CS 169 - HW 4

Sanath Nair

1)

I decieded to implement an Augmented Lagrange Method optimizer, using ChatGPT to convert the julia code to python from K&W (p. 183). This function utilizes the minimize method from the scipy library

In [ ]:
```python
from time import perf_counter
import numpy as np
from scipy.optimize import minimize


def augmented_lagrangian_method(func, equality_constraint, x, k_max=10, rho=
    start = perf_counter()
    lambda_vec = np.zeros(len(equality_constraint(x)))

    for _ in range(1, k_max + 1):
        def p(x): return func(x) + rho/2 * \
            np.sum(equality_constraint(x)**2) - np.dot(lambda_vec, equality_
        result = minimize(lambda x: func(x) + p(x), x)

        x = result.x
        rho *= gamma
        lambda_vec -= rho * equality_constraint(x)

    end = perf_counter()
    return x, end - start
```

The code below is from HW2. In order to measure the difference I wanted to use the same test functions and starting values.

Below is a generic implementation of the Rosenbrcok function with default values of `a=1` and `b=5`

In [ ]:
```python
def rosenbrock_generator(*, a=1, b=5):
    """
    arguments a, b must be passed as keyword arguments
    """
    def rosenbrock(*args):
        total = 0
        for i in range(len(args) - 1):
            x1 = args[i]
            x2 = args[i+1]
            total += (a - x1)**2 + b*(x2 - (x1**2))**2
        return total

    return rosenbrock


def rosenbrock_gradient_generator(*, a=1, b=5):
    """
    arguments a, b must be passed as keyword arguments
    """
    def rosenbrock_gradient(*args):
        dx1 = -2*(a-args[0]) + 2*b*(args[1]-args[0]**2)*(-2*args[0])
        gradients = [dx1]
        for i in range(1, len(args) - 1):
            x_prev = args[i-1]
            x = args[i]
            x_next = args[i+1]
            dxn = 2*b*(x - x_prev**2) - 2*(a-x) - 4*b*x*(x_next - x**2)*x
            gradients.append(dxn)
        dxn = 2 * b * (args[-1] - args[-2]**2)
        gradients.append(dxn)
        return np.array(gradients)

    return rosenbrock_gradient
```

Below is a simple equality constraint that simply returns an array of 0 based on the input dimensions. This should allow us to see if ALM does actually converge to the true minimum of the 10-dim rosenbrock without any constraints.

In [ ]:
```python
def eq_constraints(x):
    return np.zeros(x.shape, dtype=float)
```

The function below takes a list of different starting points and runs the gradient descent method collecting data and returning it back.

__NOTE (Couple limitation):__

1. We don't know the true minimum for higher dimension functions so we cannot measure absolute error
2. I use the minimize function from scipy's library and therefore don't have access to the true number of function calls
3. Convergence measure is simply number of iterations

In [ ]:
```
import pandas as pd


def run_test(func, starting_points, /, alpha=0.01, dimensions=10):
    data = []

    for starting_point in starting_points:
        _, wall_time = augmented_lagrangian_method(func, eq_constraints, sta
        data.append(wall_time)

    columns = ['wall time']
    df = pd.DataFrame(data, columns=columns)

    mean_std = df.agg(['mean', 'std'])

    # Concatenate the results into a new DataFrame for comparison
    return mean_std.T
```

For the starting points since the Rosenbrock's minimum is at (1,1...,1) I will randomly generate points in the range of (-4, 4).

In [ ]:
```
def generate_starting_points(N, dim, lower, upper):
    random_arrays = []

    for _ in range(N):
        # Generate a random array of size D with values between lower and up
        random_array = np.random.uniform(lower, upper, dim)
        random_arrays.append(random_array)

    return random_arrays


starting_points = generate_starting_points(50, 10, -4, 4)
```

In [ ]:
```
rosenbrock = rosenbrock_generator()

run_test(rosenbrock, starting_points)
```

Out[ ]:

	mean	std
wall time	0.003054	0.000621