In [1]:

```python
import pandas as pd
import os
from enum import Enum
import numpy as np
import scipy
from numpy import sqrt, sin, cos, tan, pi
from scipy.integrate import odeint
from scipy.interpolate import InterpolatedUnivariateSpline
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick
from matplotlib import ticker, cm
from matplotlib import ticker
from scipy.optimize import minimize
from scipy.optimize import Bounds
import import_ipynb
from linearInterpolation import interpolate
%matplotlib
```

```
importing Jupyter notebook from linearInterpolation.ipynb
Using matplotlib backend: Qt5Agg
```

In [2]:

```python
density = 997
dynamicViscosity = 0.0008891
kinematicViscosity = 8.917*10**-7

class flowParameters():
    def __init__(self, outerRadius, innerOuterRatio, discSpacing, \
                 relativeTipTangential, relativeTipRadial, reynoldMS, profileN = 2):
        self.innerRadius = innerOuterRatio*outerRadius
        self.outerRadius = outerRadius
        self.innerOuterRatio = innerOuterRatio
        self.discSpacing = discSpacing

        self.relativeTipRadial = relativeTipRadial
        self.relativeTipTangential = relativeTipTangential
        self.reynoldMS = reynoldMS
        self.Fpo = (profileN + 1)/3

        self.omega = (kinematicViscosity/4*discSpacing)*(reynoldMS/relativeTipRadial)
        self.tipVelocity = self.omega*outerRadius
        self.vRadial = -self.tipVelocity*relativeTipRadial
        self.vTheta = self.relativeTipTangential*self.tipVelocity + self.tipVelocity

def bothODE(y,x,instance):
    y0,y1 = y

    nTerm = 3*instance.Fpo - 1 # article definition
    Vr0 = instance.vRadial/instance.tipVelocity

    firstSolution = -(2*nTerm + 1)/(nTerm + 1) + (8*(2*nTerm + 1)*x/instance.reynoldMS - 1/x)*y0
    secondSolution = (4*(nTerm + 1)/(2*nTerm + 1))*(1/x**3)*(Vr0**2 + (y0*x)**2) +\
                    4*y0 + 2*x + 32*(nTerm + 1)*(Vr0**2)/(x*instance.reynoldMS)
    return [firstSolution, secondSolution]

def rotorEff(firstAnswer, rs, instance):
    return (1 - (firstAnswer[-1] + instance.innerOuterRatio)*instance.innerOuterRatio\
          /(firstAnswer[0] + 1))

def pathLine(instance, rsi, solution, startingAngle=0, k=1, relative=True):
    innerOuterRatio = instance.innerRadius/instance.outerRadius
    dt = k*abs(instance.relativeTipRadial)/(innerOuterRatio**10)
    def nextTheta(instance, rsi, solution, currentPosition, dt=dt):
        currentTheta, currentXi = currentPosition[0], currentPosition[1]
        t_Solution = solution[:, 0]
        if relative:
            V_t = instance.tipVelocity*interpolate(currentXi, rsi, t_Solution)
        else:
            V_t = instance.tipVelocity*interpolate(currentXi, rsi, t_Solution) + \
                currentXi*instance.tipVelocity
        n_Theta = (V_t/(instance.outerRadius*currentXi))*dt + currentTheta
        return n_Theta
    def nextXi(instance, rsi, currentPosition, dt=dt):
        currentXi = currentPosition[1]
        V_r = instance.tipVelocity*abs(instance.relativeTipRadial)/currentXi
        n_Xi = currentXi - V_r*dt/instance.outerRadius
        return n_Xi
    position = np.array([startingAngle, 1])  #currentPosition [theta, xi]
    storage = [position]
    storageAngle = []
    while position[1]>innerOuterRatio:
        currentVR = abs((1/position[1])*instance.relativeTipRadial)
        currentVT = interpolate(position[1], rsi, solution[:, 0])
        currentAngle = np.arctan(currentVT/ currentVR)*180/pi
        storageAngle.append([position[1],currentAngle])
        position = [nextTheta(instance, rsi, solution, position), nextXi(instance, rsi,
position)]
        storage.append(position)
    return np.array(storage), np.array(storageAngle)
```

In [7]:

```python
"""W0 = 2, Rem* = 10, radius ratio = 0.2, Vro = 0.05"""
testCase1 = flowParameters(0.5, 0.2, 0.0001, 2, 0.05, 10)

rs = np.linspace(1, testCase1.innerOuterRatio, 100)
sol = odeint(bothODE, [testCase1.relativeTipTangential, 0.0], rs, args=(testCase1,))

fig, ax = plt.subplots(2,1, figsize=(7.5,15))

ax[0].plot(rs, sol[:,0]+rs, label="Flow Absolute", color='black')
ax[0].plot(rs, rs, '--', label="Disc", color='black')
ax[0].set_ylim(bottom=0, top=3.5)
ax[0].set_xlim(left=0.2, right=1.0)
ax[0].set_xlabel(r"$\xi$", fontsize=15)
ax[0].set_ylabel("$\^{W}$", fontsize=15)
ax[0].tick_params(axis='x', labelsize=15)
ax[0].tick_params(axis='y', labelsize=15)
ax[0].legend(fontsize=15, loc="upper right")

ax[1].plot(rs, sol[:,1], label="", color='black')
ax[1].set_ylim(top=0, bottom=-25)
ax[1].set_xlim(left=0.2, right=1.0)
ax[1].set_xlabel(r"$\xi$", fontsize=15)
ax[1].set_ylabel("$\^{P}$", fontsize=15)
ax[1].tick_params(axis='x', labelsize=15)
ax[1].tick_params(axis='y', labelsize=15)

ax[0].invert_xaxis()
ax[1].invert_xaxis()
```

In [8]:

```python
"""W0 = 1.1, Rem* = 5, radius ratio = 0.2, Vro = 0.05"""
testCase2 = flowParameters(0.5, 0.2, 0.0001, 1.1, 0.05, 5)

rs = np.linspace(1, testCase2.innerOuterRatio, 100)
sol = odeint(bothODE, [testCase2.relativeTipTangential, 0.0], rs, args=(testCase2,))

fig, ax = plt.subplots(2,1, figsize=(7.5,15))

ax[0].plot(rs, sol[:,0]+rs, label="Flow Absolute", color='black')
ax[0].plot(rs, rs, '--', label="Disc", color='black')
ax[0].set_ylim(bottom=0, top=3.5)
ax[0].set_xlim(left=0.2, right=1.0)
ax[0].set_xlabel(r"$\xi$", fontsize=15)
ax[0].set_ylabel("$\^{W}$", fontsize=15)
ax[0].tick_params(axis='x', labelsize=15)
ax[0].tick_params(axis='y', labelsize=15)
ax[0].legend(fontsize=15, loc="upper right")

ax[1].plot(rs, sol[:,1], label="", color='black')
ax[1].set_ylim(top=0, bottom=-6)
ax[1].set_xlim(left=0.2, right=1.0)
ax[1].set_xlabel(r"$\xi$", fontsize=15)
ax[1].set_ylabel("$\^{P}$", fontsize=15)
ax[1].tick_params(axis='x', labelsize=15)
ax[1].tick_params(axis='y', labelsize=15)

ax[0].invert_xaxis()
ax[1].invert_xaxis()
```

In [9]:

```python
"""W0 = 1.1, Rem* = 5, radius ratio = 0.2, Vro = 0.05, profileN = 2, 4, 6"""
nList = [2,5,8]
style = ['-', '--', '-.']
fig, ax = plt.subplots(figsize=(15,15))
for i in range(len(nList)):

    testCase2 = flowParameters(0.5, 0.2, 0.0001, 1.1, 0.05, 5, profileN = nList[i])

    rs = np.linspace(1, testCase2.innerOuterRatio, 100)
    sol = odeint(bothODE, [testCase2.relativeTipTangential, 0.0], rs, args=(testCase2,))

    ax.plot(rs, sol[:,1], style[i] ,label=f"Fpo = {i+1}", color='black')

ax.set_ylim(top=0, bottom=-6)
ax.set_xlim(left=0.2, right=1.0)
ax.set_xlabel(r"$\xi$", fontsize=15)
ax.set_ylabel("$\^{P}$", fontsize=15)
ax.tick_params(axis='x', labelsize=15)
ax.tick_params(axis='y', labelsize=15)
ax.invert_xaxis()
ax.legend(fontsize=15, loc="upper right")
```

Out[9]:

```
<matplotlib.legend.Legend at 0x19f269e67c8>
```

In [14]:

```python
"""W0 = 1-10, Rem* = 1-20, radius ratio = 0.1-0.4, Vro = 0.05, profileN = 2"""
W0List = np.arange(1,10,0.5)
RemList = np.arange(1,20,0.5)
radiusRatioList = np.arange(0.1,0.4,0.1)
Vr0 = 0.05
profileN = 2
storeMechEffDict = {}
for i in range(len(radiusRatioList)):
    rs = np.linspace(1, radiusRatioList[i], 100)
    mechEffStorage = np.zeros((len(RemList), len(W0List)))
    for x in range(len(RemList)):
        for y in range(len(W0List)):
            testCase = flowParameters(0.5, radiusRatioList[i], 0.0002, W0List[y], \
                                      Vr0, RemList[x])
            sol = odeint(bothODE, [testCase.relativeTipTangential, 0.0], rs, args=(testCase,))
            mechEff = rotorEff(sol[:,0], rs, testCase)
            mechEffStorage[x, y] = mechEff
    storeMechEffDict[radiusRatioList[i]] = np.array(mechEffStorage)
```

In [25]:

```python
X,Y = np.meshgrid(RemList,W0List)
levels = np.arange(0, 105, 5)

for i in storeMechEffDict:
    fig, ax = plt.subplots(figsize=(15,15))
    cs = ax.contourf(X,Y,100*storeMechEffDict[i].transpose(),levels=levels,extend='both')

    ax.set_title("Mechanical Efficiency"+"\n"+rf"$\xi={round(i,1)}$",fontsize=30, pad=20)
    ax.set_xlabel('Rem*', fontsize=25, labelpad=10)
    ax.tick_params(axis='x', labelsize=20)

    ax.set_ylabel('W0', fontsize=25, labelpad=10)
    ax.tick_params(axis='y', labelsize=20)

    cbar = plt.colorbar(cs,ax=ax)
    ticklabs = cbar.ax.get_yticklabels()
    cbar.ax.set_ylabel('Efficiency (%)', fontsize=25, labelpad=10)
    cbar.ax.set_yticklabels(ticklabs, fontsize=20)
```

In [ ]:

```python
```