Dependencies

In [ ]:
```
#!/usr/bin/env python3

import math
import numpy as np
import random
from scipy import linalg
import statistics

import plotly.graph_objects as go
import plotly.express as px
# to make fig.show() work do this: pip install --upgrade nbformat, then rest
```

Sampling Functions

In [ ]:
```
# return Gumbel's function value
def gumbel(x):
    return math.pow(math.e, x) * math.pow(math.e, -math.pow(math.e, x))

# return an array tuple of randomly sampled datapoints
def random_points(start, end, n_points):
    x_datapoints = sorted([start+(end-start)*random.random() for _ in range(
    y_datapoints = list(map(gumbel, x_datapoints))
    return np.array(x_datapoints), np.array(y_datapoints)

# return an array tuple of equally distanced datapoints
def equal_points(start, end, n_points):
    x_datapoints = [float(_) for _ in range(start, end+1, 1)]
    y_datapoints = list(map(gumbel, x_datapoints))
    return np.array(x_datapoints), np.array(y_datapoints)

# return an array tuple of datapoints given by the Chebyshev points
def cheby_points(start, end, n_points):
    x_datapoints = sorted([
        start+(end-start)*((-math.cos((x-1)*math.pi/(n_points-1))+1)/2) for
    ])
    y_datapoints = list(map(gumbel, x_datapoints))
    return np.array(x_datapoints), np.array(y_datapoints)
```

Interpolation

Polynomial

In [ ]:
```
# @title Newton-Rhapson Interpolation

# return an array of coefficients using Newton-Rhapson's divided differeznce
def nr_coeffts(x_datapoints, y_datapoints, n_points):
```

```
    a = y_datapoints.copy()
    for k in range(1, n_points):
        a[k:n_points] = (a[k:n_points] - a[k-1])/(x_datapoints[k:n_points] -
    return a

# return the function value given by the interpolated function via the Newto
def nr_evalfunct(coefficients, x_datapoints, x, degree):
    value = coefficients[degree]
    for k in range(1, degree+1):
        value = coefficients[degree-k] + (x - x_datapoints[degree-k])*value
    return value
```

RBF

In [ ]:
```
# return multiquadratic rbf basis
def rbf_basis_multiq(x, x_i, sigma):
    return math.sqrt(math.pow((x-x_i), 2)+math.pow(sigma, 2))

# return an array of coefficients using multiquadratic rbf
def rbf_coeffts(x_datapoints, y_datapoints, n_points, sigma):
    rbfmatrx = np.zeros((9,9))
    for j in range(0, n_points):
        rbfmatrx[j][j] = rbf_basis_multiq(x_datapoints[j], x_datapoints[j],
        for i in range(j+1, n_points):
            rbfmatrx[i][j] = rbf_basis_multiq(x_datapoints[i], x_datapoints[
            rbfmatrx[j][i] = rbfmatrx[i][j]
    return linalg.solve(rbfmatrx, y_datapoints)

# return the function value given by the interpolated function via the multi
def rbf_evalfunct(x, x_datapoints, y_datapoints, n_points):
    sigma = statistics.stdev(x_datapoints)
    coeffts = rbf_coeffts(x_datapoints, y_datapoints, n_points, sigma)
    value = 0
    for k in range(n_points):
        value += coeffts[k]*rbf_basis_multiq(x, x_datapoints[k], sigma)
    return value
```

Plotting

In [ ]:
```
# plot the gumbel function
def plot_gumbel(step_size, start, end):
   xdata = []
   ydata = []
   for x in range(start*10, (end*10)+1, step_size):
      xdata.append(x/10)
      ydata.append(gumbel(x/10))

   fig = px.line(x=xdata, y=ydata)
   fig.update_layout(
      height=1080*0.5,
      width=1920*0.6,
      title_text = "Gumbel function",
```

```
        font_family="CMU Serif",
        font_size=15,
        title_font_size=25,
        font_color="#0e0f11",
        margin=dict(t=120, b=80)
    )
    fig.update_yaxes(range=[-0.05, 0.5], title_text='y')
    fig.update_xaxes(range=[-5.5, 3.5], title_text='x')
    fig.show()


# plot the newton-rhapsn polynomial interpolation
def plot_poly_interp(_title, x_datapoints, y_datapoints, step_size, start, e
    coeffts = nr_coeffts(x_datapoints, y_datapoints, n_points)
    xdata = []
    ydata = []
    ydata2= []
    for x in range(start*10, (end*10)+1, step_size):
        xdata.append(x/10)
        ydata.append(nr_evalfunct(coeffts, x_datapoints, x/10, n_points-1))
        ydata2.append(gumbel(x/10))

    fig = go.Figure()
    fig.add_traces([
        go.Scatter(x=xdata, y=ydata2, mode='lines', marker = {'color' : 'greer
        go.Scatter(x=xdata, y=ydata, mode='lines', line_dash='dash',line_width
        go.Scatter(x=x_datapoints, y=y_datapoints, mode='markers', name="Sampl
    ])

    fig.update_traces(marker=dict(size=8.5,
                                  color='red',
                                  line=dict(width=1,
                                            color='DarkSlateGrey')),
                      selector=dict(mode='markers'))
    fig.update_layout(
        height=1080*0.5,
        width=1920*0.6,
        title_text=_title,
        font_family="CMU Serif",
        font_size=15,
        title_font_size=25,
        font_color="#0e0f11",
        margin=dict(t=120, b=80)
    )
    fig.update_yaxes(range=[-0.05, 0.5], title_text='y')
    fig.update_xaxes(range=[-5.5, 3.5], title_text='x')
    fig.show()

# plot the multiquadratic rbf interpolation
def plot_rbf_interp(_title, x_datapoints, y_datapoints, step_size, start, er
    xdata = []
    ydata = []
    ydata2= []
```

```
    for x in range(start*10, (end*10)+1, step_size):
        xdata.append(x/10)
        ydata.append(rbf_evalfunct(x/10, x_datapoints, y_datapoints, n_points)
        ydata2.append(gumbel(x/10))

    fig = go.Figure()
    fig.add_traces([
        go.Scatter(x=xdata, y=ydata2, mode='lines', marker = {'color' : 'greer
        go.Scatter(x=xdata, y=ydata, mode='lines',line_dash='dash', line_width
        go.Scatter(x=x_datapoints, y=y_datapoints, mode='markers', name="Sampl
    ])

    fig.update_traces(marker=dict(size=8.5,
                                  color='red',
                                  line=dict(width=1,
                                            color='DarkSlateGrey')),
                      selector=dict(mode='markers'))
    fig.update_layout(
        height=1080*0.5,
        width=1920*0.6,
        title_text=_title,
        font_family="CMU Serif",
        font_size=15,
        title_font_size=25,
        font_color="#0e0f11",
        margin=dict(t=120, b=80)
    )
    fig.update_yaxes(range=[-0.05, 0.5], title_text='y')
    fig.update_xaxes(range=[-5.5, 3.5], title_text='x')
    fig.show()

# plot the multiquadratic rbf interpolation with gumbel superimposed
def superplot_rbf_interp(_title, x_datapoints, y_datapoints, step_size, star
    xdata = []
    ydata = []
    ydata2 = []
    for x in range(start*10, (end)*10+1, step_size):
        xdata.append(x/10)
        ydata.append(rbf_evalfunct(x/10, x_datapoints, y_datapoints, n_points)
        ydata2.append(gumbel(x/10))

    fig = go.Figure()
    fig.add_traces([
        go.Scatter(x=xdata, y=ydata2, mode='lines', marker = {'color' : 'greer
        go.Scatter(x=xdata, y=ydata, mode='lines', line_dash='dash', line_widt
        go.Scatter(x=x_datapoints, y=y_datapoints, mode='markers', marker = {'
    ])
    fig.update_layout(
        height=1080*0.5,
        width=1920*0.6,
        title_text=_title,
        font_family="CMU Serif",
        font_size=15,
```

```
        title_font_size=25,
        font_color="#0e0f11",
        margin=dict(t=120, b=80)
    )
    fig.update_traces(marker=dict(size=8.5,
                                  color='red',
                                  line=dict(width=1,
                                            color='DarkSlateGrey')),
                      selector=dict(mode='markers'))
    fig.update_yaxes(range=[-0.05, 0.5], title_text='y')
    fig.update_xaxes(range=[-5.5, 3.5], title_text='x')
    fig.show()
```

Reconstruction Error

In [ ]:

```
# return the reconstruction error of the newton interpolation
def poly_recon_error_nr(x_datapoints, y_datapoints, step_size, start, end, r
    coeffts = nr_coeffts(x_datapoints, y_datapoints, n_points)
    error = 0
    for x in range(start*10, (end*10)+1, step_size):
        error += abs(gumbel(x/10) - nr_evalfunct(coeffts, x_datapoints, x/10,
    return error

# return the reconstruction error of the rbf interpolation
def rbf_recon_error_mq(x_datapoints, y_datapoints, step_size, start, end, n_
    error = 0
    for x in range(start*10, (end*10)+1, step_size):
        error += abs(gumbel(x/10) - rbf_evalfunct(x/10, x_datapoints, y_datap
    return error
```

Getting the Points

In [ ]:

```
start = -5
end = 3
n_points = 9
step_size = 1

random_x_datapoints, random_y_datapoints = random_points(start, end, n_point
fig = px.scatter(x=random_x_datapoints, y=random_y_datapoints)
fig.update_layout(
    title_text="9 Randomly-sampled Points",
    height=1080*0.5,
    width=1920*0.6,
    font_family="CMU Serif",
    font_size=15,
    title_font_size=25,
    font_color="#0e0f11",
    margin=dict(t=120, b=80)
)
fig.update_traces(marker=dict(size=8.5,
                              color='red',
```

```
                              line=dict(width=1,
                                        color='DarkSlateGrey')),
                  selector=dict(mode='markers'))
fig.update_yaxes(range=[-0.05, 0.5], title_text='y')
fig.update_xaxes(range=[-5.5, 3.5], title_text='x')
fig.show()

equal_x_datapoints, equal_y_datapoints = equal_points(start, end, 20)
fig = px.scatter(x=equal_x_datapoints, y=equal_y_datapoints)
fig.update_layout(
    title_text="9 Equidistant Points",
    height=1080*0.5,
    width=1920*0.6,
    font_family="CMU Serif",
    font_size=15,
    title_font_size=25,
    font_color="#0e0f11",
    margin=dict(t=120, b=80)
)
fig.update_traces(marker=dict(size=8.5,
                              color='red',
                              line=dict(width=1,
                                        color='DarkSlateGrey')),
                  selector=dict(mode='markers'))
fig.update_yaxes(range=[-0.05, 0.5], title_text='y')
fig.update_xaxes(range=[-5.5, 3.5], title_text='x')
fig.show()

cheby_x_datapoints, cheby_y_datapoints = cheby_points(start, end, n_points)
fig = px.scatter(x=cheby_x_datapoints, y=cheby_y_datapoints)
fig.update_layout(
    title_text="9 Chebyshev Points",
    height=1080*0.5,
    width=1920*0.6,
    font_family="CMU Serif",
    font_size=15,
    title_font_size=25,
    font_color="#0e0f11",
    margin=dict(t=120, b=80)
)
fig.update_traces(marker=dict(size=8.5,
                              color='red',
                              line=dict(width=1,
                                        color='DarkSlateGrey')),
                  selector=dict(mode='markers'))
fig.update_yaxes(range=[-0.05, 0.5], title_text='y')
fig.update_xaxes(range=[-5.5, 3.5], title_text='x')
fig.show()
```

Polynomial Basis

Computing the Reconstruction Error

```
print(poly_recon_error_nr(random_x_datapoints, random_y_datapoints, step_siz
print(poly_recon_error_nr(equal_x_datapoints, equal_y_datapoints, step_size,
print(poly_recon_error_nr(cheby_x_datapoints, cheby_y_datapoints, step_size,
```

```
20.11791675601053
1.4152350082555476
0.6823219868208231
```

Plotting the Interpolation

```
plot_poly_interp("Randomly-sampled Points - Newton Interpolation", random_x_
plot_poly_interp("Equidistant Points - Newton Interpolation", equal_x_datapo
plot_poly_interp("Chebyshev Points - Newton Interpolation", cheby_x_datapoir
```

RBF Basis

Computing the Reconstruction Error

```
print(rbf_recon_error_mq(equal_x_datapoints, equal_y_datapoints, step_size,
print(rbf_recon_error_mq(cheby_x_datapoints, cheby_y_datapoints, step_size,
```

```
0.3871421172979905
0.39836607540270524
```

Plotting the Interpolation

```
plot_rbf_interp("Equidistant Points - Multiquadratic RBF Interpolation", equ
plot_rbf_interp("Chebyshev Points - Multiquadratic RBF Interpolation", cheby
```

Bonus

```
random_recon_error_poly = poly_recon_error_nr(random_x_datapoints, random_y_
equal_recon_error_poly = poly_recon_error_nr(equal_x_datapoints, equal_y_dat
cheby_recon_error_poly = poly_recon_error_nr(cheby_x_datapoints, cheby_y_dat
equal_recon_error_rbf = rbf_recon_error_mq(equal_x_datapoints, equal_y_datap
cheby_recon_error_rbf = rbf_recon_error_mq(cheby_x_datapoints, cheby_y_datap


minimum_error = min(
    random_recon_error_poly,
    equal_recon_error_poly,
    cheby_recon_error_poly,
    equal_recon_error_rbf,
    cheby_recon_error_rbf
)
```

```
while True:
    bonux_x_datapoints, bonux_y_datapoints = random_points(start, end, n_poi
    current_error = rbf_recon_error_mq(bonux_x_datapoints, bonux_y_datapoint
```

```
    if current_error < minimum_error:
        print(current_error)
        break
```

```
0.19122724289823217
```

In [ ]:

```
# Testing interpolation with 1000 random 9 points

try_x = []

for i in range(1000):
    try_x_datapoints, try_y_datapoints = random_points(start, end, n_points)
    current_error = poly_recon_error_nr(try_x_datapoints, try_y_datapoints,
    try_x.append(current_error)

print(statistics.median(try_x))
```

```
13.117678850288566
```

In [ ]:

```
superplot_rbf_interp("Randomly-sampled Points - Multiquadratic RBF Interpola
 1.17039973e-01, 1.34893811e-02, 3.48940704e-05, 3.21715128e-05,
 9.18502222e-06], step_size, start, end, n_points)
```