EECE 5550 HW3 Problem 1: Optimization Methods

To begin, make a copy of this notebook so you can save changes.

After completing these problems, please save this notebook as a pdf and combine it with your solutions to the written problems so that you can upload a single pdf to Gradescope.

These first few cells just implement some methods you may find useful

```
import numpy as np
import matplotlib.pyplot as plt
from typing import Callable


def draw_plot(xs: np.ndarray, f: Callable, k: int = 1) -> None:

    # Draw 2 plots:
    # On the left subplot, show:
    # - the function f(x) as contours
    # - the values of x as we search for the minimizer
    # On the right subplot, show:
    # - value of f(x) vs. current xi at each iteration of the optimization

    plt.subplots(1, 2)

    plt.subplot(1, 2, 2)
    z = f(xs)
    plt.plot(np.arange(len(z)), z)
    plt.xlabel('Iteration')
    plt.ylabel('f(x)')
    plt.gca().set_yscale('log')

    plt.subplot(1, 2, 1)

    # Optimization
    plt.plot(xs[:, 0], xs[:, 1], '-x')

    x_low = -2
    x_high = 2
    y_low = -2
    y_high = 2

    # Contour Plot
    num_xs = 20
    num_ys = 20
    x_vals = np.linspace(x_low, x_high, num_xs)
    y_vals = np.linspace(y_low, y_high, num_ys)
    X, Y = np.meshgrid(x_vals, y_vals)
    xy_pairs = np.vstack([X.ravel(), Y.ravel()]).T
    z = f(xy_pairs, k=k)
    Z = z.reshape(num_xs, num_ys)
    plt.gca().contour(X, Y, Z)

    plt.xlim([x_low, x_high])
    plt.ylim([y_low, y_high])
    plt.show()


def f(x: np.ndarray, k: int = 1) -> float:
    # f(x) = x^2 - xy + ky^2
    if x.ndim == 1:
        return x[0]**2 - x[0]*x[1] + k*x[1]**2
    elif x.ndim == 2:
        return x[:, 0]**2 - x[:, 0]*x[:, 1] + k*x[:, 1]**2
```

1.1 [2pts] Implement the gradient and hessian of f(x; k)

```
def gradf(x: np.ndarray, k: int = 1) -> np.ndarray:
    return np.array([2 * x[0] - x[1], -x[0] + 2 * k * x[1]])

def hessf(x: np.ndarray, k: int = 1) -> np.ndarray:
    return np.array([[2, -1], [-1, 2 * k]])
```

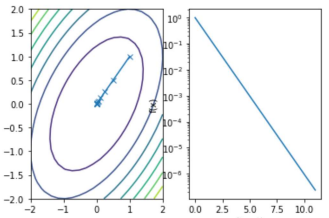
1.2 [2pts] Implement Gradient Descent

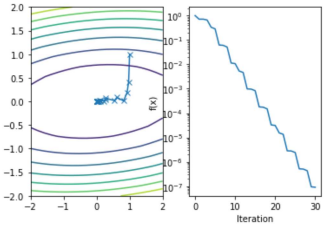
```
def gradient_descent(
        f: Callable,
        gradf: Callable,
        x0: np.ndarray,
        c: float,
        tau: float,
        epsilon: float,
        plot: bool = False,
        k: int = 1,
        ) -> np.ndarray:
    xs = [x0]
    while True:
```

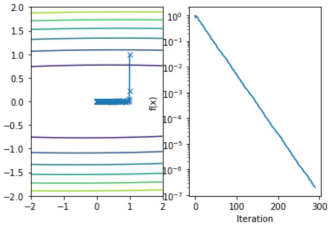
```
        x = xs[-1]
        p = -gradf(x, k)
        if np.linalg.norm(p) < epsilon:
            break
        alpha = 1
        while f(x + alpha*p, k) > f(x, k) - c*alpha*np.linalg.norm(p)**2:
            alpha *= tau
        xs.append(x + alpha*p)
    xs = np.array(xs)
    if plot:
        draw_plot(xs, f, k=k)
    return xs
```

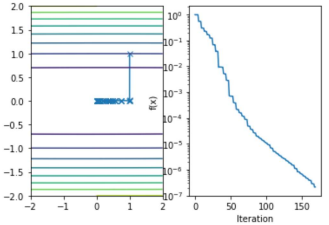
1.2 [1pt] Run your gradient descent for various values of κ

```
x0 = np.array([1., 1.])
c = 0.5
tau = 0.5
epsilon = 0.001
xs = gradient_descent(f, gradf, x0, c, tau, epsilon, k=1, plot=True)
xs = gradient_descent(f, gradf, x0, c, tau, epsilon, k=10, plot=True)
xs = gradient_descent(f, gradf, x0, c, tau, epsilon, k=100, plot=True)
xs = gradient_descent(f, gradf, x0, c, tau, epsilon, k=1000, plot=True)
```









For full credit on this problem, add some text description of what you observe about these results and why that is occurring (replace this text cell with your own thoughts).

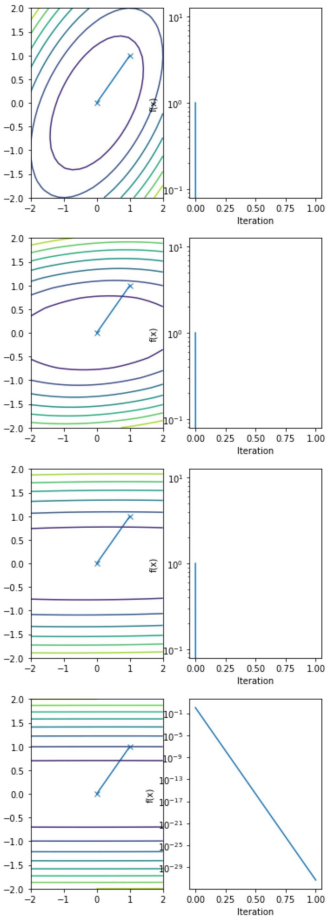
1.4 [2pts] Implement Newton's Method

```
def newton(
        f: Callable,
        gradf: Callable,
        hessf: Callable,
        x0: np.ndarray,
        epsilon: float,
```

```
    plot: bool = False,
    k: int = 1
) -> np.ndarray:
    xs = [x0]
    while True:
        x = xs[-1]
        grad = gradf(x, k=k)
        hess = hessf(x, k=k)
        if np.linalg.norm(grad) < epsilon:
            break
        xs.append(x - np.linalg.inv(hess) @ grad)
    xs = np.array(xs)
    if plot:
        draw_plot(xs, f, k=k)
    return xs
```

1.5 [1pt] Run your Newton's method for various values of κ

```
x0 = np.array([1., 1.])
epsilon = 0.001
xs = newton(f, gradf, hessf, x0, epsilon, k=1, plot=True)
xs = newton(f, gradf, hessf, x0, epsilon, k=10, plot=True)
xs = newton(f, gradf, hessf, x0, epsilon, k=100, plot=True)
xs = newton(f, gradf, hessf, x0, epsilon, k=1000, plot=True)
```



For full credit on this problem, add some text description of what you observe about these results and why that is occurring (replace this text cell with your own thoughts).

Question 2

(a) From $\mathbb{R}^n$ to $\mathbb{R}^n$ is in the same dimension space

It translates as a vector addition with $L_v(x) = x + v$

(b) $dL_v(x) = I$

(c) Based on $V_w(x) \triangleq d(L_x)_e(w)$

$\because dL_v(x) = I$

$= I(\xi) = \xi$

Since ξ is constant, the Left-invariant vector field is also constant. Which means the field does not get influence from fluctuation and transformation as long as ξ is constant.

(d) The group operation is matrix multiplication.

from $GL(n) \to GL(n)$

The left-translation map is $A \cdot x$ with x being the variant from $L_g(x)$

(e) the derivative $dL_A = A$

(f) $V_\Omega = d(L_x)_e(w) = x \cdot I(\Omega)$

$= x\Omega$

Question 3

(a) $\Omega^2 = \begin{pmatrix} -\omega^2 & 0 \\ 0 & -\omega^2 \end{pmatrix}$ or $(-\omega^2)\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

$\Omega^3 = \begin{pmatrix} 0 & -\omega^3 \\ \omega^3 & 0 \end{pmatrix}$ $\Omega^4 = \begin{pmatrix} \omega^4 & 0 \\ 0 & \omega^4 \end{pmatrix} = \omega^4 \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

For Odd k: $\Omega^k = (-1)^{\lfloor \frac{k}{2} \rfloor} \cdot \omega^k \cdot \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

For Even k: $\Omega^k = \omega^k \cdot \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \cdot (-1)^{\lfloor \frac{k}{2} \rfloor}$

$S = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$

Or write as $(-1)^{\lfloor \frac{k}{2} \rfloor} \cdot \omega^k \cdot \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}^{(k\%2)}$

(b) $\exp(x) = \sum_{k=0}^{\infty} \frac{x^k}{k!} = \sum_{k=0}^{\infty} \frac{\Omega^k}{k!} = \sum_{k=0}^{\infty} \frac{(-1)^{\lfloor \frac{k}{2} \rfloor} \cdot \omega^k \cdot S^{(k\%2)}}{k!}$

Based on even and odd:

Even k: $k\%2 = 0 \Rightarrow S^{k\%2} = I$

Odd k: $k\%2 = 1 \Rightarrow S^{k\%2} = S$

$p = \lfloor \frac{k}{2} \rfloor$

$$\text{Above} = \sum_{p=0}^{\infty} \frac{(-1)^p \omega^{2p} \cdot I}{(2p)!} + \sum_{p=0}^{\infty} \frac{(-1)^p \omega^{2p+1} \cdot S}{(2p+1)!}$$

$= \cos(\omega) I + \sin(\omega) \cdot S$ (Based on Taylor series)

$= \begin{pmatrix} \cos(\omega) & \sin(\omega) \\ -\sin(\omega) & \cos(\omega) \end{pmatrix}$

Geometrically, this is the rotation matrix in 2D space for ω

Question 4

(a) plug in $\gamma(0) = x$ and $\gamma(1) = y$

$x = x \cdot \exp(0)$ $\quad$ $y = x \cdot \exp(w)$

$w = \log\left(\frac{y}{x}\right) \Rightarrow \gamma(t) \triangleq x \cdot \exp\left(t \cdot \log\left(\frac{y}{x}\right)\right)$ is a curve on G

(b) For identity map: $\mathbb{R}^n \Rightarrow \mathbb{R}^n$

$\frac{y}{x}$ is equal to $y - x$

$\exp(t) = t$

The result for (a) is simplified as $\gamma(t) = x + t(y - x)$

This is basically a straight line in Lie Group (G)

(c) Let $X_0' = \begin{pmatrix} 0.433 & 0.1768 & 0.8839 & 1 \\ 0.25 & 0.9186 & -0.3062 & 1 \\ -0.86 & 0.3536 & 0.3536 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

$X_1' = \begin{pmatrix} 0.75 & -0.0474 & 0.6597 & 2 \\ 0.433 & 0.7891 & -0.4356 & 4 \\ -0.5 & 0.6124 & 0.6124 & 5 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

$\log(x^{-1} \cdot y) = \log\left(X_0'^{-1} \cdot X_1'\right) = \log \begin{pmatrix} 0.8706 & -0.3554 & -0.3554 & -3.1635 \\ 0.3517 & 0.9337 & -0.0662 & 4.7067 \\ 0.3517 & -0.0663 & 0.9337 & 1.7396 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

(Log of negative ?)