Airfoil from Joukowski-airfoil-rotation-online.ipynb

In [1]:
```
#!pip install plotly
import plotly.graph_objects as go
import numpy as np
import numpy.ma as ma
import matplotlib.pyplot as plt
```

In [2]:
```
def J(z, lam, alpha):
    return z+(np.exp(-1j*2*alpha)*lam**2)/z
```

In [3]:
```
def circle(C, R):
    t = np.linspace(0,2*np.pi, 200)
    return C + R*np.exp(1j*t)
```

In [4]:
```
def deg2radians(deg):
    return deg*np.pi/180
```

In [5]:

```
def flowlines1(alpha=11, beta=5, V_inf=1, R=5, ratio=1.2):
    #alpha, beta are given in degrees
    #ratio =R/lam
    alpha = deg2radians(alpha)# angle of attack
    beta = deg2radians(beta)# -beta is the argument of the complex no (Joukows
    if ratio <= 1: #R/lam must be >1
        raise ValueError('R/lambda must be > 1')
    lam = R/ratio#lam is the parameter of the Joukowski transformation

    center_c = np.exp(-1j*alpha)*(lam-R*np.exp(-1j*beta))

    Circle = circle(center_c,R)
    Airfoil = J(Circle, lam, alpha)
    X = np.arange(-3, 3, 0.1)
    Y = np.arange(-3, 3, 0.1)

    x,y = np.meshgrid(X, Y)
    z = x+1j*y
    z = ma.masked_where(np.absolute(z-center_c)<=R, z)
    w = J(z, lam, alpha)
    beta = beta+alpha
    Z = z-center_c #Also known as in paper, z'=z-zo
    #print(Z)
    Gamma = -4*np.pi*V_inf*R*np.sin(beta)#circulation
    U = np.zeros(Z.shape, dtype=np.complex)
    with np.errstate(divide='ignore'):#
        for m in range(Z.shape[0]):
            for n in range(Z.shape[1]):# due to this numpy bug https://github.
                                       #we evaluate  this term of the flow ele
                U[m,n] = Gamma*np.log((Z[m,n])/R)/(2*np.pi)
    c_flow = V_inf*Z + (V_inf*R**2)/Z - 1j*U #the complex flow

    return w, c_flow.imag, Airfoil
```

In [6]:

```
#PRACTICE WITH NEW FLOWLINES
def flowlines(alpha=25, beta=5, V_inf=1, R=1, ratio=1.2):
    #alpha, beta are given in degrees
    #ratio =R/lam
    alpha = deg2radians(alpha)# angle of attack
    beta = deg2radians(beta)# -beta is the argument of the complex no (Joukows
    if ratio <= 1: #R/lam must be >1
        raise ValueError('R/lambda must be > 1')
    lam = R/ratio#lam is the parameter of the Joukowski transformation

    center_c = np.exp(-1j*alpha)*(lam-R*np.exp(-1j*beta))

    Circle = circle(center_c,R)
    Airfoil = J(Circle, lam, alpha)
    X = np.arange(-3, 3, 0.1)
    Y = np.arange(-3, 3, 0.1)

    x,y = np.meshgrid(X, Y)
    z = x+1j*y
    z = ma.masked_where(np.absolute(z-center_c)<=R, z)
    w = J(z, lam, alpha)
    beta = beta+alpha
    t = -1.757745+0.03484753j
    T = t -center_c
    Z = z-center_c
    #ZV = z-center_c + .01j
    #print(Z)
    #print("This is ZV",ZV)
    Gamma = -4*np.pi*V_inf*R*np.sin(beta)#circulation
    U = np.zeros(Z.shape, dtype= complex)
    #UK = Gamma*np.log((T)/R)/(2*np.pi)
    #ck_flow = V_inf*T + (V_inf*R**2)/T - 1j*UK #the complex flow
    #print("This is ck_flow:",ck_flow)
    #print("This is velocity:",UK)
    with np.errstate(divide='ignore'):#
        for m in range(Z.shape[0]):
            for n in range(Z.shape[1]):# due to this numpy bug https://github.
                                       #we evaluate  this term of the flow ele
                U[m,n] = Gamma*np.log((Z[m,n])/R)/(2*np.pi)
    c_flow = V_inf*Z + (V_inf*R**2)/Z - 1j*U #the complex flow

    v_components = np.zeros(Z.shape, dtype= complex)

    v_components = (V_inf*(1-((R**2)/(((R*np.exp(-1j*beta)**2)**2))))-(1j*Gamm
    # Change above equation to vary the h-value or height and choose a few of
    #print("Velocity components:",v_components)
    return w, c_flow.imag,  Airfoil
```

In [7]:
```
def get_contours(mplcont):
    conts = mplcont.allsegs  #  get the segments of line computed via plt.cont
    xline = []
    yline = []

    for  cont in conts:
        if len(cont) != 0:
            for arr in cont:

                xline += arr[:,0].tolist()
                yline += arr[:,1].tolist()
                xline.append(None)
                yline.append(None)

    return xline, yline
```

In [8]:
```
levels = np.arange(-3, 3.7, 0.25).tolist()
plt.figure(figsize=(0.05,0.05))
plt.axis('off')
Alpha=   list(range(0, 19)) + list(range(17, -19, -1)) + list(range(-17, 1))

frames = []

for k, alpha in enumerate(Alpha):
    Jz, stream_func, Airfoil = flowlines(alpha=alpha)
    #Define an instance of the mpl class contour
    cp = plt.contour(Jz.real, Jz.imag, stream_func, levels=levels, colors='blu

    xline, yline = get_contours(cp)

    frames.append( go.Frame(data=[go.Scatter(x=xline,
                                             y=yline),
                                  go.Scatter(x=Airfoil.real,
                                             y=Airfoil.imag),
                                 ],
                            traces=[0, 1],
                            name=f'frame{k}'
                           ) )
```

```
<ipython-input-6-4936c1319db5>:39: RuntimeWarning: invalid value encountered
in multiply
  U[m,n] = Gamma*np.log((Z[m,n])/R)/(2*np.pi)
```

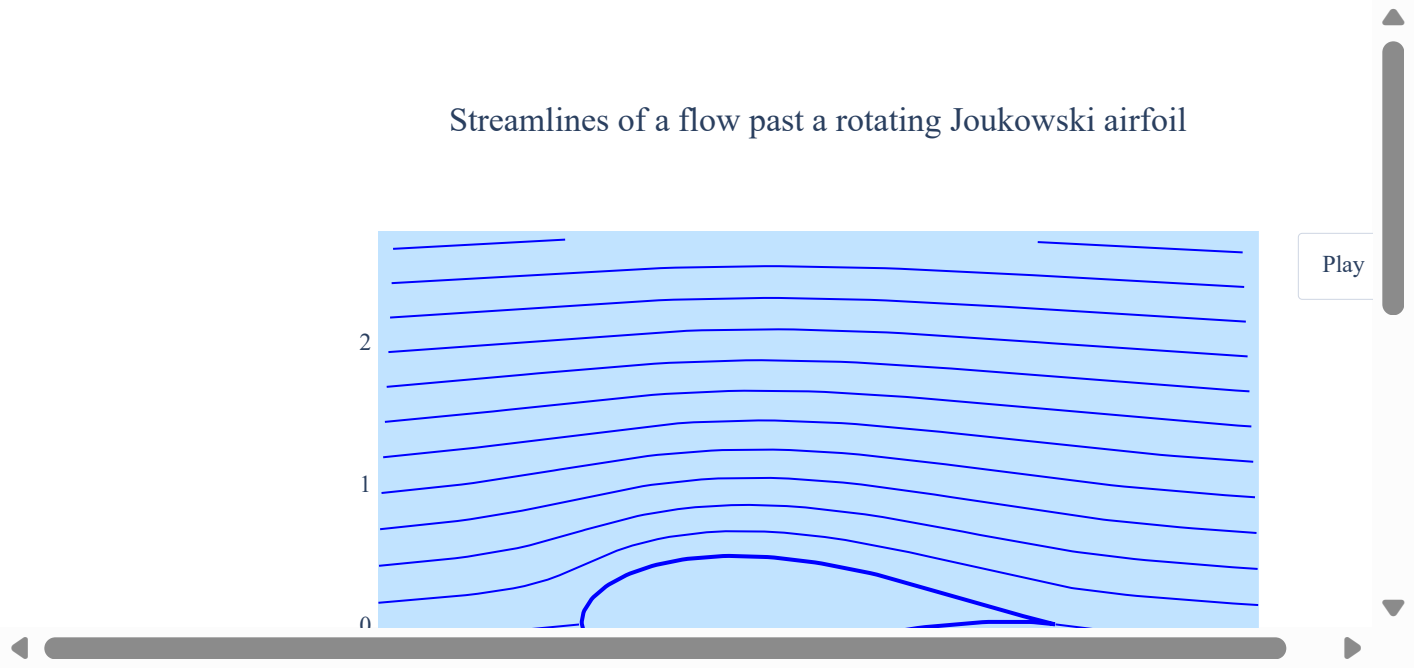
In [9]:
```
data = [go.Scatter(
           x=frames[0].data[0].x,
           y=frames[0].data[0].y,
           mode='lines',
           line=dict(color='blue', width=1),
           name=''
          ),
        go.Scatter(
           x=frames[0].data[1].x,
           y=frames[0].data[1].y,
           mode='lines',
           line=dict(color='blue', width=2),
           name=''
          )
      ]
```

In [10]:
```
def get_sliders(Alpha, n_frames, fr_duration=100, x_pos=0.0, y_pos=0, slider_l
    # n_frames= number of frames
    #fr_duration=duration in milliseconds of each frame
    #x_pos x-coordinate where the slider starts
    #slider_len is a number in (0,1] giving the slider length as a fraction of
    return [dict(steps= [dict(method= 'animate',#Sets the Plotly method to be
                                            #slider value is changed.
                              args= [ [ f'frame{k}'],#Sets the arguments value
                                                     #the Plotly meth
                                      dict(mode= 'immediate',
                                           frame= dict( duration=fr_duration,
                                           transition=dict( duration= 0)
                                          )
                                      ],
                              label=str(alpha)
                             ) for k, alpha in enumerate(Alpha)],
                transition= dict(duration= 0 ),
                x=x_pos,
                y=y_pos,
                currentvalue=dict(font=dict(size=12),
                                  prefix="Angle of attack: ",
                                  visible=True,
                                  xanchor= "center"
                                 ),
                len=slider_len)
           ]
```

In [11]:

```
axis = dict(showline=True, zeroline=False, ticklen=4, mirror=True, showgrid=Fa


layout = go.Layout(title_text="Streamlines of a flow past a rotating Joukowski
                   title_x=0.5,
             font=dict(family='Balto'),
             showlegend=False,
             autosize=False,
             width=600,
             height=600,
             xaxis=dict(axis, **{'range': [ma.min(Jz.real), ma.max(Jz.real)]}),
             yaxis=dict(axis, **{'range':[ma.min(Jz.imag), ma.max(Jz.imag)]}),

             plot_bgcolor='#c1e3ff',
             hovermode='closest',

             updatemenus=[dict(type='buttons', showactive=False,
                                 y=1,
                                 x=1.15,
                                 xanchor='right',
                                 yanchor='top',
                                 pad=dict(t=0, l=10),
                                 buttons=[dict(
                                             label='Play',
                                             method='animate',
                                             args=[None, dict(frame=dict(durati
                                                              transition=dict(d
                                                              fromcurrent=True,
                                                              mode='immediate')
                                 )
                             ])

layout.update(sliders=get_sliders(Alpha, len(frames), fr_duration=50))
fig = go.Figure(data=data,layout=layout, frames=frames)
```

In [12]:
```
#!pip install chart_studio
import chart_studio.plotly as py
import pandas as pd
#py.iplot(fig, filename='streamlJouk1485353936.44' )
#fig, filename='streamlJouk1485353936.44'.show()
fig.show(filename='streamlJouk1485353936.44')
#Aggressive goals
#Add h-distance points
#difference mesh densities, radius, camber, and angle of attack, etc
```



Repeat Flowlines analysis and add velocity components around the airfoil

In [29]:

```
# using flowlines architechture for setup
import pandas as pd
def flowlines2(alpha1=15, beta1=5, V_inf1=1, R1=1, ratio1=1.2):

    #alpha, beta are given in degrees
    #ratio =R/lam
    alpha1 = deg2radians(alpha1)# angle of attack
    beta1 = deg2radians(beta1)# -beta is the argument of the complex no (Jouko
    if ratio1 <= 1: #R/lam must be >1
        raise ValueError('R/lambda must be > 1')
    lam1 = R1/ratio1#lam is the parameter of the Joukowski transformation

    center_c1 = np.exp(-1j*alpha1)*(lam1-R1*np.exp(-1j*beta1))

    Circle1 = circle(center_c1,R1)
    Airfoil1 = J(Circle1, lam1, alpha1)
    X = np.arange(-3, 3, 0.1)
    Y = np.arange(-3, 3, 0.1)

    x,y = np.meshgrid(X, Y)
    z1 = x+1j*y
    z1 = ma.masked_where(np.absolute(z1-center_c1)<=R1, z1)
    w1 = J(z1, lam1, alpha1)
    beta1 = beta1+alpha1
    t1 = -1.757745+0.03484753j
    T1 = t1  -center_c1
    Z1 = z1-center_c1
    Z_air1 = Airfoil1.real+Airfoil1.imag+.3j#,.2j,.3j
    #print(Z)
    Gamma1 = -4*np.pi*V_inf1*R1*np.sin(beta1)#circulation
    U1 = np.zeros(Z1.shape, dtype= complex)
    with np.errstate(divide='ignore'):#
        for m in range(Z1.shape[0]):
            for n in range(Z1.shape[1]):# due to this numpy bug https://github
                                        #we evaluate  this term of the flow ele
                U1[m,n] = Gamma1*np.log((Z1[m,n])/R1)/(2*np.pi)
    c_flow1 = V_inf1*Z1 + (V_inf1*R1**2)/Z1 - 1j*U1 #the complex flow

    v_components1 = np.zeros(Z_air1.shape, dtype= complex)

    v_components1 = (V_inf1*(1-((R1**2)/(((R1*np.exp(-1j*beta1)**2)**2))))-(1j
    #changed alpha1 to 15
    # Change above equation to vary the h-value or height and choose a few of
    norms=((v_components1[1]-v_components1[0])/(np.linalg.norm(v_components1.r
    #other component in this case being imaginary
    img2=v_components1
    norms_denom = (np.linalg.norm(v_components1.real+img2))
    #print(img2)
    norms_better = ((v_components1.real)/(np.linalg.norm(v_components1.real)))
    #print("j_normal comp",(-v_components1[0])/(np.linalg.norm(v_components1))
    #print(v_components1.real[3])
    #print(norms_better.shape)
    #SF = pd.DataFrame(norms_denom)
    #SF.to_csv("norms_denom__2D_R1.5_h_0.csv")
    PDF = pd.DataFrame(v_components1.real)
    PDF.to_csv("2D__Comps_r1_00h.3.real")
    PDFz = pd.DataFrame(v_components1.imag)
```

```
    PDFz.to_csv("2D__Comps_r1_00h.3.imag")
    #norms.tofile('data2.csv')
    #DF1 = pd.DataFrame(norms)
    #DF1.to_csv("Norms1")
    #print(norms.size)
    #print(type(v_components1.real))
    return w1, c_flow1.imag,  Airfoil1
```

In [30]:
```
import pandas as pd
levels = np.arange(-3, 3.7, 0.25).tolist()
plt.figure(figsize=(0.05,0.05))
plt.axis('off')
Alpha1=   list(range(0, 19)) + list(range(17, -19, -1)) + list(range(-17, 1))

frames = []

for k, alpha1 in enumerate(Alpha1):
    Jz, stream_func, Airfoil1 = flowlines2(alpha1=alpha1)
    #Define an instance of the mpl class contour
    cp = plt.contour(Jz.real, Jz.imag, stream_func, levels=levels, colors='blu

    xline, yline = get_contours(cp)

    frames.append( go.Frame(data=[go.Scatter(x=xline,
                                             y=yline),
                                  go.Scatter(x=Airfoil1.real,
                                             y=Airfoil1.imag),
                                 ],
                            traces=[0, 1],
                            name=f'frame{k}'
                           ) )
```

```
<ipython-input-29-90d5e5caa8b6>:36: RuntimeWarning:

invalid value encountered in multiply

<ipython-input-29-90d5e5caa8b6>:44: RuntimeWarning:

invalid value encountered in cdouble_scalars

<ipython-input-29-90d5e5caa8b6>:49: RuntimeWarning:

invalid value encountered in true_divide
```

In [16]:
```
# Pick up pace!
# Include both components to normal and finite h.
```

In [ ]:

In [ ]:

In [ ]: