Introduction

In this notebook, we will be exploring the formulation for a basic 2D Navier-Stokes solver as per the activity outlined for ME 5653.

In [1]:
```
import sys
import os
import time
import numpy as np
import matplotlib.pyplot as plt

# Add the directory containing your module to sys.path
module_path = os.path.abspath(os.path.join('..', r"A:\Users\mtthl\Documents\Education\ME5653_C
sys.path.append(module_path)

from distributedObjects import *
from distributedFunctions import *
```

(A) Non-Linear Terms

The first alteration to the Navier-Stokes equation is to change the non-linear or advective terms to a flux form to allow for a Linear Algebra method to solve. This means that:

$$u\frac{\partial u}{\partial x} = \frac{\partial \frac{1}{2}u^2}{\partial x}$$

$$v\frac{\partial u}{\partial y} = \frac{\partial \frac{1}{2}uv}{\partial y}$$

$$u\frac{\partial v}{\partial x} = \frac{\partial \frac{1}{2}uv}{\partial x}$$

$$v\frac{\partial v}{\partial y} = \frac{\partial \frac{1}{2}v^2}{\partial y}$$

(B) Discretization

The next step is to then go in and discretize the Navier-Stokes equations and Poisson pressure equation. For this activity, we will be using a sixth (6th) order Finite Difference (FD).

First, we need to find the coefficients for the 1st derivative.

In [2]:
```
first_interior_gradient = numericalGradient( 1 , ( 3 , 3 ) )
first_interior_gradient.coeffs
```

Out[2]:
```
array([-1.66666667e-02,  1.50000000e-01, -7.50000000e-01,  3.70074342e-16,
        7.50000000e-01, -1.50000000e-01,  1.66666667e-02])
```

In [3]:
```
first_LHS_gradient = numericalGradient( 1 , ( 0 , 6 ) )
first_LHS_gradient.coeffs
```

Out[3]:
```
array([-2.45      ,  6.        , -7.5       ,  6.66666667, -3.75      ,
        1.2       , -0.16666667])
```

In [5]:
```
first_RHS_gradient = numericalGradient( 1 , ( 6 , 0 ) )
first_RHS_gradient.coeffs
```

Out[5]:
```
array([ 0.16666667, -1.2       ,  3.75      , -6.66666667,  7.5       ,
       -6.        ,  2.45      ])
```

Thus, the first derivative becomes:

1st Derivative - Interior Points - 6th Order

$$\frac{\partial \phi}{\partial x} = \frac{-\frac{1}{60}\phi_{i-3} + \frac{3}{20}\phi_{i-2} - \frac{3}{4}\phi_{i-1} + \frac{3}{4}\phi_{i+1} - \frac{3}{20}\phi_{i-2} + \frac{1}{60}\phi_{i-3}}{\Delta x}$$

1st Derivative - LHS Boundary Points - 6th Order

$$\frac{\partial \phi}{\partial x} = \frac{-\frac{49}{20}\phi_{i} + 6\phi_{i+1} - \frac{15}{2}\phi_{i+2} + \frac{20}{3}\phi_{i+3} - \frac{15}{4}\phi_{i+4} + \frac{6}{5}\phi_{i+5} - \frac{1}{6}\phi_{i+6}}{\Delta x}$$

1st Derivative - RHS Boundary Points - 6th Order

$$\frac{\partial \phi}{\partial x} = \frac{\frac{1}{6}\phi_{i-6} - \frac{6}{5}\phi_{i-5} + \frac{15}{4}\phi_{i-4} - \frac{20}{3}\phi_{i-3} + \frac{15}{2}\phi_{i-2} - 6\phi_{i-1} + \frac{49}{20}\phi_{i}}{\Delta x}$$

Second, we need to find the coefficients for the 2nd derivative.

In [6]:
```
second_interior_gradient = numericalGradient( 2 , ( 3 , 3 ) )
second_interior_gradient.coeffs
```

Out[6]:
```
array([ 0.01111111, -0.15      ,  1.5       , -2.72222222,  1.5       ,
       -0.15      ,  0.01111111])
```

In [7]:
```
second_LHS_gradient = numericalGradient( 2 , ( 0 , 6 ) )
second_LHS_gradient.coeffs
```

Out[7]:
```
array([  4.51111111, -17.4       ,  29.25      , -28.22222222,
        16.5       ,  -5.4       ,   0.76111111])
```

In [8]:
```
second_RHS_gradient = numericalGradient( 2 , ( 6 , 0 ) )
second_RHS_gradient.coeffs
```

Out[8]:
```
array([  0.76111111,  -5.4       ,  16.5       , -28.22222222,
        29.25      , -17.4       ,   4.51111111])
```

Thus, the second derivative becomes:

2nd Derivative - Interior Points - 6th Order

$$\frac{\partial^2 \phi}{\partial x^2} = \frac{\frac{1}{90}\phi_{i-3} - \frac{3}{20}\phi_{i-2} + \frac{3}{2}\phi_{i-1} - \frac{245}{90}\phi_i + \frac{3}{2}\phi_{i+1} - \frac{3}{20}\phi_{i+2} + \frac{1}{90}\phi_{i+3}}{\Delta x}$$

2nd Derivative - LHS Boundary Points - 6th Order

$$\frac{\partial^2 \phi}{\partial x^2} = \frac{\frac{406}{90}\phi_i - \frac{87}{5}\phi_{i+1} + \frac{117}{4}\phi_{i+2} - \frac{254}{9}\phi_{i+3} + \frac{33}{2}\phi_{i+4} - \frac{27}{5}\phi_{i+5} + \frac{137}{180}\phi_{i+6}}{\Delta x}$$

2nd Derivative - RHS Boundary Points - 6th Order

$$\frac{\partial^2 \phi}{\partial x^2} = \frac{\frac{137}{180}\phi_{i-6} - \frac{27}{5}\phi_{i-5} + \frac{33}{2}\phi_{i-4} - \frac{254}{9}\phi_{i-3} + \frac{117}{4}\phi_{i-2} - \frac{87}{5}\phi_{i-1} + \frac{406}{90}\phi_i}{\Delta x}$$

With this discretization, the NS equations can be discretized.

(C) Integration-Ready NS Equations

The next step is to re-arrange the Navier-Stokes equations so that they are time-integration ready. Note that $P' = P/\rho$

x-Component Navier Stokes

$$\frac{\partial u}{\partial t} = \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) - \frac{\partial P'}{\partial x} - \frac{\partial \frac{1}{2}u^2}{\partial x} - \frac{\partial \frac{1}{2}uv}{\partial y}$$

y-Component Navier Stokes

$$\frac{\partial v}{\partial t} = \nu \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) - \frac{\partial P'}{\partial y} - \frac{\partial \frac{1}{2} v^2}{\partial y} - \frac{\partial \frac{1}{2} uv}{\partial x}$$

(D) Time Integration Scheme

For time integration, we will use the Adams-Bashforth 3rd order scheme, which is formulated as:

$$\phi^{n+1} = \phi^n + \frac{\Delta t}{12} \left(23 f\left(t^n, u^n\right) - 16 f\left(t^{n-1}, u^{n-1}\right) + 5 f\left(t^{n-2}, u^{n-2}\right) \right)$$

Where

$$f = \frac{\partial \phi}{\partial t}$$

And n is the time step index.

(E) Poisson Equation Formulation

The next step is to use the Gauss-Seidel formulation to solve the Poisson pressure equation, as below.

$$\frac{\partial^2 P'}{\partial x^2} + \frac{\partial^2 P'}{\partial y^2} = - \left(\left(\frac{\partial u}{\partial x} \right)^2 + 2 \frac{\partial u}{\partial y} \frac{\partial v}{\partial x} + \left(\frac{\partial v}{\partial y} \right)^2 \right)$$

The Gauss-Seidel (GS) formulation for 2D is formulated as below for a Laplace equation (RHS of Poisson equation is zero (0)).

$$\phi_{i,j}^{k+1} = \frac{\phi_{i-1,j}^k + \phi_{i+1,j}^{k+1} + \beta^2 (\phi_{i,j+1}^k + \phi_{i,j-1}^{k+1})}{2(1+\beta^2)}$$

Where

$$\beta = \frac{\Delta x}{\Delta y}$$

And i & j are spatial indices, and k is the iteration index.

When we change this formulation for the Poisson equation, this scheme is formulated as:

$$\phi_{ii}^{k+1} = \frac{1}{a_{ii,ii}} \left(b_{ii} - \sum_{jj=1}^{ii-1} a_{ii,jj} \phi_{jj}^{k+1} - \sum_{jj=ii+1}^{n} a_{ii,jj} \phi_{jj}^k \right)$$

For

$$[A] < \phi > = < b >$$

And ii and jj are indices for the matrix $[A]$.

The GS formulation gets applied to the kinematic pressure (P'). The pressure will get iterated on using the GS formulation, and the values then get differentiated in the Poisson equation as above. The difference between the left and right side (LHS and RHS) of this equation becomes the residual. Thus,

$$residuals = -\left(\left(\frac{\partial u}{\partial x}\right)^2 + 2\frac{\partial u}{\partial y}\frac{\partial v}{\partial x} + \left(\frac{\partial v}{\partial y}\right)^2\right) - \frac{\partial^2 P'}{\partial x^2} - \frac{\partial^2 P'}{\partial y^2}$$

Then the field for pressure is solved until the residuals reach some acceptable level, depending on the use of the NS equations.

(F) Boundary Conditions

The important condition for the Poisson pressure equation are the boundary conditions, which need to be defined. Since pressure is likely not known at most boundaries, these will be likely Neumann or gradient-based conditions. We will derive the conditions similarly to how I did in Appendix C of my Masters thesis.

For a no-slip condition, the velocity is 0 on all components ($u = v = 0$). Since velocity is constant, x-gradient of all velocities are zero, and the second x derivative of velocity is zero. Since the x-gradient is zero, the continuity equation indicates the y-gradient of v is zero too. Also, we can assume steady for this.

Thus,

$$0 = \frac{\partial P'}{\partial x} - \nu\left(\frac{\partial^2 u}{\partial y^2}\right)$$

And

$$0 = \frac{\partial P'}{\partial y} - \nu\left(\frac{\partial^2 v}{\partial y^2}\right)$$

Which then becomes,

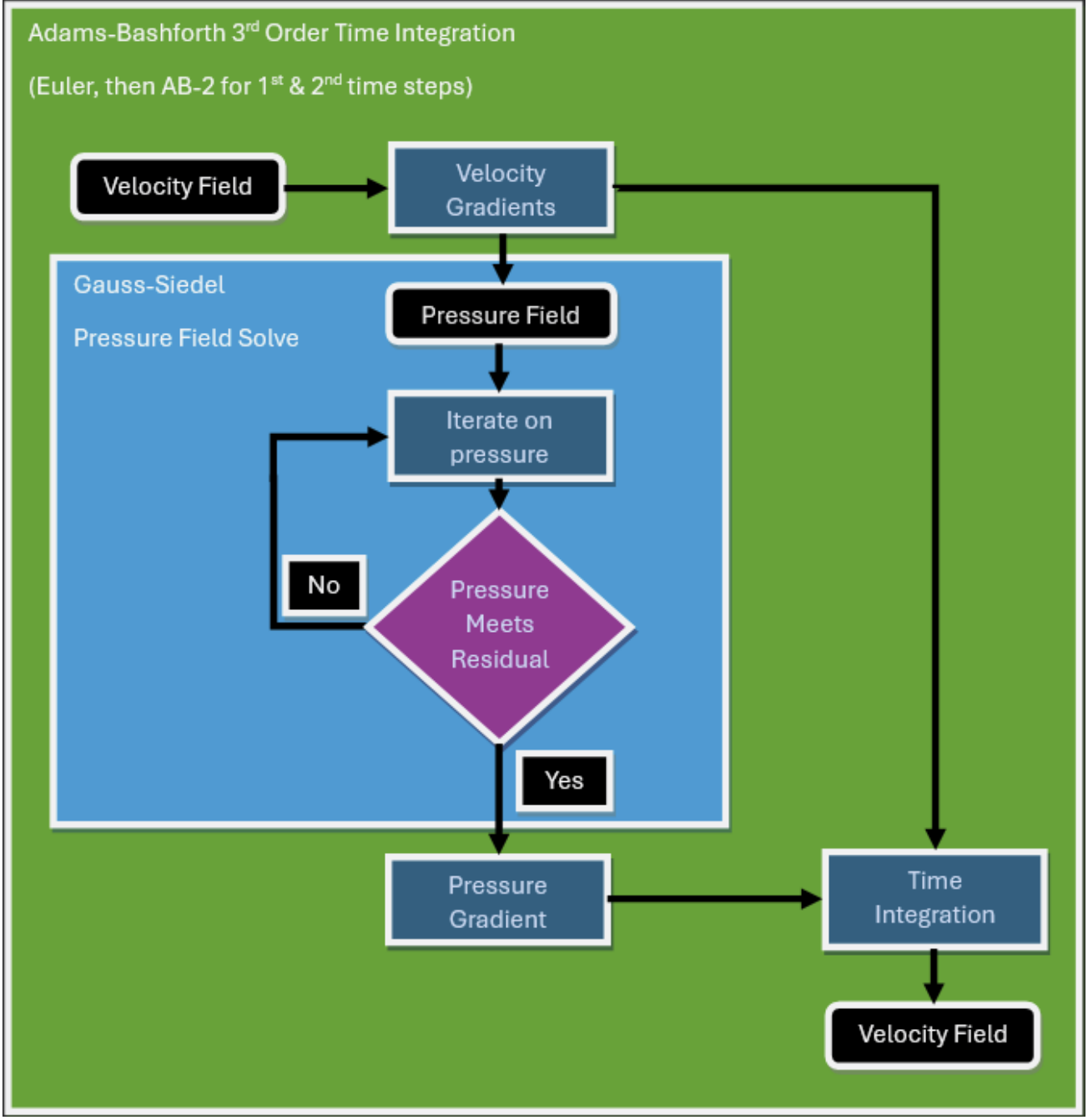
$$\frac{\partial P'}{\partial x} = \nu\frac{\partial^2 u}{\partial y^2}$$

And

$$\frac{\partial P'}{\partial y} = \nu\frac{\partial^2 v}{\partial y^2}$$

(G) Pseudo-Code

Finally, we can put all this together to show the algorithm to solve the NS equations.



To translate this helpful flowchart into pseudo-code:

In [ ]:
```
for i , t in enumerate( t_values ):
    #
    # Pull velocity field
    #

    #
    # Calculate velocity gradients
    #

    # Diffusive gradients

    # Advective/flux gradients

    #
    # Solve pressure field
    #
    converged = False
    c = 0
    while not converged:
        #
        # Iterate on pressure
        #
```

```
        #
        # Measure residuals
        #

        # Loop iteration
        c+=1
        if field_residuals <= p_residual:
            converged = True

    # Pressure gradients

    #
    # Time integration
    #

    # Add fields
    f[i]=# Current integration function if a running summation is not done

    if i==0:
        # Euler time step
        u[i+1]=f[i]*dt+u[i]

    elif i==1:
        # AB-2 time step
        u[i+1]=(dt/2)*(3*f[i]-f[i-1])+u[i]

    else:
        # AB-3 time step
        u[i+1]=(dt/12)*(23*f[i]-16*f[i-1]+5*f[i-2])+u[i]
```

(H) SIMPLE Algorithm

For implicit problems, we can use SIMPLE (Semi-Implicit method for Pressure Linked Equations). The basic premise of the solve is that a virtual intermediate point, and then correct it to find an acceptably converged solution.

Thus, the pseudo code becomes:

In [ ]:
```
#
# Pull velocity field
#

converged_u = False
while not converged_u:

    ############################
    #
    # Virtual step
    #
    ############################

    #
    # Calculate velocity gradients
    #

    # Diffusive gradients

    # Advective/flux gradients

    #
    # Solve pressure field
```

```
#
converged = False
c = 0
while not converged:
    #
    # Iterate on pressure
    #

    #
    # Measure residuals
    #

    # Loop iteration
    c+=1
    if field_residuals <= p_residual:
        converged = True

# Pressure gradients

#
# Time integration
#

# Add fields
f_0[i]=# Current integration function if a running summation is not done

if i==0:
    # Euler time step
    u_0[i+1]=f_0[i]*dt+u_0[i]

elif i==1:
    # AB-2 time step
    u_0[i+1]=(dt/2)*(3*f_0[i]-f_0[i-1])+u_0[i]

else:
    # AB-3 time step
    u_0[i+1]=(dt/12)*(23*f_0[i]-16*f_0[i-1]+5*f_0[i-2])+u_0[i]

##############################
#
# Correction step
#
##############################

#
# Calculate velocity gradients
#

# Diffusive gradients

# Advective/flux gradients

#
# Solve pressure field
#
converged = False
c = 0
while not converged:
    #
    # Iterate on pressure
    #

    #
    # Measure residuals
    #
```

```
        # Loop iteration
        c+=1
        if field_residuals <= p_residual:
            converged = True

    # Pressure gradients

    # New velocity fields
    u[i+1]=u_0[i+1]-(A/(2*dcc))*(P_c[ii+1,j]-P_c[ii-1,j])


    if velocity_residuals<=u_residual:
        converged_u = True
```