In [7]:

```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 cfdPostProcessing import postProcess, rakeProcess
from teslaModelValidation import pathLine
%matplotlib

'''Water'''
density = 997
dynamicViscosity = 0.0008891
kinematicViscosity = 8.917*10**-7
TotalMassFlowRate = 1 # 0.5, 2

'''Can be Set'''
voluteThickness = 0.005
discThickness = 0.0008
discSpacing = 0.0002
wallSpace = 0.001
wallDisplacement = 0.003

'''Base Case'''
nDisc = 5
chosenScaleDownFactor = 0.164/0.073
rotorOuter = 0.073
rotorInner = 0.3*rotorOuter
revPerMinute = [2000]

'''formatting for plots'''
formatter = ticker.ScalarFormatter(useMathText=True)
formatter.set_scientific(True)
formatter.set_powerlimits((-1,1))
class flowParameters():
    def __init__(self, innerRadius, outerRadius, discSpacing, discThickness, numberSpacing,
                 voluteThickness, voluteWallSpace, upperClearance, wallDisplacement,
                 totMassFlowRate, density, RPM, k_Width_h0 = 0.798, profileN = 2):
        self.innerRadius = innerRadius
        self.outerRadius = outerRadius
        self.discSpacing = discSpacing
        self.discThickness = discThickness
        self.voluteWallSpace = voluteWallSpace
        self.upperClearance = upperClearance
        self.numberSpacing = numberSpacing
        self.massFlowRate = totMassFlowRate
        self.density = density
        self.Fpo = (profileN + 1)/3

        self.voluteSpace = numberSpacing*discSpacing + (numberSpacing-1)*discThickness
        self.totalVoluteSpace = 2*discThickness + self.voluteSpace + 2*self.voluteWallSpace +
2*wallDisplacement
        self.h0 = k_Width_h0*self.totalVoluteSpace
        self.r0 = self.outerRadius + self.upperClearance + voluteThickness +
self.totalVoluteSpace/2 + \
            np.cos(np.arcsin((self.totalVoluteSpace - self.h0)/self.h0))*self.h0

        #formula
        self.inletAngle = flowParameters.derivedAngle(self.voluteSpace, self.h0, self.r0)
        self.vRadial, self.vTheta = flowParameters.velocityInlet(self)

        self.omega = RPM*2*pi/60
        self.DH = 2*self.discSpacing
        self.massFlowRatePD = self.massFlowRate/self.numberSpacing
        self.volumeFlowRatePD = self.massFlowRatePD/density

        self.tipVelocity = self.omega*self.outerRadius

        self.relativeTipTangential = (self.vTheta - self.tipVelocity)/self.tipVelocity
        self.relativeTipRadial = self.vRadial/self.tipVelocity

        self.innerOuterRatio = self.innerRadius/self.outerRadius
        self.reynoldM = self.massFlowRatePD/(pi*self.outerRadius*dynamicViscosity)
        self.reynoldMS = self.reynoldM * self.DH / self.outerRadius

    def derivedAngle(vSpace, vIRadius, vRadius):
        degree = np.arctan(2*vSpace*vRadius/(vIRadius**2))
        return 0.5*pi - degree

    def velocityInlet(self):
        effectiveArea = 2*pi*(self.outerRadius+self.upperClearance)*
(self.numberSpacing*self.discSpacing)
        vRadial = self.massFlowRate/(effectiveArea*self.density)
        vTheta = vRadial/tan(self.inletAngle)
        vRadialDisc = vRadial*self.outerRadius/(self.outerRadius-self.upperClearance)
        return -vRadialDisc, vTheta

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): # ignore for now
    return (1 - (firstAnswer[-1] + instance.innerOuterRatio)*instance.innerOuterRatio\
            /(firstAnswer[0] + 1))

def power(firstAnswer, rs, instance):
    firstAnswerFlip = np.squeeze(np.flip(firstAnswer))
    rsFlip = np.flip(rs)

    constantTerm = (instance.outerRadius**3)*(2*pi/instance.discSpacing)*\
            (6*dynamicViscosity*instance.tipVelocity)*instance.Fpo
    integrateTerm = firstAnswerFlip*np.power(rsFlip,2)
    return 2*instance.omega*instance.numberSpacing*\
        constantTerm*scipy.integrate.simps(integrateTerm, x = rsFlip)

def efficiencyIdeal(solution, rs, instance):
    innerOuterRatio = instance.innerRadius/ instance.outerRadius
    innerDiscSpeed = innerOuterRatio*instance.tipVelocity
    inletKE = instance.vRadial**2 + instance.vTheta**2
    pressureDrop = abs(solution[-1, 1]*((density*(instance.tipVelocity)**2)/2))

    outletVr = instance.vRadial/innerOuterRatio
    outletVt = (solution[-1, 0]*instance.tipVelocity) + instance.omega*instance.innerRadius
    outletKE = outletVr**2 + outletVt**2
    energyInput = (0.5*(inletKE-outletKE) + pressureDrop/density)

    energyOutput = (instance.tipVelocity**2)*(solution[0]+1) - \
            innerDiscSpeed*(solution[-1]*instance.tipVelocity + innerDiscSpeed)
    return (energyOutput*100/energyInput)[0] # in percentage

def solutionGenerator(rotorInner, rotorOuter, discSpacing, discThickness, numberSpacing,
                    voluteThickness, wallSpace, upperClearance, wallDisplacement,
                    TotalMassFlowRate, density, RPM, profileN = 2, rsPoint = 100):
    KJ = flowParameters(rotorInner, rotorOuter, discSpacing, discThickness, numberSpacing,
                voluteThickness, wallSpace, upperClearance, wallDisplacement,
                TotalMassFlowRate, density, RPM, profileN = profileN)
    firstODEinitial,secondODEinitial = KJ.relativeTipTangential, 0
    rs = np.linspace(1, KJ.innerOuterRatio, rsPoint)
    sol = odeint(bothODE, [firstODEinitial,secondODEinitial], rs, args=(KJ,))
    return KJ, sol, rs

'''Extras'''
def profilePlot(axPlot, instance, numberZPoints): # profile plot in between discs
    zPoints = np.linspace(-instance.discSpacing/2, instance.discSpacing/2, numberZPoints)
    nVal = 3*instance.Fpo - 1
    xVal = ((nVal+1)/nVal)*(np.full((numberZPoints,),1) -
np.power(2*zPoints/instance.discSpacing,nVal))
    axPlot.set_ylabel("Z position relative to DSC", color="white")
    axPlot.set_xlabel("Profile Magnitude (Dimensionless)", color="white")
    axPlot.plot(xVal, zPoints)

'''finding the best rpm for highest power output (not working)'''
def costJ(x, instance):
    optRPM = x[0]
    instance.omega = optRPM*2*pi/60

    firstODEinitial, secondODEinitial = instance.relativeTipTangential, 0
    rs = np.linspace(1, instance.innerOuterRatio, 100)
    sol = odeint(bothODE, [firstODEinitial,secondODEinitial], rs, args=(instance,))
    return (1000-power(sol[:,0], rs, instance))**2



def shaftLosses(instance): # negligible
    wR2 = instance.omega*instance.outerRadius**2
    reDisc = wR2/kinematicViscosity
    tipGap = (instance.totalVoluteSpace)/2 + instance.upperClearance
    endGap = instance.voluteWallSpace
    condition = 470.5*instance.outerRadius/endGap
    if reDisc < condition:
        exponent = 1 # laminar gap
    else:
        exponent = 0.25 # turbulent gap
    cFactorGap = ((instance.outerRadius/(reDisc*endGap))**exponent)*\
        (2*pi if exponent == 1 else 0.00622)*(1/(instance.numberSpacing + 1))
    cFactorTip = (instance.discThickness/tipGap)*(4*pi*kinematicViscosity/wR2)
    torqueLoss = (0.5*instance.discSpacing/instance.outerRadius)*(cFactorGap + cFactorTip)
    return torqueLoss*instance.omega*(instance.numberSpacing+1)

def shearPoints(solution, instance):
    nVal = 3*instance.Fpo - 1
    factor = 2*dynamicViscosity*(nVal+1)*instance.tipVelocity/KJ.discSpacing
    return factor*solution[:,0]

def torqueCalculator(solution, rs, instance):
    totalPower = power(solution[:,0], rs, instance)
    torquePerDisc = totalPower/(instance.omega*2*(instance.numberSpacing))
    return torquePerDisc

def safeFloatConvert(x):
    try:
        float(x)
        if np.isnan(x):
            return False
        return True
    except:
        return False
```

```
Using matplotlib backend: Qt5Agg
```

In [12]:

```python
'''W analysis with RPM'''
rpmList = [100,200,300]
fig, ax = plt.subplots()
for i in range(len(rpmList)):
    KJ, solKJ, rsKJ = solutionGenerator(rotorInner, rotorOuter, discSpacing, discThickness,
nDisc-1,
                    voluteThickness, wallSpace, 0, wallDisplacement,
                    TotalMassFlowRate, density, rpmList[i])
    ax.plot(rsKJ, solKJ[:,0])
```

In [4]:

```python
'''Base Case Simulation'''

KJ, sol, rs = solutionGenerator(rotorInner, rotorOuter, discSpacing, discThickness, nDisc-1,
                voluteThickness, wallSpace, 0, wallDisplacement,
                TotalMassFlowRate, density, revPerMinute[0])
radialAve = (1/rs)*KJ.vRadial/KJ.tipVelocity
data = {
    'rs': rs,
    'dimensionless Tangential Velocity': sol[:,0],
    'dimnesionless Radial Velocity': radialAve
}
df = pd.DataFrame(data=data)
df.to_excel("excelFiles\\dimensionlessVelocity.xlsx", index=False)
```

In [6]:

```python
'''Base Case Simulation'''
#rpmComparisonCase = [100,2000,3000]
nProfileList = [2,8]
for k in range(len(nProfileList)):

    KJ, solKJ, rsKJ = solutionGenerator(rotorInner, rotorOuter, discSpacing, discThickness,
nDisc-1,
                    voluteThickness, wallSpace, 0, wallDisplacement,
                    TotalMassFlowRate, density, 200, profileN=nProfileList[k])

    print(
            f"Disc Number:\t{KJ.numberSpacing+1}"+"\n"+
            f"Total Volute Space:\t\t{KJ.totalVoluteSpace} m"+"\n"+
            f"Total Disc Space:\t\t{KJ.voluteSpace} m"+"\n"+
            f"r0 is:\t\t{KJ.r0} m"+"\n"+
            f"Outer radius:\t{KJ.outerRadius} m"+"\n"+
            f"h is:\t\t{KJ.h0} m"+"\n"+
            f"Angle:\t\t{KJ.inletAngle*180/pi} dg from tangent line"+"\n"+
            f"R ratio:\t\t{KJ.innerOuterRatio}"+"\n"+
            f"Vr:\t\t{KJ.vRadial} \t\tVt:\t\t{KJ.vTheta}"+"\n"+
            f"V_tip:\t\t{KJ.tipVelocity}\n"+
            f"Reynold:\t{KJ.reynoldM}"+"\n"+
            f"Reynold*:\t{KJ.reynoldMS}"+"\n"+
            f"W0:\t\t{KJ.relativeTipTangential}"+"\n"+
            f"Power:\t\t{power(solKJ[:,0], rsKJ, KJ)} W"+"\n"+
            f"Efficiency:\t{efficiencyIdeal(solKJ, rsKJ, KJ)} %"+"\n"+
            f"Friction Loss: \t{shaftLosses(KJ)} W"
        )

    xs = np.linspace(0, 2*pi, 100)
    rInner = np.squeeze(np.full((1,100), rsKJ[-1]))
    rOuter = np.squeeze(np.full((1,100), rsKJ[0]))

    thetaRange = np.linspace(0, 2*pi, 6)
    fig = plt.figure(constrained_layout=True)
    ax0 = fig.add_gridspec(9, 8)
    ax = fig.add_subplot(ax0[1:8, 1:5], polar=True)
    ax1 = fig.add_subplot(ax0[1:7, 6:7])
    relative = False
    for i in range(len(thetaRange)):
        basePlot, baseAnglePlot = pathLine(KJ, rsKJ, solKJ, startingAngle=thetaRange[i], k=1e-
10, relative=relative)
        if relative:
            ax.plot(basePlot[:, 0], basePlot[:,1], "-.",color="maroon", linewidth=2.0)
        else:
            ax.plot(basePlot[:, 0], basePlot[:,1], color="maroon", linewidth=2.0)
    ax.plot(xs, rInner, color="black")
    ax.plot(xs, rOuter, color="black")
    ax.grid(False, axis='x')
    ax.tick_params(axis='x', labelsize=15)
    ax.tick_params(axis='y', labelsize=15)
    ax.set_ylim(0, rsKJ[0])

    inner = rsKJ[-1]
    outer = rsKJ[0]
    thetas = np.linspace(0, 2*pi, 201)
    radials = np.linspace(inner, outer, 201)
    xv, yv = np.meshgrid(thetas, radials)
    r = yv**2
    ax.contourf(xv, yv, r, colors="grey")

    ax1.plot(rsKJ, shearPoints(solKJ, KJ), color='black')
    ax1.set_ylabel(r"$\tau_w$", labelpad=10, fontsize = 15)
    ax1.set_xlabel(r"$\xi$", labelpad=10, fontsize = 15)
    ax1.set_title('Wall Shear', fontsize = 15)
    ax1.set_ylim(top=500)

    currentTorque = torqueCalculator(solKJ, rsKJ, KJ)
    ax1.text(0.7, 250, f'Torque: {round(currentTorque,3)}Nm\nRPM: 200\nn: {nProfileList[k]}',
color="black", fontsize=15)
```

```
Disc Number:    5
Total Volute Space:             0.0124 m
Total Disc Space:               0.0032 m
r0 is:          0.09428173264159682 m
Outer radius:   0.073 m
h is:           0.0102144 m
Angle:          8.809972573699244 dg from tangent line
R ratio:        0.3
Vr:             -2.7334... [illegible]          Vt:             15.808... [illegible]
V_tip:          [illegible]
Reynold:        [illegible]
Reynold*:       [illegible]
W0:             [illegible]
Power:          [illegible] W
Efficiency:     [illegible] %
Friction Loss:  [illegible] W
Disc Number:    5
Total Volute Space:             0.0124 m
Total Disc Space:               0.0032 m
r0 is:          0.09428173264159682 m
Outer radius:   0.073 m
h is:           0.0102144 m
Angle:          8.809972573699244 dg from tangent line
R ratio:        0.3
Vr:             [illegible]             Vt:             [illegible]
V_tip:          [illegible]
Reynold:        [illegible]
Reynold*:       [illegible]
W0:             [illegible]
Power:          [illegible] W
Efficiency:     [illegible] %
Friction Loss:  [illegible] W
```

In [5]:

```python
'''Base Case Simulation'''

KJ, sol, rs = solutionGenerator(rotorInner, rotorOuter, discSpacing, discThickness, nDisc-1,
                voluteThickness, wallSpace, 0, wallDisplacement,
                TotalMassFlowRate, density, revPerMinute[0])

'''Relative Tangential & Radial Profile'''
def f(x, y, instance):
    nVal = 3*instance.Fpo - 1
    phiZ = ((nVal+1)/nVal)*(1-np.power((2*xPoints/instance.discSpacing),nVal))
    phiZ = np.reshape(phiZ, (1, len(phiZ)))
    return phiZ.transpose()*y
tangentialAve = sol[:,0]
xPoints = np.linspace(-KJ.discSpacing/2, KJ.discSpacing/2, 100)
yPoints = rs
Z1 = f(xPoints, tangentialAve, KJ).transpose()
xPoints, yPoints = np.meshgrid(xPoints, yPoints)

X1 = xPoints
Y1 = yPoints

radialAve = (1/rs)*KJ.vRadial/KJ.tipVelocity
xPoints = np.linspace(-KJ.discSpacing/2, KJ.discSpacing/2, 100)
Z2 = f(xPoints, radialAve, KJ).transpose()
X2 = xPoints
Y2 = yPoints

'''plots'''
fig = plt.figure()

spec = mpl.gridspec.GridSpec(ncols=2, nrows=2,
                        height_ratios=[1.5, 1])


ax0 = fig.add_subplot(spec[0], projection='3d')
ax1 = fig.add_subplot(spec[1], projection='3d')
ax10 = fig.add_subplot(spec[2])
ax11 = fig.add_subplot(spec[3])

ax0.plot_surface(X1, Z1, Y1, rstride=1, cstride=1,
                cmap='viridis', edgecolor='none')
ax0.set_xlabel("$\it{b}$ (m)", labelpad=10, fontsize = 15)
ax0.xaxis.set_ticks(np.linspace(-KJ.discSpacing/2, KJ.discSpacing/2, 5))
ax0.set_ylabel("$\^{W}$", labelpad=10, fontsize = 15)
ax0.set_zlabel(r"$\xi$", labelpad=10, fontsize = 15)
ax0.set_title('Tangential', fontsize = 25)
ax0.set_ylim(np.amax(Z1),0)
ax0.grid(False)

ax1.plot_surface(X2, Z2, Y2, rstride=1, cstride=1,
                cmap='viridis', edgecolor='none')
ax1.set_xlabel("$\it{b}$ (m)", labelpad=10, fontsize = 15)
ax1.xaxis.set_ticks(np.linspace(-KJ.discSpacing/2, KJ.discSpacing/2, 5))
ax1.set_ylabel(r"$V_{r0}$", labelpad=10, fontsize = 15)
ax1.set_zlabel(r"$\xi$", labelpad=10, fontsize = 15)
ax1.set_title('Radial', fontsize = 25)
ax1.grid(False)

ax10.plot(tangentialAve, rs, color="black")
ax10.invert_xaxis()
ax10.set_ylabel(r"$\xi$", labelpad=10, fontsize = 15)
ax10.set_xlabel("$\^{W}$", labelpad=10, fontsize = 15)
ax10.set_xlim(right=0)
ax10.set_box_aspect(1)
ax10.grid()

ax11.plot(radialAve, rs, color="black")
ax11.set_ylabel(r"$\xi$", labelpad=10, fontsize = 15)
ax11.set_xlabel(r"$V_{r0}$", labelpad=10, fontsize = 15)
ax11.set_box_aspect(1)
ax11.set_xlim(right=0)
ax11.grid()
# ax.patch.set_facecolor('grey')
# fig.suptitle('Relative Velocity Profile', fontsize = 30)
```

In [4]:

```python
'''Base Case Simulation'''

KJ, sol, rs = solutionGenerator(rotorInner, rotorOuter, discSpacing, discThickness, nDisc-1,
                voluteThickness, wallSpace, 0, wallDisplacement,
                TotalMassFlowRate, density, revPerMinute[0])

'''Relative TangentialProfile'''
def f(x, y, instance):
    nVal = 3*instance.Fpo - 1
    phiZ = ((nVal+1)/nVal)*(1-np.power((2*xPoints/instance.discSpacing),nVal))
    phiZ = np.reshape(phiZ, (1, len(phiZ)))
    return phiZ.transpose()*y
tangentialAve = sol[:,0]
xPoints = np.linspace(-KJ.discSpacing/2, KJ.discSpacing/2, 100)
yPoints = rs
Z1 = f(xPoints, tangentialAve, KJ).transpose()
xPoints, yPoints = np.meshgrid(xPoints, yPoints)

X1 = xPoints
Y1 = yPoints

radialAve = (1/rs)*KJ.vRadial/KJ.tipVelocity
xPoints = np.linspace(-KJ.discSpacing/2, KJ.discSpacing/2, 100)
Z2 = f(xPoints, radialAve, KJ).transpose()
X2 = xPoints
Y2 = yPoints

'''plots'''
fig = plt.figure()

spec = mpl.gridspec.GridSpec(ncols=1, nrows=2,
                        height_ratios=[1.5, 1])


ax0 = fig.add_subplot(spec[0], projection='3d')
# ax1 = fig.add_subplot(spec[1], projection='3d')
ax10 = fig.add_subplot(spec[1])
# ax11 = fig.add_subplot(spec[3])


ax0.plot_surface(X1, Z1, Y1, rstride=1, cstride=1,
                cmap='viridis', edgecolor='none')
ax0.xaxis.set_ticks(np.linspace(-KJ.discSpacing/2, KJ.discSpacing/2, 5))
ax0.set_ylim(np.amax(Z1),0)
ax0.grid(False)

ax10.plot(tangentialAve, rs, color="black")
ax10.invert_xaxis()
ax10.set_xlim(right=0)
ax10.set_box_aspect(1)
ax10.grid()

# ax.patch.set_facecolor('grey')
fig.suptitle('Relative Tangential Profile', fontsize = 30)
```

Out[4]:

```
Text(0.5, 0.98, 'Relative Tangential Profile')
```

In [12]:

```python
'''Base Case Simulation'''

KJ, sol, rs = solutionGenerator(rotorInner, rotorOuter, discSpacing, discThickness, nDisc-1,
                voluteThickness, wallSpace, 0, wallDisplacement,
                TotalMassFlowRate, density, revPerMinute[0])
'''Pressure Drop Plots'''
fig,ax = plt.subplots()
ax.plot(rs, sol[:, 1], '--', color='black')
ax.invert_xaxis()
ax.set_ylabel("$\^{P}$", labelpad=10, fontsize = 15)
ax.set_ylim(top=0)
ax.set_xlabel(r"$\xi$", fontsize=15)
ax.set_xlim(left=rs[0])
ax.set_title('Dimensionless Pressure Drop', fontsize=30)

columns = {'rs': rs, 'dP': np.squeeze(sol[:,-1])}
df = pd.DataFrame(data=columns)
# df.to_excel("pressureDropDimensionless.xlsx")
```

In [5]:

```python
'''Base Case Simulation'''

KJ, sol, rs = solutionGenerator(rotorInner, rotorOuter, discSpacing, discThickness, nDisc-1,
                voluteThickness, wallSpace, 0, wallDisplacement,
                TotalMassFlowRate, density, revPerMinute[0])

"""Angle relative to spinning disc"""
radialPr = abs((1/rs)*KJ.relativeTipRadial)
tangentialPr = sol[:,0]
angleProfile = np.arctan(tangentialPr/ radialPr)*180/pi
fig,ax = plt.subplots(1, 2, gridspec_kw={
                        'width_ratios': [1, 1.25]})
ax[0].plot(rs, angleProfile, '--', color='black')

ax[0].set_xlabel(r"$\xi$", fontsize=15)
ax[0].set_xlim(left=rs[-1], right=rs[0])
ax[0].set_ylabel('Angle (\N{DEGREE SIGN})', fontsize=15)

radialPr = abs((1/rs)*KJ.vRadial/KJ.tipVelocity)
tangentialPr = sol[:,0]
angleProfile = np.arctan(tangentialPr/ radialPr)*180/pi

'''Unit Vector Case'''
startX = 2.5
for i in range(len(angleProfile)):
    if i%10 == 0:
        currentAngle = angleProfile[i]
        vectorToX, vectorToY = sin(currentAngle*pi/180)/5, -cos(currentAngle*pi/180)/5
        ax[1].arrow(startX, rs[i], -vectorToX, vectorToY, color="crimson", shape="full",
head_width=0.025)

ax[1].plot([startX, startX],[rs[-1], rs[0]], "--", color="black")

ax[1].plot([0, 5],[rs[-1], rs[-1]], ":", color="black")
ax[1].plot([0, 5],[rs[0], rs[0]], ":", color="black")
ax[1].set_ylabel(r"$\xi$", fontsize=15)
ax[1].axes.xaxis.set_visible(False)

theta = np.linspace(0, np.pi, 200)

r1 = rs[-1]
x1 = r1*np.cos(theta) + startX
y1 = r1*np.sin(theta)
ax[1].plot(x1, y1, color="black")

r2 = rs[0]
x2 = r2*np.cos(theta) + startX
y2 = r2*np.sin(theta)
ax[1].plot(x2, y2, color="black")

inner = rs[-1]
xs = np.linspace(-rs[0]+startX,rs[0]+startX, 201)
ys = np.linspace(-rs[0],rs[0], 201)
xv,yv = np.meshgrid(xs,ys)
r = (xv-startX)**2 + yv**2
ax[1].contourf(xv, yv, r, levels=[inner**2, rs[0]**2], colors="grey")

ax[1].set_xlim(left=1.5, right=3.5)
ax[1].set_ylim(bottom=0, top=rs[0]+0.1)
ax[1].set_aspect(1)

fig.suptitle('Velocity Angle', fontsize = 30)
```

Out[5]:

```
Text(0.5, 0.98, 'Velocity Angle')
```

In [15]:

```python
'''Base Case Simulation'''
'''Pressure Drop Contour'''

KJ, solKJ, rsKJ = solutionGenerator(rotorInner, rotorOuter, discSpacing, discThickness, nDisc-1,
                voluteThickness, wallSpace, 0, wallDisplacement,
                TotalMassFlowRate, density, revPerMinute[0])

'''Pressure Drop Contour'''
xs = np.linspace(0, 2*pi, 100)
rInner = np.squeeze(np.full((1,100), rsKJ[-1]))
rOuter = np.squeeze(np.full((1,100), rsKJ[0]))

thetaRange = np.linspace(0, 2*pi, 6)
fig, ax = plt.subplots(subplot_kw={'projection': 'polar'})
ax.plot(xs, rInner, color="black")
ax.plot(xs, rOuter, color="black")
ax.grid(False, axis='x')
ax.tick_params(axis='x', labelsize=15)
ax.tick_params(axis='y', labelsize=15)
ax.set_title('Pressure Drop Contour', fontsize = 30, pad=10)
ax.set_ylim(0, rsKJ[0])

thetas = np.linspace(0, 2*pi, 201)
radials = rsKJ
xv, yv = np.meshgrid(thetas, radials)

pressureContour = []
pDropBar = -1e-5*(0.5*density*KJ.tipVelocity**2)*solKJ[:,1]
for i in range(len(thetas)):
    pressureContour.append(pDropBar)
pressureContour = np.transpose(pressureContour)
highP, lowP = np.amax(pressureContour), np.amin(pressureContour)
# levels = np.linspace(lowP,highP,10)

cs = ax.contourf(xv, yv, pressureContour,  cmap= 'YlOrRd_r')
cbar = plt.colorbar(cs,ax=ax)
cbar.ax.set_ylabel('Pressure Drop (bar)', fontsize=25, labelpad=25)
cbar.ax.tick_params(labelsize=20)
plt.axis('off')
```

Out[15]:

```
(0.0, 6.283185307179586, 0.0, 1.0)
```

In [2]:

```python
'''Power Countour'''
for n in range(4):

    maxRotorOuter = 0.164
    kFactorRange = np.linspace(1,5,50)

    nDiscRange = np.arange(1,17,1)
    discSpacingRange = np.arange(0.0002,0.001,0.0001)
    rpmRange = np.linspace(50,5000,100)

    bSpacing = discSpacingRange[0] # choosing smallest possible disc spacing
    hiOutput = 0

    powerStorageStore = []
    bestLine = []
    for j in range(len(nDiscRange)):
        powerStorage = np.zeros([len(rpmRange),len(kFactorRange)])
        X,Y = np.meshgrid(rpmRange,kFactorRange)

        discNumberSpacing = nDiscRange[j]

        for l in range(len(kFactorRange)):
            highRPM, highPower = 0, 0
            for k in range(len(rpmRange)):
                effectiveRPM = rpmRange[k]
                maxRotorOuterCase = maxRotorOuter/kFactorRange[l]
                KJ, sol, rs = solutionGenerator(0.3*maxRotorOuterCase, maxRotorOuterCase,
bSpacing, \
                                discThickness, discNumberSpacing, voluteThickness, wallSpace, 0,
wallDisplacement,
                                 TotalMassFlowRate, density, effectiveRPM, profileN=2*(n+1))

                powerStorage[k,l] = power(sol[:,0], rs, KJ)

                '''Base Case'''
                if powerStorage[k,l]>highPower and discNumberSpacing==4:
                    highRPM, highPower = effectiveRPM, powerStorage[k,l]
                if powerStorage[k,l]>hiOutput:
                    hiOutput = powerStorage[k,l]
            if highPower == 0 or highRPM == 5000:
                pass
            else:
                bestLine.append([highRPM, kFactorRange[l]])
        powerStorageStore.append(powerStorage)


    X,Y = np.meshgrid(rpmRange,kFactorRange)
    powerStorageStore = np.array(powerStorageStore)
    bestLine = np.array(bestLine)

    '''Plots'''

    plt.rcParams["figure.figsize"]=12, 12
    fig, axs = plt.subplots(4,4)
    # levels = np.arange(0,int(hiOutput)+1,1)
    levels = np.arange(0,301,5)
    countX, countY = 0,0
    for i in range(len(powerStorageStore)):
        if(countY==4):
            countX+=1
            countY=0
        cs = axs[countX,countY].contourf(X,Y,powerStorageStore[i].transpose(),levels=levels,\
                                         extend='both',cmap="OrRd")
        axs[countX,countY].set_title(f'n = {i+2}',color='white',fontsize=15)
        axs[countX,countY].tick_params(axis='x', colors='white',labelsize=12)
        axs[countX,countY].tick_params(axis='y', colors='white',labelsize=12)
        if countX==0 and countY==3:
            axs[countX,countY].plot(bestLine[:,0], bestLine[:,1], "-.", color="black")
            axs[countX,countY].plot([rpmRange[0],rpmRange[-1]],
                                    [chosenScaleDownFactor,chosenScaleDownFactor], ":",
color="black")
            axs[countX,countY].grid(color='white')
        countY+=1

    plt.subplots_adjust(wspace = .2)
    fig.add_subplot(111, frame_on=False)
    fig.suptitle("Power Output Contour", color="white", fontsize=30, x=0.45)

    plt.tick_params(labelcolor="none", bottom=False, left=False)
    plt.xlabel("RPM", fontsize=25, color='white', position=(0.4, 0.5), labelpad=15)
    plt.ylabel("Scale Down Factor", fontsize=25, color='white', position=(0.4, 0.5),
labelpad=15)

    fig.tight_layout()
    cbar = plt.colorbar(cs,ax=axs)
    cbar.ax.set_ylabel('Power (W)', fontsize=25)
    cbar.ax.yaxis.label.set_color('white')
    cbar.ax.tick_params(axis='y', colors='white')

    ticklabs = cbar.ax.get_yticklabels()
    cbar.ax.set_yticklabels(ticklabs, fontsize=15)

    fig.patch.set_facecolor('#373E4B')
```

In [9]:

```python
'''Power Contour for Fred'''

maxRotorOuter = 0.164
kFactorRange = np.linspace(1,5,50)

nDiscRange = np.arange(2,8,1)
discSpacingRange = np.arange(0.0002,0.001,0.0001)
rpmRange = np.linspace(50,5000,100)

bSpacing = discSpacingRange[0] # choosing smallest possible disc spacing
hiOutput = 0

powerStorageStore = []
bestLine = []
for j in range(len(nDiscRange)):
    powerStorage = np.zeros([len(rpmRange),len(kFactorRange)])
    X,Y = np.meshgrid(rpmRange,kFactorRange)

    discNumberSpacing = nDiscRange[j]

    for l in range(len(kFactorRange)):
        highRPM, highPower = 0, 0
        for k in range(len(rpmRange)):
            effectiveRPM = rpmRange[k]
            maxRotorOuterCase = maxRotorOuter/kFactorRange[l]
            KJ, sol, rs = solutionGenerator(0.3*maxRotorOuterCase, maxRotorOuterCase, bSpacing,
\
                            discThickness, discNumberSpacing, voluteThickness, wallSpace, 0,
wallDisplacement,
                             TotalMassFlowRate, density, effectiveRPM)

            powerStorage[k,l] = power(sol[:,0], rs, KJ)

            '''Base Case'''
            if powerStorage[k,l]>highPower and discNumberSpacing==4:
                highRPM, highPower = effectiveRPM, powerStorage[k,l]
            if powerStorage[k,l]>hiOutput:
                hiOutput = powerStorage[k,l]
        if highPower == 0 or highRPM == 5000:
            pass
        else:
            bestLine.append([highRPM, kFactorRange[l]])
    powerStorageStore.append(powerStorage)


X,Y = np.meshgrid(rpmRange,kFactorRange)
powerStorageStore = np.array(powerStorageStore)
bestLine = np.array(bestLine)

'''Plots'''

plt.rcParams["figure.figsize"]=12, 12
# levels = np.arange(0,int(hiOutput)+1,1)
levels = np.arange(0,201,5)
for i in range(len(powerStorageStore)):
    fig, axs = plt.subplots()
    cs = axs.contourf(X,Y,powerStorageStore[i].transpose(),levels=levels,\
                                     extend='both',cmap="OrRd")
    axs.set_title(f'n = {i+2}',color='white',fontsize=15)
    axs.tick_params(axis='x', colors='white',labelsize=12)
    axs.tick_params(axis='y', colors='white',labelsize=12)
    plt.subplots_adjust(wspace = .2)
    fig.add_subplot(111, frame_on=False)
    fig.suptitle("Power Output Contour", color="white", fontsize=30, x=0.45)

    plt.tick_params(labelcolor="none", bottom=False, left=False)
    plt.xlabel("RPM", fontsize=25, color='white', position=(0.4, 0.5), labelpad=15)
    plt.ylabel(r"Scale Down Factor", fontsize=25, color='white', position=(0.4, 0.5),
labelpad=15)

    cbar = plt.colorbar(cs,ax=axs)
    cbar.ax.set_ylabel('Power (W)', fontsize=25)
    cbar.ax.yaxis.label.set_color('white')
    cbar.ax.tick_params(axis='y', colors='white')

    ticklabs = cbar.ax.get_yticklabels()
    cbar.ax.set_yticklabels(ticklabs, fontsize=15)

    fig.patch.set_facecolor('#373E4B')
    if i==3:
        fig1, axs1 = plt.subplots()
        cs = axs1.contourf(X,Y,powerStorageStore[i].transpose(),levels=levels,\
                                         extend='both',cmap="OrRd")
        axs1.plot(bestLine[:,0], bestLine[:,1], "-.", color="black")
        axs1.plot([rpmRange[0],rpmRange[-1]],
                                [chosenScaleDownFactor,chosenScaleDownFactor], ":",
color="black")
        axs1.grid(color='white')
        axs1.set_title(f'n = {i+2}',color='white',fontsize=15)
        axs1.tick_params(axis='x', colors='white',labelsize=12)
        axs1.tick_params(axis='y', colors='white',labelsize=12)
        plt.subplots_adjust(wspace = .2)
        fig1.add_subplot(111, frame_on=False)
        fig1.suptitle("Power Output Contour", color="white", fontsize=30, x=0.45)

        plt.tick_params(labelcolor="none", bottom=False, left=False)
        plt.xlabel("RPM", fontsize=25, color='white', position=(0.4, 0.5), labelpad=15)
        plt.ylabel(r"Scale Down Factor", fontsize=25, color='white', position=(0.4, 0.5),
labelpad=15)

        cbar1 = plt.colorbar(cs,ax=axs1)
        cbar1.ax.set_ylabel('Power (W)', fontsize=25)
        cbar1.ax.yaxis.label.set_color('white')
        cbar1.ax.tick_params(axis='y', colors='white')

        ticklabs = cbar.ax.get_yticklabels()
        cbar1.ax.set_yticklabels(ticklabs, fontsize=15)

        fig1.patch.set_facecolor('#373E4B')
```

In [3]:

```python
'''Efficiency Contour'''

for n in range(4):
    maxRotorOuter = 0.164
    kFactorRange = np.linspace(1,5,50)

    nDiscRange = np.arange(1,17,1)
    discSpacingRange = np.arange(0.0002,0.001,0.0001)
    rpmRange = np.linspace(50,5000,100)

    bSpacing = discSpacingRange[0] # choosing smallest possible disc spacing
    hiEfficiency = 0

    effStorageStore = []
    bestLine = []
    for j in range(len(nDiscRange)):
        effStorage = np.zeros([len(rpmRange),len(kFactorRange)])
        X,Y = np.meshgrid(rpmRange,kFactorRange)

        discNumberSpacing = nDiscRange[j]

        for l in range(len(kFactorRange)):
            highRPM, highEff = 0, 0
            for k in range(len(rpmRange)):
                effectiveRPM = rpmRange[k]
                maxRotorOuterCase = maxRotorOuter/kFactorRange[l]
                KJ, sol, rs = solutionGenerator(0.3*maxRotorOuterCase, maxRotorOuterCase,
bSpacing, \
                                discThickness, discNumberSpacing, voluteThickness, wallSpace, 0,
wallDisplacement,
                                 TotalMassFlowRate, density, effectiveRPM, profileN=2*(n+1))

                effStorage[k,l] = efficiencyIdeal(sol, rs, KJ)
                '''Base Case'''
                if effStorage[k,l]>highEff and discNumberSpacing==4:
                    highRPM, highEff = effectiveRPM, effStorage[k,l]
                if effStorage[k,l]>hiEfficiency:
                    hiEfficiency = effStorage[k,l]
            if highEff == 0 or highRPM == 5000:
                pass
            else:
                bestLine.append([highRPM, kFactorRange[l]])
        effStorageStore.append(effStorage)


    X,Y = np.meshgrid(rpmRange,kFactorRange)
    effStorageStore = np.array(effStorageStore)
    bestLine = np.array(bestLine)

    '''Plots'''

    plt.rcParams["figure.figsize"]=12, 12
    fig, axs = plt.subplots(4,4)
    levels = np.arange(0,int(hiEfficiency)+1,2.5)
    # levels = np.arange(0,101,5)
    countX, countY = 0,0
    for i in range(len(effStorageStore)):
        if(countY==4):
            countX+=1
            countY=0
        cs = axs[countX,countY].contourf(X,Y,effStorageStore[i].transpose(),levels=levels,\
                                         extend='both',cmap="summer_r")
        axs[countX,countY].set_title(f'n = {i+2}',color='white',fontsize=15)
        axs[countX,countY].tick_params(axis='x', colors='white',labelsize=12)
        axs[countX,countY].tick_params(axis='y', colors='white',labelsize=12)
        if countX==0 and countY==3:
            axs[countX,countY].plot(bestLine[:,0], bestLine[:,1], "-.", color="black")
            axs[countX,countY].plot([rpmRange[0],rpmRange[-1]],
                                    [chosenScaleDownFactor,chosenScaleDownFactor], ":",
color="black")
            axs[countX,countY].grid(color='white')
        countY+=1

    plt.subplots_adjust(wspace = .2)
    fig.add_subplot(111, frame_on=False)
    fig.suptitle("Efficiency Contour", color="white", fontsize=30, x=0.45)

    plt.tick_params(labelcolor="none", bottom=False, left=False)
    plt.xlabel("RPM", fontsize=25, color='white', position=(0.4, 0.5), labelpad=15)
    plt.ylabel("Scale Down Factor", fontsize=25, color='white', position=(0.4, 0.5),
labelpad=15)

    fig.tight_layout()
    cbar = plt.colorbar(cs,ax=axs)
    cbar.ax.set_ylabel('Effciency (%)', fontsize=25)
    cbar.ax.yaxis.label.set_color('white')
    cbar.ax.tick_params(axis='y', colors='white')

    ticklabs = cbar.ax.get_yticklabels()
    cbar.ax.set_yticklabels(ticklabs, fontsize=15)

    fig.patch.set_facecolor('#373E4B')
```

In [104…

```python
'''Mass Flow Rate Comparison on different application settings'''
inletArea = 0.0004

def damVelocity(height):
    g = 9.81
    c0 = sqrt(g*height)
    return 2*c0

def stream(hydraulicRadius, Slope, formulaType):
    u_m = 1
    u_c = 0.552
    manningN = 0.035
    chezyC = (u_m*hydraulicRadius**(1/6))/(u_c*manningN)
    if formulaType == "manning":
        velocity = (u_m/manningN)*(hydraulicRadius**(2/3))*(Slope**0.5)
    elif formulaType == "chezy":
        velocity = u_c*chezyC*(hydraulicRadius**0.5)*(Slope**0.5)
    else:
        raise Exception("Unidentified formula type")
    return velocity

smallDam = density*inletArea*damVelocity(10)
tapWater = 1
smallRiver = 0.7

hRadius = 0.5035345/pi*0.17526
river10 = stream(hRadius, tan(10*pi/180), "manning")
river20 = stream(hRadius, tan(20*pi/180), "manning")
river30 = stream(hRadius, tan(30*pi/180), "manning")
waterFall = 8.9
referenceStream = {
    "river": smallRiver,
    "river 10\N{DEGREE SIGN}": river10,
    "river 20\N{DEGREE SIGN}": river20,
    "river 30\N{DEGREE SIGN}": river30,
    "tap water": tapWater,
    "small dam": smallDam,
    "waterfall": waterFall
}
massFlowRateRange = np.arange(0.5, 3, 0.1)
rpmRange = np.linspace(50,5000,100)

'''Base Case'''
voluteThickness = 0.005
discThickness = 0.0008
discSpacing = 0.0002
wallSpace = 0.001
wallDisplacement = 0.003

nDiscBase = 5
rotorOuter = 0.073
rotorInner = 0.3*rotorOuter

hiOutput = 0

powerStorageStore = []

for i in range(len(rpmRange)):
    powerStorageStore.append([])
    effectiveRPM = rpmRange[i]
    for j in range(len(massFlowRateRange)):
        currentMFR = massFlowRateRange[j]
        KJ, sol, rs = solutionGenerator(rotorInner, rotorOuter, discSpacing, \
                        discThickness, nDiscBase - 1, voluteThickness, wallSpace, 0,
wallDisplacement,
                         currentMFR, density, effectiveRPM)

        currentOutput = power(sol[:,0], rs, KJ)
        powerStorageStore[i].append(currentOutput)
        if currentOutput > hiOutput:
            hiOutput = currentOutput
X,Y = np.meshgrid(rpmRange,massFlowRateRange)
powerStorageStore = np.array(powerStorageStore)

'''Plots'''
contourLevel = int(round((massFlowRateRange[-1]-massFlowRateRange[0])/1.5, 0)*50)
fig, axs = plt.subplots(figsize=(15,15))
levels = np.arange(0, int(hiOutput)+1, contourLevel)
# levels = np.arange(0,101,5)
cs = axs.contourf(X,Y,powerStorageStore.transpose(),levels=levels,\
                                 extend='both',cmap="OrRd")
for i in list(referenceStream.keys()):
    if referenceStream[i] >= massFlowRateRange[-1]:
        pass
    else:
        axs.plot([rpmRange[0], rpmRange[-1]], [referenceStream[i],referenceStream[i]],'--',
color='black')
        axs.text(rpmRange[5], referenceStream[i] + 0.025, f'{i}', color="Red", fontsize=15)
axs.set_title("Performance Matrix",color='white',fontsize=30, pad=20)

axs.set_xlabel('RPM', color='white', fontsize=25)
axs.tick_params(axis='x', colors='white')
axs.set_ylabel('Mass Flow Rate (kg/s)', color='white', fontsize=25)
axs.tick_params(axis='y', colors='white')
axs.set_ylim(top=massFlowRateRange[-1], bottom=massFlowRateRange[0])
axs.set_xlim(left=rpmRange[0], right=rpmRange[-1])

cbar = plt.colorbar(cs,ax=axs)
cbar.ax.set_ylabel('Power (W)', fontsize=25)
cbar.ax.yaxis.label.set_color('white')
cbar.ax.tick_params(axis='y', colors='white')

ticklabs = cbar.ax.get_yticklabels()
cbar.ax.set_yticklabels(ticklabs, fontsize=25)
fig.patch.set_facecolor('#373E4B')
```

In [7]:

```python
'''upScaling & PowerRatio'''
'''every aspect set as constant except radius and disc spacing'''
defaultRPM = np.array(revPerMinute)
nomKJ, nomSol, nomRs = solutionGenerator(rotorInner, rotorOuter, discSpacing, \
                        discThickness, nDisc - 1, voluteThickness, wallSpace, 0,
wallDisplacement,
                         TotalMassFlowRate, density, defaultRPM[0])
nominalPower = power(nomSol[:,0], nomRs, nomKJ)

kScale = np.arange(0.0,1.0,0.2)
rScale = np.arange(0.31, 1.5, 0.01)

scalingNominalPowerStorage = []
scalingMechanicalEff = []
scalingIdealEff = []
for i in range(len(kScale)):
    scalingNominalPowerStorageTemp = []
    scalingMechanicalEffTemp = []
    scalingIdealEffTemp = []
    for j in range(len(rScale)):
        currentRotorOuter = rotorOuter*rScale[j]
        currentRotorInner = rotorInner
        currentDiscSpacing = (rScale[j]**kScale[i])*discSpacing

        KJ, sol, rs = solutionGenerator(currentRotorInner, currentRotorOuter,
currentDiscSpacing, \
                        discThickness, nDisc - 1, voluteThickness, wallSpace, 0,
wallDisplacement,
                         TotalMassFlowRate, density, defaultRPM[0])

        currentRotorEff = rotorEff(sol[:,0], rs, KJ)
        scalingMechanicalEffTemp.append(currentRotorEff)

        currentIdealEff = efficiencyIdeal(sol, rs, KJ)
        scalingIdealEffTemp.append(currentIdealEff)

        currentPowerOut = power(sol[:,0], rs, KJ)
        powerRatio = currentPowerOut/nominalPower
        scalingNominalPowerStorageTemp.append(powerRatio)
    scalingIdealEff.append(scalingIdealEffTemp)
    scalingMechanicalEff.append(scalingMechanicalEffTemp)
    scalingNominalPowerStorage.append(scalingNominalPowerStorageTemp)

scalingIdealEff = np.array(scalingIdealEff)
scalingMechanicalEff = 100*np.array(scalingMechanicalEff) # convert to percentage format
scalingNominalPowerStorage = np.array(scalingNominalPowerStorage)

fig,ax = plt.subplots(figsize=(15,15))
for i in range(len(scalingNominalPowerStorage)):
    ax.plot(rScale, scalingNominalPowerStorage[i], label=f'k = {round(kScale[i],2)}')
ax.set_xlabel('$r_scale$', color='white', fontsize=25)
ax.set_ylabel('Normalised Power Output', color='white', fontsize=25)
ax.tick_params(axis='x', colors='white', labelsize=15)
ax.tick_params(axis='y', colors='white', labelsize=15)
plt.legend(fontsize=15)

fig1,ax1 = plt.subplots(figsize=(15,15))
for i in range(len(scalingMechanicalEff)):
    ax1.plot(rScale, scalingMechanicalEff[i], label=f'k = {round(kScale[i],2)}')
ax1.set_xlabel('$r_0$ Scale', color='white', fontsize=25)
ax1.set_ylabel('Mechanical Efficiency (%)', color='white', fontsize=25)
ax1.set_ylim(top=80, bottom=30)
ax1.tick_params(axis='x', colors='white', labelsize=15)
ax1.tick_params(axis='y', colors='white', labelsize=15)
plt.legend(fontsize=15)

fig2,ax2 = plt.subplots(figsize=(15,15))
for i in range(len(scalingIdealEff)):
    ax2.plot(rScale, scalingIdealEff[i], label=f'k = {round(kScale[i],2)}')
ax2.set_xlabel('$r_{scale}$', color='white', fontsize=25)
ax2.set_ylabel('Ideal Efficiency (%)', color='white', fontsize=25)
ax2.tick_params(axis='x', colors='white', labelsize=15)
ax2.tick_params(axis='y', colors='white', labelsize=15)
plt.legend(fontsize=15)

ax.grid(ls=":")
ax1.grid(ls=":")
ax2.grid(ls=":")

fig.patch.set_facecolor('#373E4B')
fig1.patch.set_facecolor('#373E4B')
fig2.patch.set_facecolor('#373E4B')
```

In [8]:

```python
'''upScaling & powerDensity'''
'''every aspect set as constant except radius and disc spacing'''
kScale = np.array([0.0,0.15,0.33,0.5,1.0])
rScale = np.arange(0.31, 1.5, 0.01)

scalingPowerDensityStorage = []
scalingPowerStorage = []
scalingVolumeStorage = []
defaultRPM = np.array(revPerMinute)

for i in range(len(kScale)):
    scalingPowerDensityStorageTemp = []
    scalingPowerStorageTemp = []
    scalingVolumeStorageTemp = []
    for j in range(len(rScale)):
        currentRotorOuter = rotorOuter*rScale[j]
        currentRotorInner = rotorInner # rotor inner set as constant
        currentDiscSpacing = (rScale[j]**kScale[i])*discSpacing

        KJ, sol, rs = solutionGenerator(currentRotorInner, currentRotorOuter,
currentDiscSpacing, \
                        discThickness, nDisc - 1, voluteThickness, wallSpace, 0,
wallDisplacement,
                         TotalMassFlowRate, density, defaultRPM[0])

        currentPowerOut = power(sol[:,0], rs, KJ)
        scalingPowerStorageTemp.append(currentPowerOut)
        currentVolume = (nDisc - 1)*(pi*currentRotorOuter**2)*currentDiscSpacing # 4 signifies
the number spacing
        scalingVolumeStorageTemp.append(currentVolume)
        scalingPowerDensityStorageTemp.append(currentPowerOut/currentVolume)
    scalingVolumeStorage.append(scalingVolumeStorageTemp)
    scalingPowerStorage.append(scalingPowerStorageTemp)
    scalingPowerDensityStorage.append(scalingPowerDensityStorageTemp)

'''Reference'''
KJ, sol, rs = solutionGenerator(rotorInner, rotorOuter, discSpacing, \
                        discThickness, nDisc - 1, voluteThickness, wallSpace, 0,
wallDisplacement,
                         TotalMassFlowRate, density, defaultRPM[0])
referencePowerOut = power(sol[:,0], rs, KJ)
referencePowerDensity = referencePowerOut/((nDisc-1)*(pi*rotorOuter**2)*discSpacing)
'''End'''

scalingVolumeStorage = np.array(scalingVolumeStorage)
scalingPowerStorage = np.array(scalingPowerStorage)
scalingPowerDensityStorage = np.array(scalingPowerDensityStorage)
scalingPowerDensityStorageRef = scalingPowerDensityStorage/referencePowerDensity

fig,ax = plt.subplots(figsize=(15,15))
for i in range(len(scalingPowerDensityStorageRef)):
    ax.plot(rScale, scalingPowerDensityStorageRef[i], label=f'k = {kScale[i]}')
ax.set_xlabel('$r_{scale}$', color='white', fontsize=25)
ax.set_ylabel('Normalised Power Density', color='white', fontsize=25)
ax.tick_params(axis='x', colors='white', labelsize=15)
ax.tick_params(axis='y', colors='white', labelsize=15)
ax.set_yscale('log')
ax.legend(fontsize=15, loc="upper right")
ax.grid(True, which="both", ls=":")
fig.patch.set_facecolor('#373E4B')
"""
fig2,ax2 = plt.subplots(figsize=(15,15))
for i in range(len(scalingPowerDensityStorage)):
    ax2.plot(scalingVolumeStorage[i], scalingPowerDensityStorage[i], label=f'k = {kScale[i]}')
ax2.set_xlabel('volume (apr)', color='white', fontsize=40)
ax2.set_ylabel('power density (W/m3)', color='white', fontsize=40)
ax2.xaxis.set_major_formatter(formatter)
ax2.tick_params(axis='x', colors='white', labelsize=40)
ax2.tick_params(axis='y', colors='white', labelsize=40)
ax2.set_yscale('log')
ax2.legend(fontsize=40, loc="upper right")
"""
```

Out[8]:

```
'\nfig2,ax2 = plt.subplots(figsize=(15,15))\nfor i in range(len(scalingPowerDensityStorage)):\n    ax2.plot(scalingVolumeStorage[i], scalingPowerDensityStorage[i], label=f\'k = {kScale[i]}\')\nax2.set_xlabel(\'volume (apr)\', color=\'white\', fontsize=40)\nax2.set_ylabel(\'power density (W/m3)\', color=\'white\', fontsize=40)\nax2.xaxis.set_major_formatter(formatter)\nax2.tick_params(axis=\'x\', colors=\'white\', labelsize=40)\nax2.tick_params(axis=\'y\', colors=\'white\', labelsize=40)\nax2.set_yscale(\'log\')\nax2.legend(fontsize=40, loc="upper right")\n'
```

In [136…]:

```python
"""df = pd.read_excel("E:\\UoS\\UoS Y4\\GDP Tesla Turbine\\CFD\\fluentData.xlsx")

cfdRadius = np.array(df["Radius"])
cfdRadiusRatio = cfdRadius/rotorOuter

firstTurbGap = np.array(df["1st gap turb"]) - np.amax(np.array(df["1st gap turb"]))
secondTurbGap = np.array(df["2nd gap turb"]) - np.amax(np.array(df["2nd gap turb"]))
thirdTurbGap = np.array(df["3rd gap turb"]) - np.amax(np.array(df["3rd gap turb"]))
fourthTurbGap = np.array(df["4th gap turb"]) - np.amax(np.array(df["4th gap turb"]))
aveTurbGap = np.mean(np.array([firstTurbGap, secondTurbGap, thirdTurbGap, fourthTurbGap]), 
axis=0)
aveTurbGap = aveTurbGap - np.amax(aveTurbGap)

firstLamGap = np.array(df["1st gap lam"]) - np.amax(np.array(df["1st gap lam"]))
secondLambGap = np.array(df["2nd gap lam"]) - np.amax(np.array(df["2nd gap lam"]))
thirdLamGap = np.array(df["3rd gap lam"]) - np.amax(np.array(df["3rd gap lam"]))
fourthLambGap = np.array(df["4th gap lam"]) - np.amax(np.array(df["4th gap lam"]))
aveLamGap = np.mean(np.array([firstLamGap, secondLambGap, thirdLamGap, fourthLambGap]), axis=0)
aveLamGap = aveLamGap - np.amax(aveLamGap)

profileNFactor = [2,4,6,8]
for i in range(len(profileNFactor)):

    KJ, sol, rs = solutionGenerator(rotorInner, rotorOuter, discSpacing, discThickness, nDisc-1, 
                    voluteThickness, wallSpace, 0, wallDisplacement, 
                    TotalMassFlowRate, density, revPerMinute[0], profileN = profileNFactor[i])

    fig, ax = plt.subplots(1,2, figsize=(15,6))
    suptitleString = f"profile N = {profileNFactor[i]}"
    plt.subplots_adjust(left = 0.025)
    unnormalisedPressure = sol[:,1]*(density/2)*(KJ.tipVelocity)**2
    ax[0].plot(rs, unnormalisedPressure, label="Numerical")

    if i==0:
        fig.suptitle(suptitleString + " (Laminar)", x=0.45, fontsize=20, color="white")
        try:
            ax[0].plot(cfdRadiusRatio, firstLamGap, label="1st Gap")
            ax[0].plot(cfdRadiusRatio, secondLambGap, label="2nd Gap")
            ax[0].plot(cfdRadiusRatio, thirdLamGap, label="3rd Gap")
            ax[0].plot(cfdRadiusRatio, fourthLambGap, label="4th Gap")
            ax[0].plot(cfdRadiusRatio, aveLamGap, label="CFD Average")
        except:
            pass
    else:
        fig.suptitle(suptitleString + " (Turbulent)", x=0.45, fontsize=20, color="white")
        try:
            ax[0].plot(cfdRadiusRatio, firstTurbGap, label="1st Gap")
            ax[0].plot(cfdRadiusRatio, secondTurbGap, label="2nd Gap")
            ax[0].plot(cfdRadiusRatio, thirdTurbGap, label="3rd Gap")
            ax[0].plot(cfdRadiusRatio, fourthTurbGap, label="4th Gap")
            ax[0].plot(cfdRadiusRatio, aveTurbGap, label="CFD Average")
        except:
            pass
    ax[0].yaxis.set_major_formatter(formatter)
    ax[0].set_ylabel('Static pressure drop (Pa)', color='white')
    ax[0].set_xlabel('Radius Ratio', color='white')
    ax[0].legend(loc="upper right")

    ax[0].tick_params(axis='x', colors='white')
    ax[0].tick_params(axis='y', colors='white')

    profilePlot(ax[1],KJ, 50)
    ax[1].tick_params(axis='x', colors='white')
    ax[1].tick_params(axis='y', colors='white')

    plt.show()
    print(
        f"Power:\t\t{power(sol[:,0], rs, KJ)} W"+"\n"
    )"""
```

Out[136…]:

```
'df = pd.read_excel("E:\\\\UoS\\\\UoS Y4\\\\GDP Tesla Turbine\\\\CFD\\\\fluentData.xlsx")\\n\\ncfdRadius = np.array(df["Rad
ius"])\\ncfdRadiusRatio = cfdRadius/rotorOuter\\n\\nfirstTurbGap = np.array(df["1st gap turb"]) - np.amax(np.array
(df["1st gap turb"]))\\nsecondTurbGap = np.array(df["2nd gap turb"]) - np.amax(np.array(df["2nd gap turb"]))\\nth
irdTurbGap = np.array(df["3rd gap turb"]) - np.amax(np.array(df["3rd gap turb"]))\\nfourthTurbGap = np.array(df
["4th gap turb"]) - np.amax(np.array(df["4th gap turb"]))\\naveTurbGap = np.mean(np.array([firstTurbGap, secondT
urbGap, thirdTurbGap, fourthTurbGap]), axis=0)\\naveTurbGap = aveTurbGap - np.amax(aveTurbGap)\\n\\nfirstLamGap = 
np.array(df["1st gap lam"]) - np.amax(np.array(df["1st gap lam"]))\\nsecondLambGap = np.array(df["2nd gap lam"])
 - np.amax(np.array(df["2nd gap lam"]))\\nthirdLamGap = np.array(df["3rd gap lam"]) - np.amax(np.array(df["3rd ga
p lam"]))\\nfourthLambGap = np.array(df["4th gap lam"]) - np.amax(np.array(df["4th gap lam"]))\\naveLamGap = np.m
ean(np.array([firstLamGap, secondLambGap, thirdLamGap, fourthLambGap]), axis=0)\\naveLamGap = aveLamGap - np.ama
x(aveLamGap)\\n\\nprofileNFactor = [2,4,6,8]\\nfor i in range(len(profileNFactor)):\\n\\n    KJ, sol, rs = solutionG
enerator(rotorInner, rotorOuter, discSpacing, discThickness, nDisc-1,\\n                    voluteThickness, wa
llSpace, 0, wallDisplacement,\\n                    TotalMassFlowRate, density, revPerMinute[0], profileN = pro
fileNFactor[i])\\n\\n    fig, ax = plt.subplots(1,2, figsize=(15,6))\\n    suptitleString = f"profile N = {pro
fileNFactor[i]}"\\n    plt.subplots_adjust(left = 0.025)\\n    unnormalisedPressure = sol[:,1]*(density/2)*(KJ.ti
pVelocity)**2\\n    ax[0].plot(rs, unnormalisedPressure, label="Numerical")\\n\\n    if i==0:\\n        fig.sup
title(suptitleString + " (Laminar)", x=0.45, fontsize=20, color="white")\\n        try:\\n            ax[0].plot
(cfdRadiusRatio, firstLamGap, label="1st Gap")\\n            ax[0].plot(cfdRadiusRatio, secondLambGap, label="2n
d Gap")\\n            ax[0].plot(cfdRadiusRatio, thirdLamGap, label="3rd Gap")\\n            ax[0].plot(cfdRadius
Ratio, fourthLambGap, label="4th Gap")\\n            ax[0].plot(cfdRadiusRatio, aveLamGap, label="CFD Average")
\\n        except:\\n            pass\\n    else:\\n        fig.suptitle(suptitleString + " (Turbulent)", x=0.45, f
ontsize=20, color="white")\\n        try:\\n            ax[0].plot(cfdRadiusRatio, firstTurbGap, label="1st Gap")
\\n            ax[0].plot(cfdRadiusRatio, secondTurbGap, label="2nd Gap")\\n            ax[0].plot(cfdRadiusRati
o, thirdTurbGap, label="3rd Gap")\\n            ax[0].plot(cfdRadiusRatio, fourthTurbGap, label="4th Gap")\\n 
           ax[0].plot(cfdRadiusRatio, aveTurbGap, label="CFD Average")\\n        except:\\n            pass\\n    ax
[0].yaxis.set_major_formatter(formatter)\\n    ax[0].set_ylabel(\\'Static pressure drop (Pa)\\', color=\\'white\\')\\
n    ax[0].set_xlabel(\\'Radius Ratio\\', color=\\'white\\')\\n    ax[0].legend(loc="upper right")\\n\\n    ax
[0].tick_params(axis=\\'x\\', colors=\\'white\\')\\n    ax[0].tick_params(axis=\\'y\\', colors=\\'white\\')\\n\\n    p
rofilePlot(ax[1],KJ, 50)\\n    ax[1].tick_params(axis=\\'x\\', colors=\\'white\\')\\n    ax[1].tick_params(axis=\\'y
\\', colors=\\'white\\')\\n\\n    plt.show()\\n    print(\\n        f"Power:\\t\\t{power(sol[:,0], rs, KJ)}
W"+"\\n"\\n    )'
```

In [4]:

```python
'''XY Tangential & Radial Flow Profile Validation _ 10 gaps'''
'''Base Case Simulation'''
KJ, solKJ, rsKJ = solutionGenerator(rotorInner, rotorOuter, discSpacing, discThickness, nDisc-1, 
                  voluteThickness, wallSpace, 0, wallDisplacement, 
                  TotalMassFlowRate, density, revPerMinute[0])
nProfile = 3*KJ.Fpo - 1
CFDNormPoint = 0.0015

VTangentialRake = rakeProcess("VTangential_10")
VRadialRake = rakeProcess("VRadial_10")
rakeNameList = ['rake-0','rake-1','rake-2','rake-3','rake-4','rake-5','rake-6','rake-7','rake-
8','rake-9']


zs = np.linspace(-discSpacing/2, discSpacing/2, 11)
normalisedZS = 2*zs/discSpacing

fig,ax = plt.subplots(2,1)

tangentialAverage = []
for i in range(len(rakeNameList)):
    rValueTaken = VTangentialRake[rakeNameList[i]][:,0][0]
    print(rValueTaken)
    currentRRatio = rValueTaken/KJ.outerRadius
    currentDiscSpeed = currentRRatio*KJ.tipVelocity

    yRaw = VTangentialRake[rakeNameList[i]][:,1] - currentDiscSpeed
    yFiltered = yRaw/max(abs(yRaw))

    # ax[0].plot(normalisedZS, yFiltered, label=rakeNameList[i])
    tangentialAverage.append(yFiltered)
tangentialAverage = np.mean(np.array(tangentialAverage), axis=0)
ax[0].plot(normalisedZS, tangentialAverage, label="CFD", linewidth=3, color="red")

radialAverage = []
for i in range(len(rakeNameList)):
    rValueTaken = VRadialRake[rakeNameList[i]][:,0][0]

    yRaw = VRadialRake[rakeNameList[i]][:,1]
    yFiltered = yRaw/max(abs(yRaw))

    # ax[1].plot(2*(VRadialRake[rakeNameList[i]][:,0]+CFDNormPoint)/KJ.discSpacing, yFiltered, 
label=rakeNameList[i])
    radialAverage.append(yFiltered)
radialAverage = np.mean(np.array(radialAverage), axis=0)
ax[1].plot(normalisedZS,radialAverage, label="CFD", linewidth=3, color="red")

'''Validation Line'''
refinedZS = np.linspace(-discSpacing/2, discSpacing/2, 100)
normalisedRefinedZS = 2*refinedZS/discSpacing

nProfileList={
    2:[(1-np.power((2*refinedZS/KJ.discSpacing),2)), "-"],
    4:[(1-np.power((2*refinedZS/KJ.discSpacing),4)), "--"],
    6:[(1-np.power((2*refinedZS/KJ.discSpacing),6)), "-."],
    8:[(1-np.power((2*refinedZS/KJ.discSpacing),8)), ":"]
}

for i in nProfileList:
    ax[0].plot(normalisedRefinedZS, nProfileList[i][0], nProfileList[i][1],\
            linewidth=1.5, color="black", label=f"n={i}")
    ax[1].plot(normalisedRefinedZS, -nProfileList[i][0], nProfileList[i][1],\
            linewidth=1.5, color="black", label=f"n={i}")


ax[0].set_xlabel('Dimensionless Disc Gap', fontsize=25)
ax[0].set_ylabel('Normalised Tangential', fontsize=25, labelpad=22)

ax[1].set_ylabel('Normalised Radial', fontsize=25, labelpad=10)

ax[0].tick_params(axis='x', labelsize=15)
ax[0].tick_params(axis='y', labelsize=15)
ax[0].set_xlim(right=normalisedZS[-1]+0.5, left=normalisedZS[0]-0.5)
ax[0].set_ylim(top=None,bottom=None)

ax[0].plot([-1, -1], [0, 11], ":", color="black")
ax[0].plot([1, 1], [0, 11], ":", color="black")

ax[1].tick_params(axis='x', labelsize=15)
ax[1].tick_params(axis='y', labelsize=15)
ax[1].set_xlim(right=normalisedZS[-1]+0.5, left=normalisedZS[0]-0.5)
ax[1].set_ylim(top=0,)

ax[1].plot([-1, -1], [-11, 0], ":", color="black")
ax[1].plot([1, 1], [-11, 0], ":", color="black")

handles, labels = ax[1].get_legend_handles_labels()
fig.legend(handles, labels, loc='right', fontsize=25)
plt.xlabel('Dimensionless Disc Gap', fontsize=25, position=(0.4, 0.5))
```

Out[4]: Text(0.4, 0.5, 'Dimensionless Disc Gap')

In [10]:

```python
'''XY Tangential & Radial Flow Profile Validation _ 20 gaps'''
'''Base Case Simulation'''
KJ, solKJ, rsKJ = solutionGenerator(rotorInner, rotorOuter, discSpacing, discThickness, nDisc-1, 
                  voluteThickness, wallSpace, 0, wallDisplacement, 
                  TotalMassFlowRate, density, revPerMinute[0])
nProfile = 3*KJ.Fpo - 1
CFDNormPoint = 0.0015

VTangentialRake = rakeProcess("VTangential_20")
VRadialRake = rakeProcess("VRadial_20")
rakeNameList = ['rake-0','rake-1','rake-2','rake-3','rake-4','rake-5','rake-6','rake-7','rake-
8','rake-9']


normalisedZS = 2*(VRadialRake[rakeNameList[0]][:,0]+CFDNormPoint)/KJ.discSpacing
fig,ax = plt.subplots(2,1)

rValueList = [
    0.0231,
    0.0286444,
    0.0341889,
    0.0397333,
    0.0452778,
    0.0508222,
    0.0563667,
    0.0619111,
    0.0674556,
    0.0730
]


tangentialAverage = []
for i in range(len(rakeNameList)):
    rValueTaken = rValueList[i]
    currentRRatio = rValueTaken/KJ.outerRadius
    currentDiscSpeed = currentRRatio*KJ.tipVelocity

    yRaw = VTangentialRake[rakeNameList[i]][:,1] - currentDiscSpeed
    yFiltered = yRaw/max(abs(yRaw))

    tangentialAverage.append(yFiltered)
tangentialAverage = np.mean(np.array(tangentialAverage), axis=0)
ax[0].plot(normalisedZS, tangentialAverage, label="CFD", linewidth=3, color="red")



radialAverage = []
for i in range(len(rakeNameList)):
    yRaw = VRadialRake[rakeNameList[i]][:,1]
    yFiltered = yRaw/max(abs(yRaw))

    radialAverage.append(yFiltered)
radialAverage = np.mean(np.array(radialAverage), axis=0)
ax[1].plot(normalisedZS,radialAverage, label="CFD", linewidth=3, color="red")

'''Validation Line'''
refinedZS = np.linspace(-discSpacing/2, discSpacing/2, 100)
normalisedRefinedZS = 2*refinedZS/discSpacing

nProfileList={
    2:[(1-np.power((2*refinedZS/KJ.discSpacing),2)), "-"],
    4:[(1-np.power((2*refinedZS/KJ.discSpacing),4)), "--"],
    6:[(1-np.power((2*refinedZS/KJ.discSpacing),6)), "-."],
    8:[(1-np.power((2*refinedZS/KJ.discSpacing),8)), ":"]
}

for i in nProfileList:
    ax[0].plot(normalisedRefinedZS, nProfileList[i][0], nProfileList[i][1],\
            linewidth=1.5, color="black", label=f"n={i}")
    ax[1].plot(normalisedRefinedZS, -nProfileList[i][0], nProfileList[i][1],\
            linewidth=1.5, color="black", label=f"n={i}")


ax[0].set_xlabel('Dimensionless Disc Gap', fontsize=25)
ax[0].set_ylabel('Normalised Tangential', fontsize=25, labelpad=22)

ax[1].set_ylabel('Normalised Radial', fontsize=25, labelpad=10)

ax[0].tick_params(axis='x', labelsize=15)
ax[0].tick_params(axis='y', labelsize=15)
ax[0].set_xlim(right=normalisedZS[-1]+0.5, left=normalisedZS[0]-0.5)
ax[0].set_ylim(top=None,bottom=None)

ax[0].plot([-1, -1], [0, 11], ":", color="black")
ax[0].plot([1, 1], [0, 11], ":", color="black")

ax[1].tick_params(axis='x', labelsize=15)
ax[1].tick_params(axis='y', labelsize=15)
ax[1].set_xlim(right=normalisedZS[-1]+0.5, left=normalisedZS[0]-0.5)
ax[1].set_ylim(top=0,)

ax[1].plot([-1, -1], [-11, 0], ":", color="black")
ax[1].plot([1, 1], [-11, 0], ":", color="black")

handles, labels = ax[1].get_legend_handles_labels()
fig.legend(handles, labels, loc='right', fontsize=25)
plt.xlabel('Dimensionless Disc Gap', fontsize=25, position=(0.4, 0.5))
```

Out[10]: Text(0.4, 0.5, 'Dimensionless Disc Gap')

In [15]:

```python
'''3D Relative Tangential Flow Profile Validation _ 20 gaps'''
'''Base Case Simulation'''
KJ, solKJ, rsKJ = solutionGenerator(rotorInner, rotorOuter, discSpacing, discThickness, nDisc-1, 
                  voluteThickness, wallSpace, 0, wallDisplacement, 
                  TotalMassFlowRate, density, revPerMinute[0])
nProfile = 3*KJ.Fpo - 1
CFDNormPoint = 0.0015

VTangentialRake = rakeProcess("VTangential_20")
rakeNameList = ['rake-0','rake-1','rake-2','rake-3','rake-4','rake-5','rake-6','rake-7','rake-
8','rake-9']

refinedZS = np.linspace(-discSpacing/2, discSpacing/2, 100)
normalisedZS = 2*(VTangentialRake[rakeNameList[0]][:,0]+CFDNormPoint)/KJ.discSpacing
normalisedRefinedZS = 2*refinedZS/discSpacing
```

```python
VRadialRake = rakeProcess("VRadial_20")
rakeNameList = ['rake-0','rake-1','rake-2','rake-3','rake-4','rake-5','rake-6','rake-7','rake-
8','rake-9']

refinedZS = np.linspace(-discSpacing/2, discSpacing/2, 100)
normalisedZS = 2*(VRadialRake[rakeNameList[0]][:,0]+CFDNormPoint)/KJ.discSpacing
normalisedRefinedZS = 2*refinedZS/discSpacing

rValueList = [
    0.0231,
    0.0286444,
    0.0341889,
    0.0397333,
    0.0452778,
    0.0508222,
    0.0563667,
    0.0619111,
    0.0674556,
    0.0730
]


nProfileList={
    2:[(1-np.power((2*refinedZS/KJ.discSpacing),2)), "-"],
    4:[(1-np.power((2*refinedZS/KJ.discSpacing),4)), "--"],
    6:[(1-np.power((2*refinedZS/KJ.discSpacing),6)), "-."],
    8:[(1-np.power((2*refinedZS/KJ.discSpacing),8)), ":"]
}

fig = plt.figure(constrained_layout=True)
gridSpace = fig.add_gridspec(4, 10)

ax0 = fig.add_subplot(gridSpace[1:4, 1:6], projection='3d')
ax1 = fig.add_subplot(gridSpace[1:4, 6:9])


radialAverage = []
for i in range(len(rakeNameList)):
    rValueTaken = rValueList[i]
    currentRRatio = rValueTaken/KJ.outerRadius
    yRaw = VRadialRake[rakeNameList[i]][:,1]
    yFiltered = yRaw/max(abs(yRaw))
    radialAverage.append(yFiltered)

    rPlotPoints = np.full(len(normalisedZS), currentRRatio)
    ax0.plot(normalisedZS, yRaw, rPlotPoints)
    ax1.plot(normalisedZS, yFiltered, label = f"r/ro = {round(currentRRatio, 3)}")

ax0.set_xlabel("2z/b", fontsize=15)
ax0.set_ylabel("Relative Velocity (m/s)", fontsize=15)
ax0.set_zlabel("r/ro", fontsize=15)

ax0.invert_xaxis()
ax0.grid(False)

ax1.set_xlabel("2z/b", fontsize=15)
ax1.set_ylabel("Dimensionless Velocity", fontsize=15)

handles, labels = ax1.get_legend_handles_labels()
fig.legend(handles, labels, loc='right', fontsize=15)

for i in nProfileList:
    ax1.plot(normalisedRefinedZS, -nProfileList[i][0], nProfileList[i][1],\
            linewidth=1.5, color="black", label=f"n={i}")
fig.suptitle("Radial Profile", fontsize = 20)
```

Out[18]: Text(0.5, 0.98, 'Radial Profile')

In [9]:

```python
A1, A2 = postProcess("VAverageRadial_20")[0][0],postProcess("VAverageRadial_20")[1][0]
B1, B2 = postProcess("VAverageTangential_20")[0][0],postProcess("VAverageTangential_20")[1][0]

A2=abs(A2)
B2 = B2-B1*KJ.omega*KJ.outerRadius
angleCFD = np.arctan(A2/B2)*180/pi


KJ, sol, rs = solutionGenerator(rotorInner, rotorOuter, discSpacing, discThickness, nDisc-1, 
                  voluteThickness, wallSpace, 0, wallDisplacement, 
                  TotalMassFlowRate, density, revPerMinute[0])

"""Angle relative to spinning disc"""
radialPr = abs(sol[:,3]*KJ.relativeTipRadial)
tangentialPr = sol[:,2]
angleProfile = np.arctan(tangentialPr/ radialPr)*180/pi
fig,ax = plt.subplots(1, 2)
ax[0].plot(rs, angleProfile, "--", color="black")

ax[0].set_xlabel("r/ro", fontsize=15)
ax[0].set_xlim(left=rs[0], right=rs[-1])
ax[0].set_ylabel("Angle ($\THEORDER $\\theta$ (Numerical)", fontsize=15)


ax[1].plot(A1, angleCFD, "--", color="black")

ax[1].set_xlabel("r/ro", fontsize=15)
ax[1].set_xlim(left=A1[0], right=A1[-1])
ax[1].set_ylabel("Angle ($\THEORDER $\\theta$ CFD)", fontsize=15)
ax[1].invert_xaxis()
fig.suptitle("Velocity Angle", fontsize = 20, x=0.5)
```

Out[9]: Text(0.5, 0.98, 'Velocity Angle')

In [8]:

```python
"""Base Case Simulation"""
KJ, solKJ, rsKJ = solutionGenerator(rotorInner, rotorOuter, discSpacing, discThickness, nDisc-1, 
                  voluteThickness, wallSpace, 0, wallDisplacement, 
                  TotalMassFlowRate, density, revPerMinute[0], profileN = 2)
print(f"{power(solKJ[:,0], rsKJ, KJ)} W")

rProfile, shearValue = postProcess("WallShear_10")[0][0],postProcess("WallShear_10")[1][0]
rProfile = rProfile[::-1]
shearValue = shearValue[::-1]

constantTerm = (KJ.outerRadius**3)*2*pi
integrateTerm = shearValue*rProfile**2
totalPower = 2*KJ.omega*KJ.numberSpacing*\
    constantTerm*scipy.integrate.simps(integrateTerm, x = rProfile)
print("power for mesh 10: ", totalPower)

rProfile, shearValue = postProcess("WallShear_20")[0][0],postProcess("WallShear_20")[1][0]
rProfile = rProfile[::-1]
shearValue = shearValue[::-1]

constantTerm = (KJ.outerRadius**3)*2*pi
integrateTerm = shearValue*rProfile**2
totalPower = 2*KJ.omega*KJ.numberSpacing*\
    constantTerm*scipy.integrate.simps(integrateTerm, x = rProfile)
print("power for mesh 20: ", totalPower)
```

In [6]:

```python
"""Base Case Simulation"""
KJ, solKJ, rsKJ = solutionGenerator(rotorInner, rotorOuter, discSpacing, discThickness, nDisc-1, 
                  voluteThickness, wallSpace, 0, wallDisplacement, 
                  TotalMassFlowRate, density, revPerMinute[0], profileN = 2)

VTangentialRake20 = rakeProcess("VRakeTangential_10")
CFDNormPoint = 0.0015

rakeList = []
rakeListName = []
rakeProfileXYList = []
for i in VTangentialRake20:
    keyName = i
    keyList = keyName.split("-")
    rakeName, radialRake = keyList[0], float(keyList[1])
    radialRake/=KJ.outerRadius
    currentDiscSpeed = radialRake*KJ.tipVelocity

    rakeListName.append(rakeName)
    rakeList.append(radialRake)
    #currentXY = 2*VTangentialRake20[keyName][:,0]-CFDNormPoint)/KJ.discSpacing,
    #            (VTangentialRake20[keyName][:,1]-currentDiscSpeed)/currentDiscSpeed
    currentXY = VTangentialRake20[keyName][:,0]+CFDNormPoint,\
                (VTangentialRake20[keyName][:,1]-currentDiscSpeed)/currentDiscSpeed
    rakeProfileXYList.append(currentXY)
rakeList = np.array(rakeList)
rakeProfileXYList = np.array(rakeProfileXYList)

XMesh = rakeProfileXYList[0][0][::-1]
ZMesh = rakeList

xPoints, zPoints = np.meshgrid(XMesh, ZMesh)

YMesh = []
for i in range(len(rakeProfileXYList)):
    YMesh.append(rakeProfileXYList[i][1])
yPoints = np.array(YMesh)


fig = plt.figure()
spec = mpl.gridspec.GridSpec(nrows=1, ncols=1)


ax0 = fig.add_subplot(spec[0], projection='3d')
# ax1 = fig.add_subplot(spec[1], projection='3d')
# ax11 = fig.add_subplot(spec[2])


ax0.plot_surface(xPoints, YMesh, zPoints, rstride=1, cstride=1,
                 cmap="viridis", edgecolor="none")
ax0.xaxis.set_ticks(np.linspace(-KJ.discSpacing/2, KJ.discSpacing/2, 5))
ax0.set_xlim(np.amin(xPoints), np.amax(xPoints))
ax0.set_ylim(np.amax(yPoints),0)
ax0.grid(False)

# ax.patch.set_facecolor('grey')
plt.xticks(np.arange(np.amin(xPoints), np.amax(xPoints), 0.0005))
fig.suptitle("Velocity Magnitude Profile", fontsize = 18)
```

Out[6]: Text(0.5, 0.98, 'Velocity Magnitude Profile')

In [7]:

```python
nProfileList = [2, 8, 12, 16]
rpmRange = np.arange(100, 6100, 100)

storeDict = {}
for i in range(len(nProfileList)):
    storeRPMPower = np.zeros((len(rpmRange), 2))
    for j in range(len(rpmRange)):
        KJ, solKJ, rsKJ = solutionGenerator(rotorInner, rotorOuter, discSpacing, discThickness, 
nDisc-1, 
                  voluteThickness, wallSpace, 0, wallDisplacement, 
                  TotalMassFlowRate, density, rpmRange[j], profileN = nProfileList[i])
        powerOutput = power(solKJ[:,0], rsKJ, KJ)
        storeRPMPower[j, :] = rpmRange[j], powerOutput
    storeDict[nProfileList[i]] = storeRPMPower

'''RPM Power validation plot'''
fig, ax = plt.subplots()
for i in storeDict:
    ax.plot(storeDict[i][:,0], storeDict[i][:,1], label=f"n_pr = {i}")
ax.set_title("Power vs RPM", fontsize=20)

ax.set_xlabel("RPM", labelpad=10, fontsize=15)
ax.set_ylabel("Power (W)", labelpad=10, fontsize=15)
ax.tick_params(axis='x', labelsize=15)
ax.tick_params(axis='y', labelsize=15)
ax.set_ylim(bottom=0)

ax.legend(fontsize=15, loc="upper left")

columns = {"RPM": storeDict[2][:,0]}
for i in storeDict:
    columns[f"n = {i}"] = storeDict[i][:,1]

df = pd.DataFrame(data=columns)
df.to_excel("profileN_powerRPM.xlsx")
```

In [43]:

```python
'''CFD RPM Power Plots'''

'''Base Case Simulation'''
KJ, solKJ, rsKJ = solutionGenerator(rotorInner, rotorOuter, discSpacing, discThickness, nDisc-1,
                  voluteThickness, wallSpace, 0, wallDisplacement,
                  TotalMassFlowRate, density, revPerMinute[0], profileN = 2)
df = pd.read_excel("excelFiles\\shearStressesCFD.xlsx")
columnList = list(df.columns)

rowToAdd = [None] * len(df.columns)
indexToAdd = []
storeDict = {}
for i in range(len(columnList)):
    columnName = columnList[i]
    try:
        justInCase = columnName.split()
    except:
        justInCase = [columnName]
    if 'Unnamed:' not in justInCase:
        x_df =
np.array(list(df[columnName].loc[df[columnName].map(safeFloatConvert)]))/KJ.outerRadius
        y_df =
np.array(list(df[columnList[i+1]].loc[df[columnList[i+1]].map(safeFloatConvert)]))
        if x_df[0] > x_df[-1]:
            x_df = x_df[::-1]
            y_df = y_df[::-1]
        if y_df[0] <= 0:
            y_df = -y_df
        storeDict[columnName] = [x_df, y_df]
        indexToAdd.append(i)

constantTerm = (KJ.outerRadius**3)*2*pi
lastRowIndex = len(df)
omegaRange = (2*pi/60)*np.array([500,1500,2000,2500,3000,3500,4000])
counter = 0
for i in storeDict:
    rProfile, shearValue = storeDict[i][0], storeDict[i][1]
    integrateTerm = shearValue*rProfile**2
    totalPower = 2*omegaRange[counter]*KJ.numberSpacing*\
        (constantTerm*scipy.integrate.simps(integrateTerm, x = rProfile))
    rowToAdd[indexToAdd[0]] = f"power: {totalPower}"
    indexToAdd.pop(0)
    if "power:" in df[i].iloc[-1].split():
        lastRowIndex = len(df) - 1
    counter += 1
df.loc[lastRowIndex] = rowToAdd
df.to_excel("excelFiles\\shearStressesCFD.xlsx", index=False)
```

In [14]:

```python
'''Base Case Simulation'''
KJ, solKJ, rsKJ = solutionGenerator(rotorInner, rotorOuter, discSpacing, discThickness, nDisc-1,
                  voluteThickness, wallSpace, 0, wallDisplacement,
                  TotalMassFlowRate, density, revPerMinute[0], profileN = 2)

refinedZS = np.linspace(-discSpacing/2, discSpacing/2, 100)
normalisedRefinedZS = 2*refinedZS/discSpacing

nProfileList={
    2:[(1-np.power((2*refinedZS/KJ.discSpacing),2)), "-"],
    4:[(1-np.power((2*refinedZS/KJ.discSpacing),4)), "--"],
    6:[(1-np.power((2*refinedZS/KJ.discSpacing),6)), "-."],
    8:[(1-np.power((2*refinedZS/KJ.discSpacing),8)), ":"]
}

fig, ax = plt.subplots()
for i in nProfileList:
    ax.plot(normalisedRefinedZS, nProfileList[i][0], nProfileList[i][1],\
            linewidth=1.5, color="black", label=f"n={i}")
ax.set_xlabel("Normalised Disc Spacing", fontsize=25, labelpad=10)
ax.set_ylabel("Velocity", fontsize=25, labelpad=10)
ax.legend(fontsize=25)
```

Out[14]:

```
<matplotlib.legend.Legend at 0x2d9001da488>
```

In [ ]: