \tag{15}$$

$$\tag{16}$$

In Cartesian coordinate the flux at the surface V^i is identical to the velocity component u_i. Thus we can write this equation in semi-discrete form with u and V represent velocity and flux vectors respectively. We seek for V^* - the intermediate solution for the momentum

$$\frac{\partial u}{\partial t} \approx \frac{3V^* - 4V^n + V^{n-1}}{2\triangle t} = RHS(V^*) \tag{17}$$

$$\tag{18}$$

Take the divergence of the above equation

$$\triangle\phi = \frac{3}{2\triangle t}\nabla \cdot V^* \tag{19}$$

$$\tag{20}$$

and finally the update of V^* is necessary for the next time step using gradient of the pressure ∇p.:

$$V^{n+1} = V^* - \frac{2\triangle t}{3}\nabla\phi \tag{21}$$

$$p = p^n + \phi \tag{22}$$

$$\tag{23}$$

As mentioned above stagger mesh has superior quality for mass conservation. However imposing boundary conditions for pure stagger mesh is cumbersome due to specifying the velocity at the cell surface on the boundaries and especially the pressure. In recent years hybrid stagger and non-stagger mesh is becoming more and more popular []. The main theme is to use both stagger and non-stagger meshes at the same time and interpolate back and forth between this two mesh. On non-stagger mesh, ones can impose boundary conditions and calculate all the terms of the right hand side in momentum equation. After that, stagger mesh is reconstructed from non-stagger one. Poisson equation is solved on the stagger mesh to ensure the incompressibility. Finally, non-stagger mesh is updated from the stagger one and advance the solution.

```
Algorithm:
   for time = 1 :MAXTIME
         Find V* flux in stagger mesh
         Solve Poisson equation
         Update solution for mass fluxes
         Convert to Cartesian component
   end
```

For the solver parts, the choice of numerical methods for Poisson equation is now well established with preconditioned conjugate gradient method. Multigrid is excellent candidate for the preconditioner due to the symmetry of elliptic operator. In this code, Poisson equation is solved either with gmres or "backslash" from MATLAB. The remaining difficulty is to solve the non-linear equation 17 with iterative methods. These methods are fix point iteration, Newton-Krylov methods and ADI schemes. I focus on Newton-Krylov methods. Fix point iteration is only used to compared with Newton-Krylov in test cases.

4 Matlab implementations

The code is straight-forward implementation of the algorithm. In the following piece of codes, all the essential components are presented. Momentum_Solver function solves (17) while Poisson_Solver solves (19). FormBCS is to do forming

boundary conditions on the Cartesian grid. Divergence is used to estimate the residual of Poisson solver. Ucont, Ucat denotes contravariant flux and Cartesian components with respect to x and y.

```
for time_step = 1:MAXTIME

 t = time_step * dt

 [U_im_x U_im_y] =  Momentum_Solver(dU_x , dU_y, Ucont_x, Ucont_y,Ucat_x,....
                    Ucat_y, Ubcs_x, Ubcs_y, Pressure, Re, dx, dy, dt,t);

 [phi]  = Poisson_Solver(U_im_x, U_im_y, dx, dy, dt);

 [U_x_new U_y_new P_new] = Update_Solution(U_im_x, U_im_y, Pressure,...
                                           phi,dx,dy,dt);

 %Update dU
  dU_x = U_x_new - Ucont_x;
  dU_y = U_y_new - Ucont_y;

 % Go next time step
 [Ucat_x Ucat_y Ucont_x Ucont_y] = FormBCS(U_x_new, U_y_new,..
                                   Ubcs_x,Ubcs_y, dx, dy,Re,t);

 Pressure = P_new;
 % Assess the divergence of flow field
 Div = Divergence(U_x_new, U_y_new,dx ,dy);
end
```

Spatial discretization is finite diffecing with QUICK scheme [] for convective term, central difference for all other terms. Time derivative is approximated by second order backward scheme. The following code is an example of auxiliary Fpx1 of QUICK scheme for convective term in the x direction uu.

```
         ucon = Ucont_x(i,j) /  2;
         up = ucon + abs(ucon);
         um = ucon - abs(ucon);

      if (i > 1 && i < M-1)
            Fpx1(i,j) =  um * ( coef * ( - Ucat_x(i+2,j) -...
             2 * Ucat_x(i+1,j) + 3 * Ucat_x(i,j)  )...
             + Ucat_x(i+1,j)) +  up * ( coef * ( - Ucat_x(i-1,j) - ....
             2 * Ucat_x(i,j)   + 3 * Ucat_x(i+1,j)) + Ucat_x(i,j));
      else
          if (i == 1)
              Fpx1(i,j) =um * ( coef * ( - Ucat_x(i+2,j) -...
               2 * Ucat_x(i+1,j) + 3 * Ucat_x(i,j)  )...
               + Ucat_x(i+1,j)) +  up * ( coef * ( - Ucat_x(i,j)  -...
                2 * Ucat_x(i,j)   + 3 * Ucat_x(i+1,j)) + Ucat_x(i,j));
```

```
          else
                 if (i == M-1)
                    Fpx1(i,j) =  um * ( coef * ( - Ucat_x(i+1,j) -...
                     2 * Ucat_x(i+1,j) + 3 * Ucat_x(i,j))...
                     + Ucat_x(i+1,j)) + up * ( coef * ( - Ucat_x(i-1,j) -...
                      2 * Ucat_x(i,j)   + 3 * Ucat_x(i+1,j)) + Ucat_x(i,j));
                 end
          end
       end
     end
end
```

5 Test Cases

For the incompressible flow solver, Taylor-Green vortex and driven cavity problem are two standard test cases. We test our code performance on these two cases. The Taylor-Green vortex in Figure 1 is defined:

$$U = -cos(x) \cdot sin(y) \cdot exp(-2 \cdot t) \quad (24)$$

$$V = -cos(x) \cdot sin(y) \cdot exp(-2 \cdot t) \quad (25)$$

$$P = -\frac{cos(2x) + cos(2y)}{4} \quad (26)$$

$$(27)$$

Driven cavity problem is a unit square cavity with a continuous driven lid on the top with velocity of U = 1 as shown in Figure 2.

6 Fix point iteration

For comparison purpose, fix point iteration is implemented using Runge-Kutta fourth order method. The iteration is of the form:

$$x_l = x_n - \alpha_i \cdot \triangle\tau \cdot F(x) \quad (28)$$

$$x_{n+1} = x_4 \quad (29)$$

$$i = 1, 2, 3, 4 \quad (30)$$

$$(31)$$

Its pseudocode is written as follows:

```
while (iter < MAXITERS && ||e|| > tol)

    U_im = U_p;

    for stage=1:4
         [RHS] = RHS_Calculation(U_im);
```

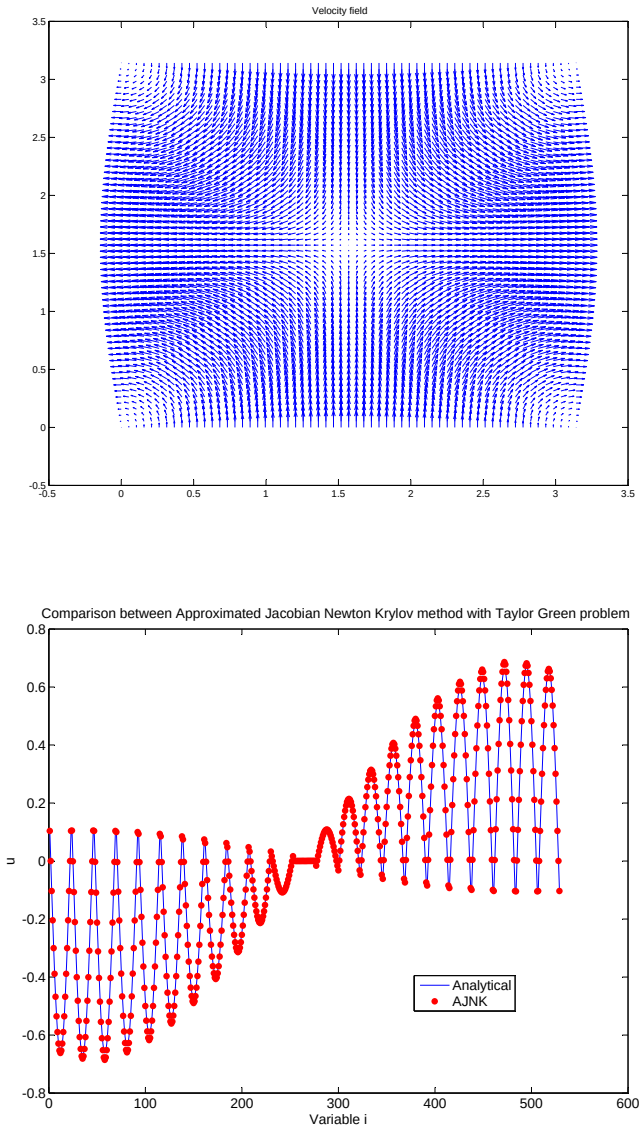
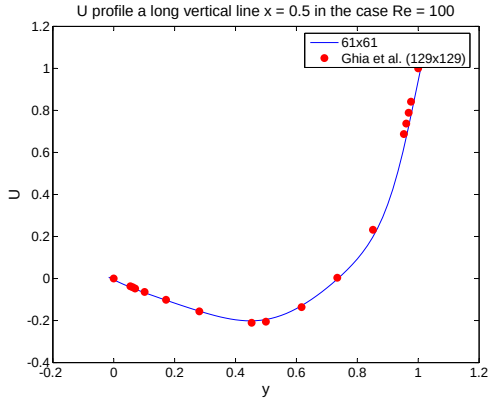


Figure 1: Taylor Green vortex comparison at t = 0.2



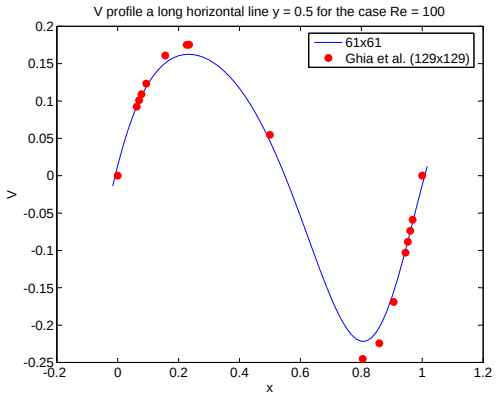


Figure 2: Driven cavity with Re = 100

```
        U_im = U_p + alpha(stage) * dt * RHS;
        [Ucat] = FormBCS(U_im);
    end

    e = norm(U_p - U_im,inf);
    U_p = U_im;
    iter++;
end
```

It takes about 16 iterations for fix point iteration to drop 5 order of magnitude:

```
subitr = 1, Momentum convergence e = 4.278896\\
subitr = 2, Momentum convergence e = 0.440627\\
subitr = 3, Momentum convergence e = 0.189015\\
subitr = 4, Momentum convergence e = 0.105856\\
subitr = 5, Momentum convergence e = 0.053833\\
subitr = 6, Momentum convergence e = 0.025163\\
subitr = 7, Momentum convergence e = 0.013426\\
subitr = 8, Momentum convergence e = 0.007951\\
subitr = 9, Momentum convergence e = 0.004445\\
subitr = 10, Momentum convergence e = 0.002197\\
subitr = 11, Momentum convergence e = 0.001113\\
subitr = 12, Momentum convergence e = 0.000650\\
subitr = 13, Momentum convergence e = 0.000363\\
subitr = 14, Momentum convergence e = 0.000184\\
subitr = 15, Momentum convergence e = 0.000101\\
subitr = 16, Momentum convergence e = 0.000058\\
```

7 Newton-Krylov methods

As discussed above, in the momentum step (17) we solve the nonlinear system with the flux V^*

$$F(V^*) = V^* - RHS(V^*) = 0 \tag{32}$$

$$\tag{33}$$

There are several ways to solve this equation by iterative techniques including: fix point iteration and Newton methods. It is well known that fix point iteration has linear convergence rate and $\triangle t$ is required to be small. Newton methods are promising because it has quadratic convergence rate and allowing large $\triangle t$ which is important in real life applications. However, the convergence of Newton methods are not globally guaranteed and depends strongly on the initial guess. This characteristic has been proposed to be solved using dual problem. However, it is not the focus of this project which uses only original Newton method. The line search techniques and trust regions can be added as the code evolves over larger amount of time.

In this project, both fix point iteration and Newton methods are implemented. The Jacobian of the right hand side is either approximated by Frechet derivative or "analytical" derivation. Let us review Newton algorithm in following pseudo codes:

```
set x0, r0 = F(x0)

while F(x) > tol
```

$$J = \frac{\partial F}{\partial x} \tag{34}$$

$$J\delta = -F \tag{35}$$

$$x = x_0 + \delta \tag{36}$$

```
end do
```

Assuming we know Jacobian J we can use a Krylov method such as BiCGSTABB or GMRES to solve the system to find δ and advance the search. There are several issues with this original algorithm. The first is initial guess x_0 which dictates the convergence. However, in time-dependent problems with the $\triangle t$ not so large x_0 is reasonably set as previous time step. The second problem is the evaluation of the Jacobian J. The virtue of Newton method is that it does not requires J to be exact. In other words, J can be approximated via several ways as discuss below. In large system (i.e millions of unknowns) cost of Jacobian evaluation is extremely expensive to compute fully and it is done only when it is necessary to do so. The last issue is solving large linear system $J\delta = -F$. In fine mesh simulation the condition number of J becomes extremely large and the cost of solving linear system dominates Newton iteration. We will tackle these issues one by one.

7.1 Approximation of Jacobian

The approximated Jacobian can be estimated for small system using Frechet's derivative:

$$J(u)v = \frac{F(u + \epsilon v) - F(u)}{\epsilon} \tag{37}$$

As suggested [] the machine zero can be specified as ϵ. In this work, $\epsilon = 1e - 9$. Using above-mentioned consideration the Matlab code for Frechet derivative is as follows:

```
function F_v = Frechet_v(F,u, v)
n = length(v);

P   = feval(F,u);
epsilon = 1e-9;

P_1 = feval(F,u+epsilon*v);
F_v = (P_1 - P) / epsilon;
```

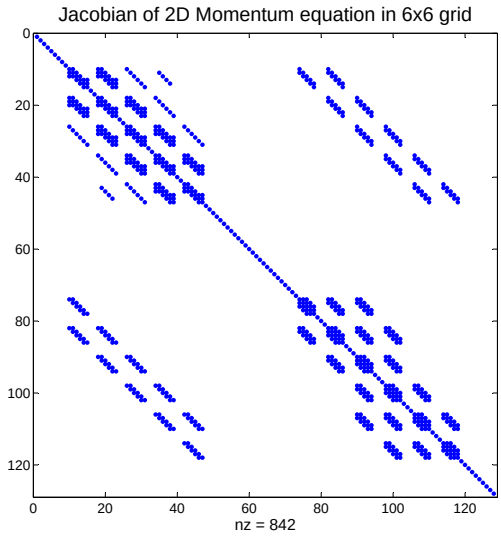
It should be pointed out that direction v in the evaluation can simply be chosen as unit vector e_i of dimension N of problem. Thus as the size N of the problem increases it is extremely expensive to fully form the whole matrix J. Practically N should be less than 200.
Though it has limitation but this automatic approximation has its own value. In fact, it provides an excellent estimation of Jacobian in small problems as shown in Figure 3 which can be served as comparison for "hand" calculation of Jacobian discussed in the next section.

The second fact is that Krylov method does not need an explicit formation of J. Alnordi process needs only formation of operator A as Av which we can substitute by Jv using (37).

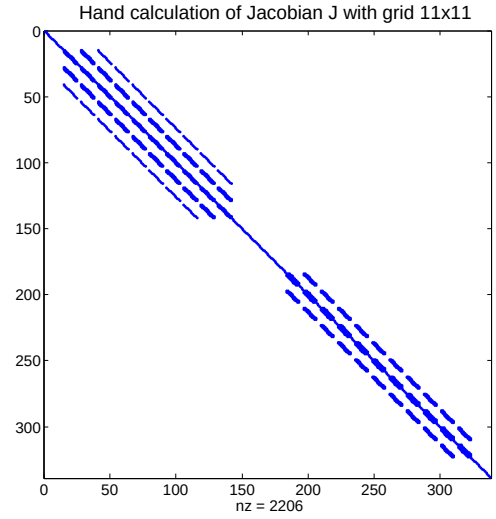
```
%Preconditioning
 z = feval(precfun,PRE,vv(:,i));
 Z(:,i) = z;

%Modified GS
 vv(1:n,i1) = JFNK_Frechet_v(myfunc,sol,z);
 for j=1:i
      t = vv(1:n,j)'*vv(1:n,i1);
      hh(j,i) = t ;
vv(1:n,i1) = vv(1:n,i1) - t*vv(1:n,j);
 end  ;
 t = norm(vv(1:n,i1),2);
 hh(i1,i) = t;
 if (t  ~= 0.0)
   t = 1.0 / t;
   vv(1:n,i1) = vv(1:n,i1)*t;
 end
```

Let us focus on approximation of J using "hand" estimation. As suggested in [] the Jacobian J can be approximated with a lower order method. We approximate J using central differencing scheme for all terms. The resulted matrix is heptadiagonal in Figure 3. The following codes demonstrates assemble of the viscous term:

```
Viscous_coeff_x = dD_dui(i,j,M,N,Re,dx,dy,0);
Viscous_coeff_y = dD_dui(i,j,M,N,Re,dx,dy,1);

%% ------------ Diagonal \partial F \partial u_i
counter= counter + 1;
ii(counter) = index_P_U;
jj(counter) = index_P_U;

% Viscous contribution
if (j ~= 1 && j~=N && i ~=1 && i ~=M && i~=M-1)
    vals(counter) = Viscous_coeff_x(5);
else
```



(a)



(b)

Figure 3: Automated calculation of Jacobian using Frechet derivative and hand calculation

```
        vals(counter) = 0;
    end

    %P
    counter = counter + 1;
    ii(counter) = index_P_V;
    jj(counter) = index_P_V;

    % Viscous contribution
    if (i ~= 1 && i~=M && j ~=1 && j~=N && j~=N-1)
        vals(counter) = Viscous_coeff_y(5);
    else
        vals(counter) = 0;
    end
```

Note that forming the full "hand" calculation of Jacobian is doable but still not practical since it has many off diagonals. In subsequent calculation we will use only the diagonal terms of the Jacobian. If necessary we use extra terms on the off-diagonals but we keep it minimum as possible.

7.2 Inexact and Jacobian Free Newton Krylov

As discussed above, finding exact J is possible but expensive. The remedy is to use some simpler realization of J such as diagonal of J to advance the search. Moreover, as recommended by [] J needs not to be computed very iteration. In fact, it can be delayed and needs to be updated after p iterations. So combining these two simplifications we can comprise an efficient inexact Newton Krylov method.

```
for  iter = 1: maxits

e   = feval(myfunc,x);
Newton_Krylov_norm = norm(e,inf);

if (Newton_Krylov_norm < tol*r0 || iter == maxits)
    its = iter;
    break;
 else

 if (mod(iter,p) == 0 || iter == 1)
  J_b = Jacobian_Assemble(x);
 end
 A   = J_b;
 rhs = -e;

 % Solve to advance the solution
 d_x = fgmres (A,PRE,precfun,rhs,initial_guess,im,max_subit,tolIts);

 [delta_U delta_V] = Un_Vectorize(d_x,M,N);
```

```
   % Advance the solution
   x.U_im_x   = x.U_im_x + delta_U;
   x.U_im_y   = x.U_im_y + delta_V;
  end
end
```

The performance of above algorithm in the case driven cavity 61x61 grid, Re = 100 is as follows:

```
subiter=1 - NK norm = 31.852136
  Newton with approximated J - No Precond
 its 0  res 2.699947e+002
 its 1  res 6.898686e-001
 its 2  res 7.101983e-003
 its 3  res 5.206366e-005
subiter=2 - NK norm = 7.287169
  its 0  res 7.656502e+001
 its 1  res 1.519839e-001
 its 2  res 1.720584e-003
 its 3  res 1.231654e-005
subiter=3 - NK norm = 4.038540
  its 0  res 3.549784e+001
 its 1  res 6.343719e-002
 its 2  res 7.059858e-004
 its 3  res 5.827773e-006
subiter=4 - NK norm = 2.621503
  its 0  res 2.016856e+001
 its 1  res 5.188286e-002
 its 2  res 5.555669e-004
 its 3  res 3.817457e-006
subiter=5 - NK norm = 1.032451
  its 0  res 7.640492e+000
 its 1  res 2.171978e-002
 its 2  res 2.060061e-004
 ........................
```

This method requires about 5 iterations to drops 2 order of magnitude of the residual. GMRES requires about 4 iterations to drop 7 order of magnitudes. The limitation of inexact method is required small time step. Because when $\triangle t$ is large the search might fail to converge to the correct solution. We will try to remedy this using the next method.

As discussed in previous section, Krylov methods do not require explicit evaluation of J so that we can embedded J in the Arnoldi procedure. This method is call Jacobian Free Newton Krylov (JFNK) since the Jacobian is not formed. JFNK is tested in the case of driven cavity with 61x61 grid, Re = 100:

```
subiter= 1 - NK norm = 72.000000
 JFNK with no precond
```

$\triangle t$=0.05	41x41	61x61	101x101	121x121
Fix point iteration	0.85s	Blow up	Blow up	Blow up
Inexact Newton	3.57s	10.8s	102 s	271s
JFNK	4.9s	28.9s	206 s	547s

Table 1: Convergence of different methods, model problem: driven cavity Re = 100, tol = 1e-5

```
 its 0  res 5.577096e+002
 its 1  res 7.667841e+001
 its 2  res 1.260541e+001
 its 3  res 2.094594e+000
 its 4  res 3.483406e-001
 its 5  res 5.794775e-002
 its 6  res 9.643846e-003
 its 7  res 1.605130e-003

subiter= 2 - NK norm = 2.441997
  its 0  res 3.390851e+000
 its 1  res 5.072189e-001
 its 2  res 9.276402e-002
 its 3  res 1.847684e-002
 its 4  res 3.716629e-003
 its 5  res 7.507519e-004
 its 6  res 1.524511e-004
 its 7  res 3.103756e-005
```

This method allows large $\triangle t$. However, if the grid is continuously refined the cost of solving linear system increases rapidly. Its cost supersedes the cost of forming Jacobian as shown in Table .

8 Preconditioning

For inexact Newton method the Jacobian is an approximated one. Thus J is quite well conditioned

9 Conclusions

Fix point iteration is constrained by small time step Newton method can run with larger time-step Preconditioning plays key improvement in fine mesh simulation

References

[1] L. Ge and F. Sotiropoulos. A numerical method for solving the 3d unsteady incompressible navier-stokes equations in curvilinear domains with complex immersed boundaries. *Journal of Computational Physics*, 225(2):1782, 2007.