COMPSCI 371D Homework 0 (Prerequisites)

Some Plotting Code

In [1]:
```
%matplotlib inline

import numpy as np
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D    # Do not remove this import
from math import floor, ceil
import warnings


def cleanup_ticks(get_lim, set_ticks):
    lim = get_lim()
    lim = [ceil(lim[0]), floor(lim[1])]
    if lim[0] * lim[1] < 0:
        set_ticks([lim[0], 0, lim[1]])
    else:
        set_ticks([lim[0], lim[1]])
```

In [2]:
```
def plot_slices(function, eigenvectors, ax, variable_range=(-1, 1), samp
les=101):
    # We want division by zero to raise an exception, so we can print ou
r own warning and abort
    with warnings.catch_warnings():
        warnings.simplefilter("error")
        try:
            for i in range(2):
                eigenvectors[i] /= np.linalg.norm(eigenvectors[i])
        except RuntimeWarning:
            print('Zero-norm eigenvector(s). No plot produced')
        else:
            t = np.linspace(variable_range[0], variable_range[1], num=sa
mples)
            for plot in range(2):
                x, y = (eigenvectors[plot][component] * t for component
in range(2))
                ax.plot(t, function(x, y), label='Along v_{}'.format(plo
t + 1))
            cleanup_ticks(ax.get_xlim, plt.xticks)
            cleanup_ticks(ax.get_ylim, plt.yticks)
            plt.legend()
            plt.xlabel('t')
```

In [3]:
```
def plot_function(function, ax, variable_range=(-1, 1), samples=101):
    t = np.linspace(variable_range[0], variable_range[1], num=samples)
    x, y = np.meshgrid(t, t)
    ax.plot_surface(x, y, function(x, y), cmap=plt.get_cmap('viridis'))
    cleanup_ticks(ax.get_xlim, ax.set_xticks)
    cleanup_ticks(ax.get_ylim, ax.set_yticks)
    cleanup_ticks(ax.get_zlim, ax.set_zticks)
    plt.xlabel('x')
    plt.ylabel('y')
```

In [4]:
```
def plot_both(name, function, eigenvectors, fig, variable_range=(-1, 1),
samples=101):
    subplot_1 = fig.add_subplot(1, 2, 1, projection='3d')
    plot_function(function, subplot_1, variable_range=variable_range, sa
mples=samples)
    subplot_2 = fig.add_subplot(1, 2, 2)
    plot_slices(function, eigenvectors, subplot_2, variable_range=variab
le_range,
                samples=samples)
    fig.suptitle('{}(x, y)'.format(name))
```

In [5]:
```
def answer(name, function, eigenvectors):
    print('(5)')
    figure = plt.figure(figsize=(12, 5))
    plot_both(name, function, eigenvectors, figure)
    plt.show()
```

Part 1: Gradient and Hessian

(1) Gradient and Hessian:

$$\nabla ?(\mathbf{x}) = \begin{bmatrix} ? \\ ? \end{bmatrix} \qquad H_?(\mathbf{x}) = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix}$$

(2) Gradient and Hessian at $\mathbf{x}_0 = (0, 0)$:

$$\nabla ?(\mathbf{x}_0) = \begin{bmatrix} ? \\ ? \end{bmatrix} \qquad H_?(\mathbf{x}) = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix}$$

(3) Eigenvalues and eigenvectors of the Hessian at $\mathbf{x}_0$:

$$\lambda_1 = ?\ , \quad \lambda_2 = ?\ , \quad \mathbf{v}_1 = \begin{bmatrix} ? \\ ? \end{bmatrix}\ , \quad \mathbf{v}_2 = \begin{bmatrix} ? \\ ? \end{bmatrix}$$

(4) The point $\mathbf{x}_0$ is a ? because ?.

Problem 1.1

$$d(x, y) = x^2 + 2y$$

(1) Gradient and Hessian:

$$\nabla d(\mathbf{x}) = \begin{bmatrix} 2x \\ 2 \end{bmatrix} \qquad H_d(\mathbf{x}) = \begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix}$$

(2) Gradient and Hessian at $\mathbf{x}_0 = (0, 0)$:

$$\nabla d(\mathbf{x}_0) = \begin{bmatrix} 0 \\ 2 \end{bmatrix} \qquad H_d(\mathbf{x}) = \begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix}$$

(3) Eigenvalues and eigenvectors of the Hessian at $\mathbf{x}_0$:

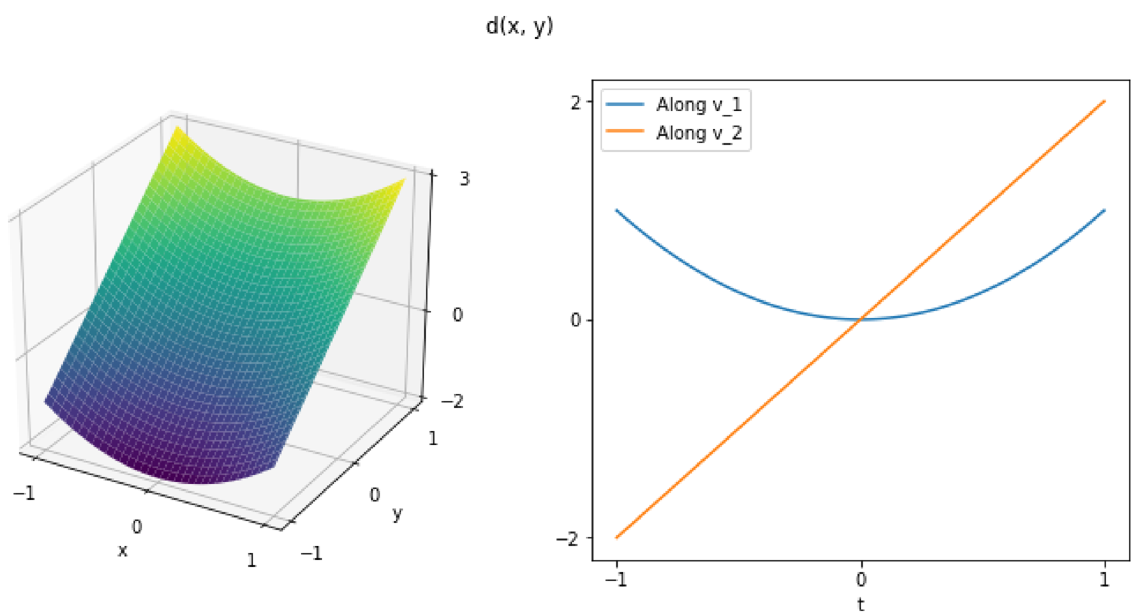
$$\lambda_1 = 2\,, \quad \lambda_2 = 0\,, \quad \mathbf{v}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}\,, \quad \mathbf{v}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

(4) The point $\mathbf{x}_0$ is a regular point because the gradient isn't 0 and the function is differentiable at that point.

In [6]:
```
def d(x, y):
    return (np.power(x, 2.0) + 2.0 * np.power(y, 1.0))


d_eigenvectors = [np.array([1.0, 0.0]), np.array([0.0, 1.0])]
answer('d', d, d_eigenvectors)
```

(5)



Problem 1.2

$$e(x, y) = \frac{1}{3}(y - x^2)^3$$

(1) Gradient and Hessian:

$$\nabla e(\mathbf{x}) = \begin{bmatrix} -2x(y-x^2)^2 \\ (y-x^2)^2 \end{bmatrix} \qquad H_e(\mathbf{x}) = \begin{bmatrix} 8x^2(y-x^2) & -4x(y-x^2) \\ -4(y-x^2) & 2(y-x^2) \end{bmatrix}$$

(2) Gradient and Hessian at $\mathbf{x}_0 = (0,0)$:

$$\nabla e(\mathbf{x}_0) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad H_e(\mathbf{x}) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

(3) Eigenvalues and eigenvectors of the Hessian at $\mathbf{x}_0$:

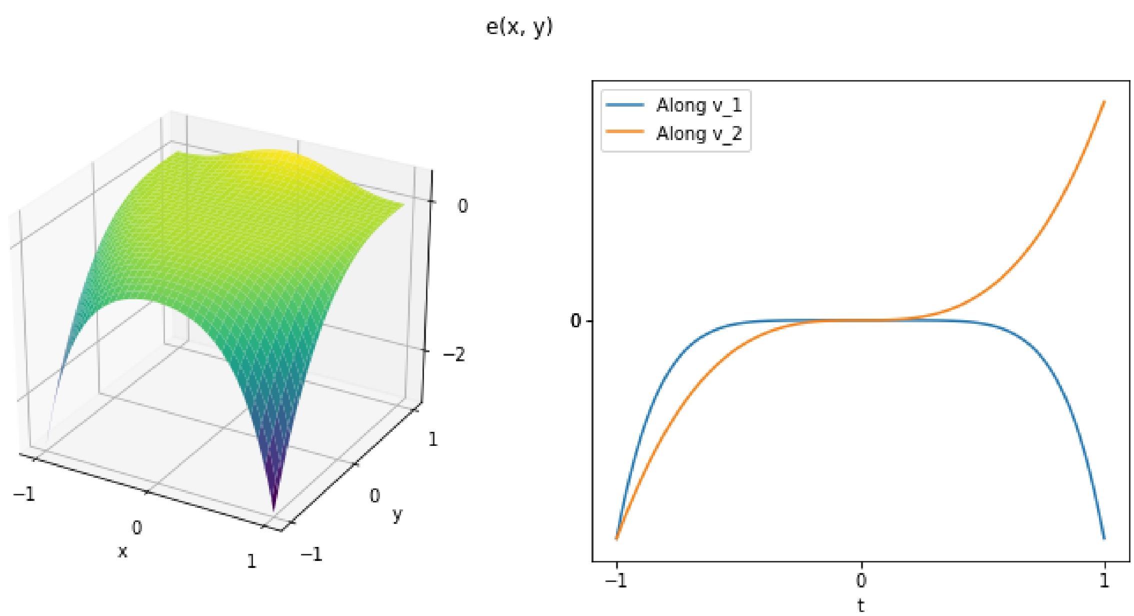
$$\lambda_1 = 0\ , \quad \lambda_2 = 0\ , \quad \mathbf{v}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}\ , \quad \mathbf{v}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

(4) The point $\mathbf{x}_0$ is a nonisolated minimum because although it is a minimum, plugging in e(tv2) gives us a function of 1/3t^3, which isn't strictly positive in all directions.

In [7]:
```
def e(x, y):
    return np.power((np.power(y, 1) - np.power(x, 2)),3)/ 3.0


e_eigenvectors = [np.array([1.0, 0.0]), np.array([0.0, 1.0])]
answer('e', e, e_eigenvectors)
```

```
(5)
```



Problem 1.3

$$f(x, y) = \frac{1}{2}x^2y^2$$

In [ ]:

(1) Gradient and Hessian:

$$\nabla f(\mathbf{x}) = \begin{bmatrix} xy^2 \\ x^2y \end{bmatrix} \qquad H_f(\mathbf{x}) = \begin{bmatrix} y^2 & 2xy \\ 2xy & x^2 \end{bmatrix}$$

(2) Gradient and Hessian at $\mathbf{x}_0 = (0, 0)$:

$$\nabla f(\mathbf{x}_0) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad H_?(\mathbf{x}) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

(3) Eigenvalues and eigenvectors of the Hessian at $\mathbf{x}_0$:

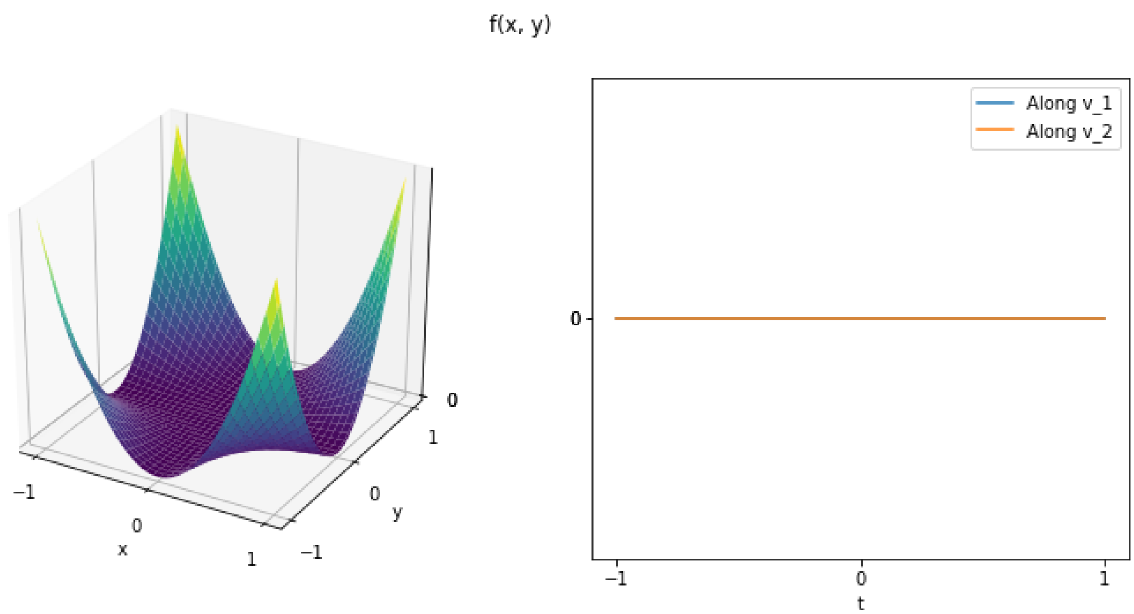
$$\lambda_1 = 0\ , \quad \lambda_2 = 0\ , \quad \mathbf{v}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}\ , \quad \mathbf{v}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

(4) The point $\mathbf{x}_0$ is an isolated saddle point because along v1 or v2, the values don't increase.

In [8]:
```
def f(x, y):
    return (np.power(x, 2) * np.power(y, 2))/ 2.0


f_eigenvectors = [np.array([1.0, 0.0]), np.array([0.0, 1.0])]
answer('f', f, f_eigenvectors)
```

(5)



Part 2: Fitting Sums of Functions

Problem 2.1

Because the function only has domain from 0 to 1, we must use a fourier sum to approxime the function f(x) since using only sine wouldn't work. Since sine is only positive on 0 to 1, if the function were to be negative, we would not be able to create an accurate approximation, and using both sine and cosine (which becomes negative after pi/2) enables us to be more confident in creating a better approximation.

Problem 2.2

In [9]:
```
import numpy as np

def Fourier(k):
    if k % 2:
        return lambda x: np.sin(np.pi * (k + 1) * x)
    else:
        return lambda x: np.cos(np.pi * k * x)
```

In [10]:
```
Fourier(3)(np.array([1.2, 0.4]))
```

Out[10]:
```
array([ 0.58778525, -0.95105652])
```

In [11]:
```
def evaluate_basis(x, basis, K):
    a = []
    for i in range(K):
        a.append(basis(i)(x))
    a = np.array(a)
    a = a.T
    return a
```

In [12]:
```
evaluate_basis(np.array([.1, .12, .7]), Fourier, 5)
```

Out[12]:
```
array([[ 1.        ,  0.58778525,  0.80901699,  0.95105652,  0.3090169
9],
       [ 1.        ,  0.68454711,  0.72896863,  0.99802673,  0.0627905
2],
       [ 1.        , -0.95105652, -0.30901699,  0.58778525, -0.8090169
9]])
```

Problem 2.3

In [13]:
```
def polynomial(k):
    return lambda x: np.power(x, k)
```

In [14]:
```
evaluate_basis(np.array([1, 2, 3]), polynomial, 5)
```

Out[14]:
```
array([[ 1,  1,  1,  1,  1],
       [ 1,  2,  4,  8, 16],
       [ 1,  3,  9, 27, 81]])
```

Problem 2.4

In [15]:
```
T_exact = {'x': [1, 2, 4], 'y': [2, -1, 3]}
```

In [16]:
```
def fit(T, K, basis=Fourier):

    A = evaluate_basis(T_exact['x'], basis, K)
    b = T_exact['y']
    b = np.array(b)
    printing(T_exact['x'],b, K, basis=Fourier)
    return np.linalg.lstsq(A, b, rcond=None)[0]
```

In [17]:
```
fit(T_exact, 3, basis=polynomial)
```

```
-----------------------------------------------------------------------
----
NameError                                 Traceback (most recent call l
ast)
<ipython-input-17-49789e7ee2f7> in <module>
----> 1 fit(T_exact, 3, basis=polynomial)

<ipython-input-16-2bd3e2b16506> in fit(T, K, basis)
      4     b = T_exact['y']
      5     b = np.array(b)
----> 6     printing(T_exact['x'],b, K, basis=Fourier)
      7     return np.linalg.lstsq(A, b, rcond=None)[0]

NameError: name 'printing' is not defined
```

In [ ]:
```
def plotter(x, y, basis, K):
    plt.scatter(T_exact['x'], T_exact['y'])
    plt.xlabel('x')
    plt.ylabel('y')
   # plt.show()
```

In [ ]:
```
def printing(x, y, basis, K):
    plotter(x, y, basis, K)
    plt.show()
```

Problem 2.5

In [ ]:
```
T_over = {'x': [1, 2, 3, 4], 'y': [2, -1, 1, 3]}
```

Problem 2.6

In [ ]:
```
T_f = {'x': [.1, .12, .7, .85], 'y': [2, -1, 1, 3]}
```

Part 3: Probability

Problem 3.1

$$p(x): \begin{array}{ccccc} 0.16 & 0.17 & 0.11 & 0.23 & 0.34 \\ \hline x_1 & x_2 & x_3 & x_4 & x_5 \end{array} \quad \text{and} \quad p(y): \begin{array}{c|c} y_1 & 0.26 \\ y_2 & 0.47 \\ y_3 & 0.27 \end{array}$$

X and Y are not independent because P(X1|Y1)!=P(X1)*P(Y1)

Problem 3.2

$$p(x|y): \begin{array}{c|ccccc} y_1 & ? & ? & ? & ? & ? \\ y_2 & ? & ? & ? & ? & ? \\ y_3 & ? & ? & ? & ? & ? \\ \hline & x_1 & x_2 & x_3 & x_4 & x_5 \end{array}$$

$$p(y|x): \begin{array}{c|ccccc} y_1 & ? & ? & ? & ? & ? \\ y_2 & ? & ? & ? & ? & ? \\ y_3 & ? & ? & ? & ? & ? \\ \hline & x_1 & x_2 & x_3 & x_4 & x_5 \end{array}$$

Problem 3.3

(This problem is from *Mathematics for Machine Learning* by M. P. Deisenroth, A. A. Faisal, and C. S. Ong, Cambridge University Press, 2020. If you are rusty on the prerequisites, this may be a good book for you.)

Let H be the event that the coin comes up heads, which corresponds to the fruit coming from bag 1. Similarly, let T be the event that the coin comes up tails (the fruit comes from bag 2). Finally, let M be the event "a mango is drawn" and A be the event "an apple is drawn."

The problem asks to compute $p(T|M)$, the probability that the fruit comes from bag 2 given that it is a mango.

We have

$$p(H) = \frac{3}{5} \quad , \quad p(T) = 1 - p(H) = \frac{2}{5}$$

and, given the compositions of the contents in the two bags,

$$p(M|H) = \frac{2}{3}\,, \quad p(A|H) = \frac{1}{3}\,,$$

$$p(M|T) = \frac{1}{2}\,, \quad p(A|T) = \frac{1}{2}\,.$$

$p(T|M)$ = $p(M|T)$ $P(T)$ / $P(M)$

$P(M)$ = $P(H)$ $P(M|H)$ + $P(T)$ $(M|T)$

$p(T|M)$ = (1/2 *2/5*)/((3/5 2/3)+(2/5 * 1/2)) = 1/3

```
In [ ]:
```