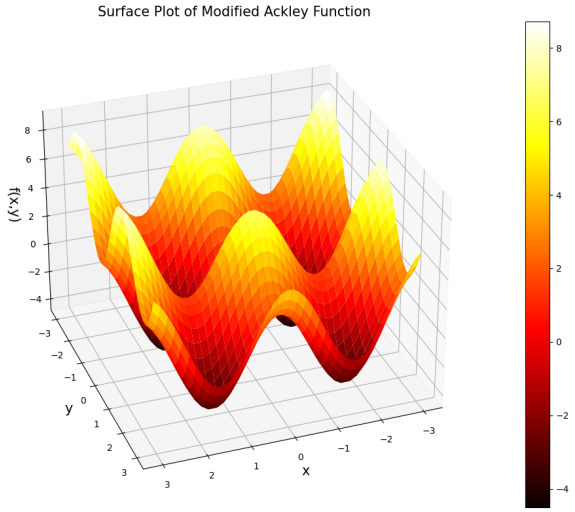
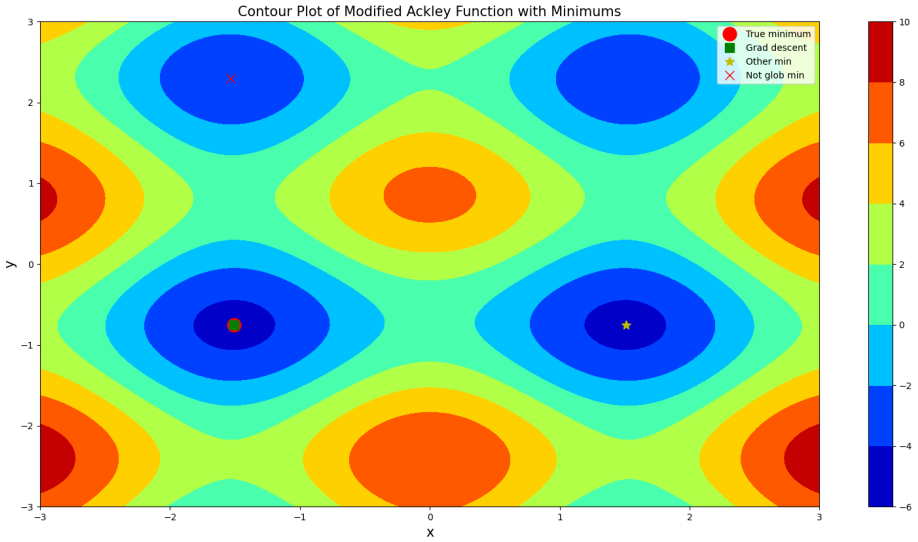
Homework 4 Writeup

Christian Diangco
AMATH 301 B

Problem 1






```
f = lambda x, y: np.exp(-0.2) * np.sqrt(x**2 + y**2) + 3 * (np.cos(2*x) + np.sin(2*y))
fv = lambda v: np.exp(-0.2) * np.sqrt(v[0]**2 + v[1]**2) + 3 * (np.cos(2*v[0]) + np.sin(2*v[1]))
x, y = np.meshgrid(np.linspace(-3,3,40), np.linspace(-3,3,40))
fig = plt.figure()
ax = fig.gca(projection="3d")
surf = ax.plot_surface(x, y, f(x,y), cmap=cm.hot)
fig.colorbar(surf)
ax.view_init(30,70)
ax.set_xlabel("x", fontsize=15)
ax.set_ylabel("y", fontsize=15)
```

```
ax.set_zlabel("f(x,y)", fontsize=15)
ax.set_title("Surface Plot of Modified Ackley Function", fontsize=15)

x, y = np.meshgrid(np.linspace(-3,3,100), np.linspace(-3,3,100))
fig2, ax2 = plt.subplots()
fcont = ax2.contourf(x, y, f(x,y), cmap=cm.jet)
ax2.set_xlabel("x", fontsize=15)
ax2.set_ylabel("y", fontsize=15)
fig2.colorbar(fcont)
ax2.set_title("Contour Plot of Modified Ackley Function with Minimums", fontsize=15)

argmin = np.genfromtxt("A3.csv", delimiter=",")
argmin2 = np.genfromtxt("A5.csv", delimiter=",")
guess = np.array([2,0])
argmin3 = scipy.optimize.fmin(fv, guess)
guess = np.array([-1,0.9])
argmin4 = scipy.optimize.fmin(fv, guess)
ax2.plot(argmin[0], argmin[1], "ro", label="True minimum", markersize=15)
ax2.plot(argmin2[0], argmin2[1], "gs", label="Grad descent", markersize=10)
ax2.plot(argmin3[0], argmin3[1], "y*", label="Other min", markersize=10)
ax2.plot(argmin4[0], argmin4[1], "rx", label="Not glob min", markersize=10)
ax2.legend()
```

Problem 2

	Number Iterations	Time	Converged(Yes/No)
tstep=0.01	198	0.00698399543762207	Yes
tstep=0.1	17	0.0009951591491699219	Yes
tstep=0.15	162	0.005984067916870117	Yes
tstep=0.2	10000	0.3530547618865967	No
fminbound	1527	1.9029085636138916	Yes

The Gradient Descent method **did not** always converge. It did not converge for the fixed time step method where `tstep = 0.2`. It did not converge because the time step is too large, causing it to overshoot the minimum.
The method that converged the fastest was the **time step method**. The method that converged in the fewest number of iterations was also the **time step method**. The time step method is faster because it doesn't have to find `tmin` for `f_of_phi` every iteration, which slows down the algorithm.

```
fgrad = lambda v: np.array([
    np.exp(-0.2) * (v[0] / np.sqrt(v[0]**2 + v[1]**2)) - 6*np.sin(2*v[0]),
    np.exp(-0.2) * (v[1] / np.sqrt(v[0]**2 + v[1]**2)) + 6*np.cos(2*v[1])
])
phi = lambda t: p - t*grad
f_of_phi = lambda t: fv(phi(t))

p = np.array([-1,-0.5])
start = time.time()
for i in range(10000+1):
    grad = fgrad(p)
    if np.linalg.norm(grad, np.inf) < 1e-10:
        break
    tmin = scipy.optimize.fminbound(f_of_phi,0,1)
    p = phi(tmin)
at = time.time() - start
ai = i

p = np.array([-1,-0.5])
tstep = 0.01
start = time.time()
for i in range(10000+1):
```

```
    grad = fgrad(p)
    if np.linalg.norm(grad, np.inf) < 1e-10:
        break
    p = p - tstep * grad
bt = time.time() - start
bi = i

p = np.array([-1,-0.5])
tstep = 0.1
start = time.time()
for i in range(10000+1):
    grad = fgrad(p)
    if np.linalg.norm(grad, np.inf) < 1e-10:
        break
    p = p - tstep * grad
ct = time.time() - start
ci = i

p = np.array([-1,-0.5])
tstep = 0.15
start = time.time()
for i in range(10000+1):
    grad = fgrad(p)
    if np.linalg.norm(grad, np.inf) < 1e-10:
        break
    p = p - tstep * grad
dt = time.time() - start
di = i

p = np.array([-1,-0.5])
tstep = 0.2
start = time.time()
for i in range(10000+1):
    grad = fgrad(p)
    if np.linalg.norm(grad, np.inf) < 1e-10:
        break
    p = p - tstep * grad
et = time.time() - start
ei = i
```