In [ ]:
```
import numpy as np
from scipy.optimize import minimize
from scipy.stats import ortho_group
import scipy as sp
import sympy as sm
import scipy.integrate as integrate
import matplotlib.pyplot as plt
import types
import math
import random
```

In [ ]:
```
class Remez:

    def __init__(self, f_: types.FunctionType, lb_: float, ub_: float, n_: int , MAX_ITER_: int):
        self.f = f_
        self.lb = lb_
        self.ub = ub_
        self.n = n_
        self.MAX_ITER = MAX_ITER_
        self.ys = np.zeros(n_ + 2)
        self.iterations = 0

    def Error(self, x: float, alphas: np.ndarray):
        # x is the evaluation point
        # alphas are the coefficients for the approximating polynomial
        # n is the degree of approximating polynomial

        poly = 0
        for j in range(self.n+1):
            poly += alphas[j]*(x**(float(j)))

        return self.f(x) - poly

    def find_maxs(self, alphas: np.ndarray)-> np.array:
        ep = 1e-6
        xlen = (self.ub - self.lb) * 1000
        xs = np.linspace(self.lb, self.ub, xlen)
        fxs = np.abs(self.Error(xs, alphas))
        #plt.plot(fxs)
        dfxs = np.zeros(xlen-1)

        for i in range(xlen-1):
            dfxs[i] = (fxs[i+1] - fxs[i]) / ep

        maximizers = []

        for i in range(xlen-2):
            if dfxs[i] > 0 and dfxs[i+1] < 0:
                maximizers.append((xs[i] + xs[i+1])/2.)

        #plt.plot(dfxs)

        xstar = np.array(maximizers)

        return xstar

    def form_mat(self, xs: np.ndarray):
        # Forms the matrix for the Remez Algorithm
        # n is the degree of polynomial we are approximating with
        n = xs.size - 2

        #A is size n+2 x n+1
        A = np.zeros((n+2, n+2))

        for i in range(n+2):
            for j in range(n+2):
                if j == n+1:
                    A[i, j] = (-1)**(i+1)
                else:
                    A[i][j] = xs[i]**j
        return A

    def convergence(self, xs: np.ndarray, alphas: np.ndarray):
        eps = 1e-5

        f_maxs = self.Error(xs, alphas)
        for i in range(len(f_maxs)-1):
            if abs(abs(f_maxs[i]) - abs(f_maxs[i+1])) > eps:
                return False
            if f_maxs[i] * f_maxs[i+1] > 0:
                return False

        return True

    def remez(self):
        xs = np.linspace(self.lb, self.ub, self.n+2)

        it = 0
        while it < self.MAX_ITER:
```

```python
            A = self.form_mat(xs)
            bs = self.f(xs)

            # LU decomp on A to improve stability
            lu, piv = sp.linalg.lu_factor(A)
            ys = sp.linalg.lu_solve((lu, piv), bs)

            ang = self.find_maxs(ys[:-1])
            if self.convergence(ang, ys[:-1]):
                self.ys = ys
                self.iterations = it
                return
            else:
                # Replace the closest points in xs with angs
                for i in range(ang.size):
                    distx = np.zeros_like(xs)
                    for j in range(len(xs)):
                        distx[j] = abs(xs[j] - ang[i])
                    replaceIdx = np.argmin(distx)
                    xs[replaceIdx] = ang[i]
            it+=1
        self.ys = ys
        self.iterations = it
        print("Remez needs more iterations")
```

In [ ]:
```python
def f(x: np.ndarray):
    return 1./x
```

In [ ]:
```python
n = 4
ub = 5
lb = 1
MAX_ITER = 100

my_remez = Remez(f, lb, ub, n, MAX_ITER)
my_remez.remez()
ys = my_remez.ys
```

In [ ]:
```python
x_vals = np.linspace(lb, ub, 500)
x_ticks = np.round(np.linspace(lb + (ub - lb)/5, ub - (ub - lb)/5, 5), 2)
f_true = f(x_vals)
fig, axs = plt.subplots(1, 1, figsize=(8, 6))

poly_approx = np.polyval(ys[::-1][1:], x_vals)

axs.plot(x_vals, poly_approx, '--', label=f"n = {n}" , zorder = 1)

axs.plot(x_vals, f_true, label=r"True", color='k', zorder = 2, linewidth = 2)
axs.legend()
axs.grid(True)
axs.set_axisbelow(True)
plt.gca().set_facecolor((0.9, 0.9, 0.9))
axs.set_title(f"Remez Exchange Algorithm for f(x) = 1/x, Error: {ys[-1]:e}")
plt.show()
```