Homework 4 writeups

Name: Oorjit Chowdhary

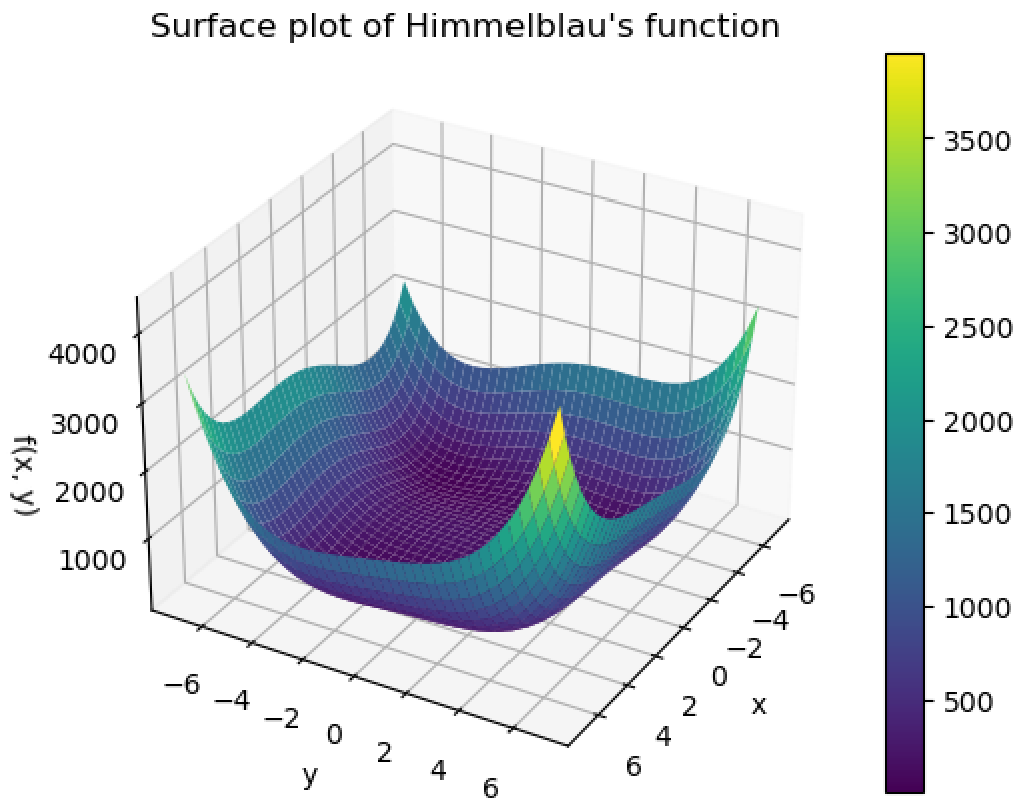
Section: AMATH 301 B

Problem 1

Part (a)

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy

# Himmeblau's function
f = lambda x, y: (x**2 + y - 11) ** 2 + (x + y**2 - 7) ** 2
fp = lambda p: f(p[0], p[1])

x = np.linspace(-7, 7, 40)
y = np.linspace(-7, 7, 40)

X, Y = np.meshgrid(x, y)

fig = plt.figure()
ax = plt.axes(projection='3d')

surface = ax.plot_surface(X, Y, f(X, Y), cmap='viridis')
fig.colorbar(surface)
ax.view_init(30, 30)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('f(x, y)')
ax.set_title('Surface plot of Himmelblau\'s function')
```

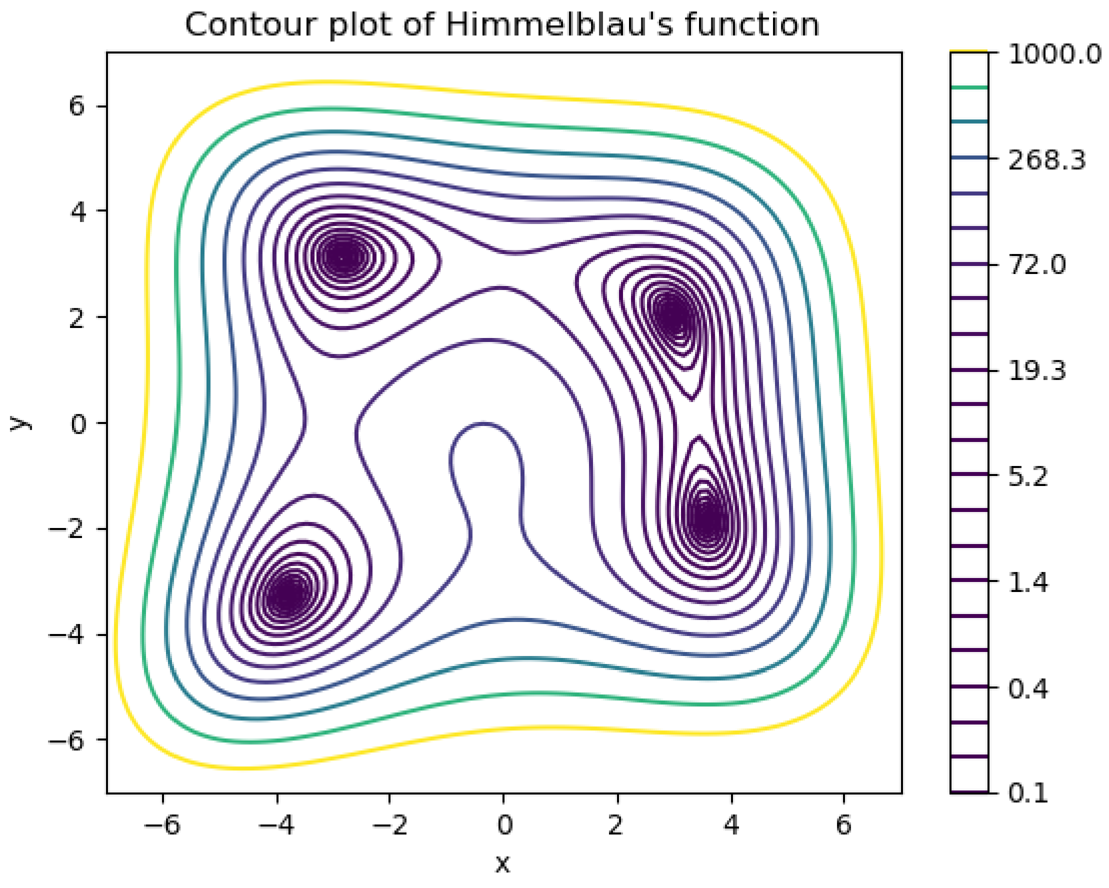
Out[ ]:
```
Text(0.5, 0.92, "Surface plot of Himmelblau's function")
```



Part (b)

```
In [ ]: x = np.linspace(-7, 7, 100)
        y = np.linspace(-7, 7, 100)

        X, Y = np.meshgrid(x, y)

        fig2 = plt.figure()
        ax2 = plt.axes()

        contour = ax2.contour(X, Y, f(X, Y), np.logspace(-1, 3, 22), cmap='viridis')
        fig2.colorbar(contour)
        ax2.set_xlabel('x')
        ax2.set_ylabel('y')
        ax2.set_title('Contour plot of Himmelblau\'s function')
```

```
Out[ ]: Text(0.5, 1.0, "Contour plot of Himmelblau's function")
```



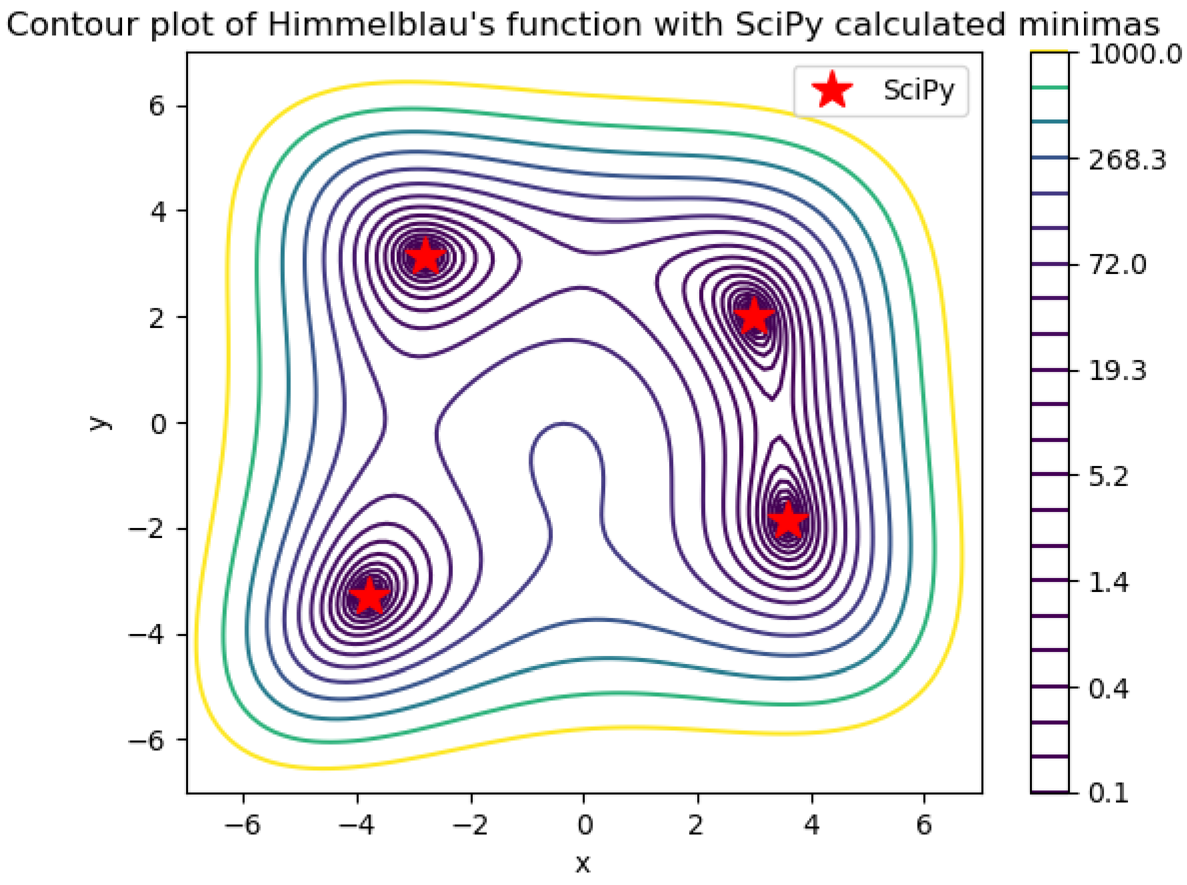
Part (c)

Based on the plot again, we can see 4 approximate locations of minima.

```
In [ ]: # Guesses
min_1, min_2, min_3, min_4 = [4, -3], [-4, -2], [3, 4], [-3, 3]
minimas = [min_1, min_2, min_3, min_4]

# Calculate minimas using scipy.optimize.fmin
for guess in range(len(minimas)):
    minimas[guess] = scipy.optimize.fmin(fp, minimas[guess])

# Plot minimas
for i in range(len(minimas)):
    if i == 0:
        ax2.plot(minimas[i][0], minimas[i][1], 'r*', markersize=15, label='SciI
    else:
        ax2.plot(minimas[i][0], minimas[i][1], 'r*', markersize=15)

ax2.set_title('Contour plot of Himmelblau\'s function with SciPy calculated min
ax2.legend()
fig2
```

```
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 35
         Function evaluations: 68
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 41
         Function evaluations: 79
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 38
         Function evaluations: 71
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 30
         Function evaluations: 59
```

Out[ ]:

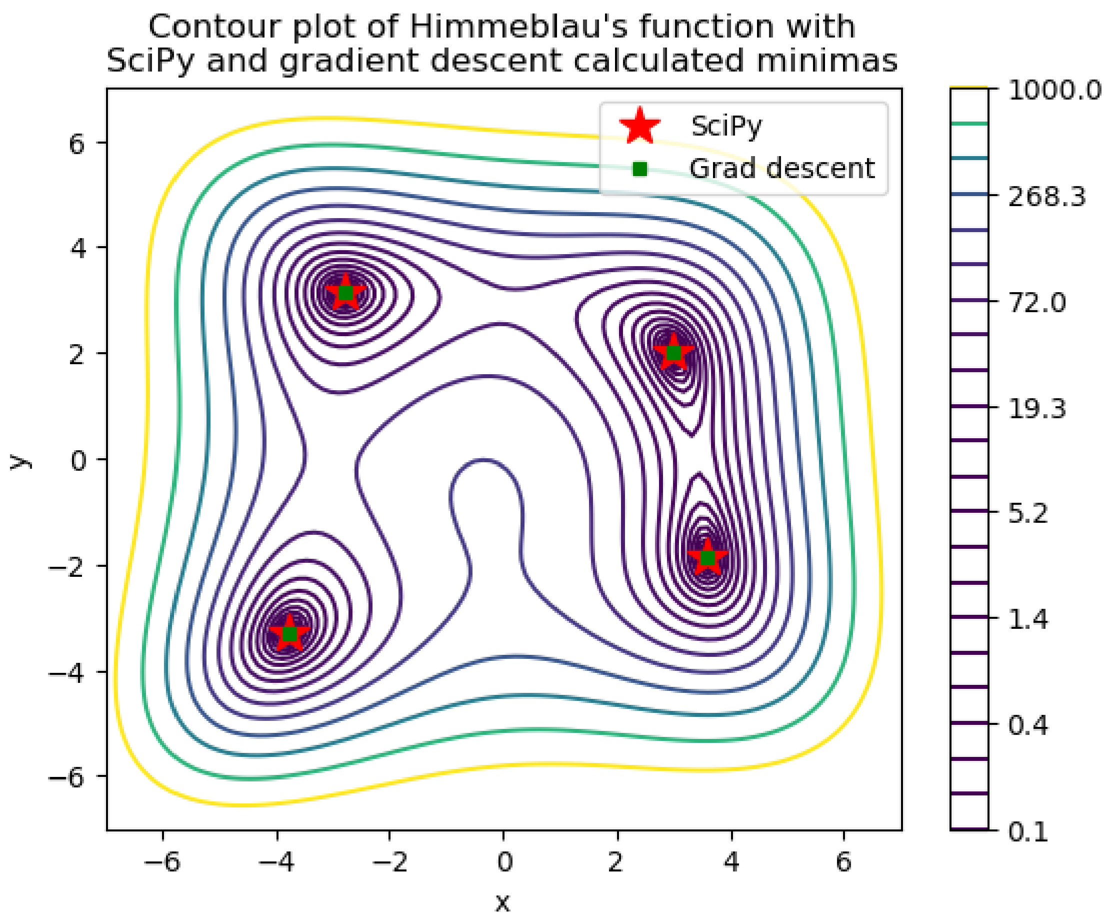


Part (d)

In [ ]:

```
gradf_xy = lambda x,y: np.array([4*x**3 - 42*x + 4*x*y + 2*y**2 - 14,
                                 4*y**3 - 26*y + 4*x*y + 2*x**2 - 22])
gradf = lambda p: gradf_xy(p[0], p[1])

tol = 10**-7

for i in range(len(minimas)):
    p = minimas[i]
    for j in range(2000):
        grad = gradf(p)
        if np.linalg.norm(grad) < tol:
            break
        phi = lambda t: p - t*grad
        f_phi = lambda t: fp(phi(t))

        t_min = scipy.optimize.fminbound(f_phi, 0, 1)
        min = phi(t_min)
        minimas[i] = min

# Plot gradient descent minimas
for i in range(len(minimas)):
    if i == 0:
        ax2.plot(minimas[i][0], minimas[i][1], 'gs', markersize=5, label='Grad
    else:
        ax2.plot(minimas[i][0], minimas[i][1], 'gs', markersize=5)

ax2.set_title('Contour plot of Himmeblau\'s function with\nSciPy and gradient c

ax2.legend()
fig2
```

Out[ ]:



Problem 2

Part (a)

In [ ]:

```
import time

tol = 1e-9
p = [2, 3]
```

```
init = time.time()
iterations = 0

for i in range(8000):
    grad = gradf(p)
    iterations += 1
    if np.linalg.norm(grad) < tol:
        break

    phi = lambda t: p - t*grad
    f_phi = lambda t: fp(phi(t))

    t_min = scipy.optimize.fminbound(f_phi, 0, 1)
    p = phi(t_min)

print('Gradient descent took', time.time() - init, 'seconds to converge.')
print('Iterations:', iterations)
```

```
Gradient descent took 0.004166841506958008 seconds to converge.
Iterations: 17
```

Part (b) - (d)

In [ ]:

```
import time

tol = 1e-9
p = [2, 3]
t_step = 0.025 # Step size

init = time.time()
iterations = 0

for i in range(8000):
    grad = gradf(p)
    iterations += 1
    if np.linalg.norm(grad) < tol:
        break
    else:
        p -= t_step * grad

print('Iterations:', iterations)
print('Time taken', time.time() - init)
```

```
Iterations: 8000
Time taken 0.1137230396270752
```

Part (e) - Results

	Number of Iterations	Time (s)	Converged (Yes/No)
tstep = 0.01	82	0.0022728443145751953	Yes
tstep = 0.02	56	0.001561880111694336	Yes
tstep = 0.025	8000	0.11360621452331543	No
`fminbound`	17	0.004582881927490234	Yes

Part (f) - Discussion

- I found that the Gradient descent algorithm does *not* always converge. In this problem, using a fixed step size of 0.025 led it to not converge. I believe the reason for this is that t = 0.025 is too large of a step size for this problem, and this leads the algorithm to keep hopping back and forth across the minimum but never quite reaching it as it did with smaller step sizes of 0.01 and 0.02.

- I found that gradient descent algorithm converged the fastest when we used the smallest step size of t = 0.01. It converged the slowest when using `fminbound`. Although the algorithm with a step size of t = 0.025 ran the longest, it did not converge in that case. I also found out that the calculated runtimes for each case are not a constant and change minutely at each run, but the relative order of fastest to slowest remains the same.

- I found that the gradient descent algorithm converged with the fewest number of iterations when using the `fminbound` function.

- No, my answers are different for the fastest runtime and the fewest iterations. I found that the fastest runtime for the algorithm to converge was when I used the smallest step size of t = 0.01 and the fewest iterations it took to converge was when it used the `fminbound` function. I think the reason for this to happen is that the math involved to adjust `p` when using a fixed step size is less expensive for the machine to process (`p -= tstep * grad`) than when using the `fminbound` function. When the algorithm uses the `fminbound` function, it solves an optimization problem at every iteration that usually tends to be more expensive to process, making the algorithm slower.

- **Optional:** This is essentially an optimization problem too. We can define a function that outputs the time taken by the gradient descent algorithm to run with the given step size. Then, we can use `scipy.optimize.fmin` on that function to return the `tstep` values for which the gradient descent algorithm will run the fastest. The only catch here is that the calculated runtime is not constant for a given `tstep` size as it varies at each instance, but it remains to close enough for a fair estimate. The code I wrote in the next cell implements this idea.

In [ ]:
```
def calculate_tstep_time(t_step):
    tol = 1e-9
    p = [2, 3]
    init = time.time()
    iterations = 0

    for i in range(8000):
        grad = gradf(p)
        iterations += 1
        if np.linalg.norm(grad) < tol:
            break
        else:
            p -= t_step * grad
    return time.time() - init

min_time = scipy.optimize.fmin(calculate_tstep_time, 0.1)
print('Minimum time:', min_time)
```

```
/var/folders/ng/hs3dclln4lv73c8vrw4wy0m40000gn/T/ipykernel_854/2980592599.py:
1: RuntimeWarning: overflow encountered in double_scalars
  gradf_xy = lambda x,y: np.array([4*x**3 - 42*x + 4*x*y + 2*y**2 - 14,
/var/folders/ng/hs3dclln4lv73c8vrw4wy0m40000gn/T/ipykernel_854/2980592599.py:
2: RuntimeWarning: overflow encountered in double_scalars
  4*y**3 - 26*y + 4*x*y + 2*x**2 - 22])
/var/folders/ng/hs3dclln4lv73c8vrw4wy0m40000gn/T/ipykernel_854/2980592599.py:
2: RuntimeWarning: invalid value encountered in double_scalars
  4*y**3 - 26*y + 4*x*y + 2*x**2 - 22])
/var/folders/ng/hs3dclln4lv73c8vrw4wy0m40000gn/T/ipykernel_854/2980592599.py:
1: RuntimeWarning: invalid value encountered in double_scalars
  gradf_xy = lambda x,y: np.array([4*x**3 - 42*x + 4*x*y + 2*y**2 - 14,
Optimization terminated successfully.
         Current function value: 0.101076
         Iterations: 16
         Function evaluations: 41
Minimum time: [0.11499878]
```