COMPSCI 371D Homework 1

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), samples=101):
    # We want division by zero to raise an exception, so we can print our own warning a
nd 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=samples)
            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(plot + 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, samples=samples)
    subplot_2 = fig.add_subplot(1, 2, 2)
    plot_slices(function, eigenvectors, subplot_2, variable_range=variable_range,
                samples=samples)
    fig.suptitle('{}(x, y)'.format(name))
```

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

Part 1: Gradient and Hessian

Problem 1.1

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

(1) Gradient and Hessian:

$$\nabla d(\mathbf{x}, \mathbf{y}) = \begin{bmatrix} 4x \\ 1 \end{bmatrix} \qquad H_d(\mathbf{x}) = \begin{bmatrix} 4 & 0 \\ 0 & 0 \end{bmatrix}$$

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

$$\nabla d(\mathbf{x}_0) = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad H_d(\mathbf{x}) = \begin{bmatrix} 4 & 0 \\ 0 & 0 \end{bmatrix}$$

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

$$\lambda_1 = 4\ , \quad \lambda_2 = 0\ , \quad \mathbf{v}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}\ , \quad \mathbf{v}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

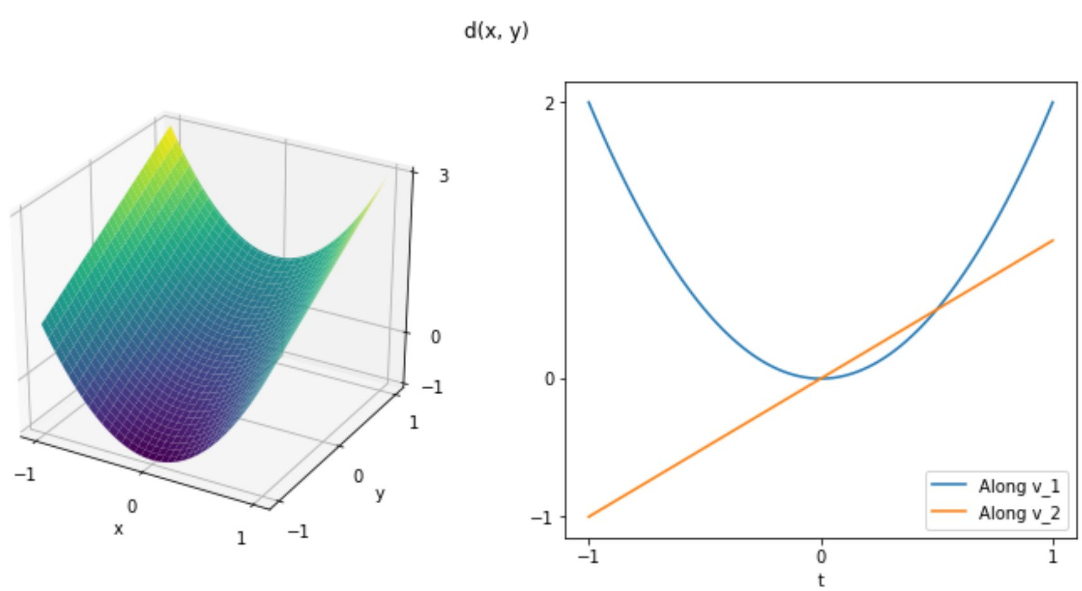
(4) The point $\mathbf{x}_0$ is an isolated regular point. It is a regular point because along $\mathbf{v}_2$, the function d becomes $\nabla d(t\mathbf{v}_2) = t$, with the point $t = 0$, being neither a min or max.

It is isolated because along $\mathbf{v}_2$, d is linear while along $\mathbf{v}_2$, d becomes a parabola.

In [6]:
```
def d(x, y):
    return (2.0 * np.power(x, 2) + y)

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

```
(5) Plot of The Function
```



Problem 1.2

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

(1) Gradient and Hessian:

$$\nabla e(\mathbf{x},\mathbf{y}) = \begin{bmatrix} x-3y \\ -3x+9y \end{bmatrix} \qquad H_e(\mathbf{x}) = \begin{bmatrix} 1 & -3 \\ -3 & 9 \end{bmatrix}$$

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

$$\nabla e(\mathbf{x}_0) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad H_e(\mathbf{x}) = \begin{bmatrix} 1 & -3 \\ -3 & 9 \end{bmatrix}$$

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

$$\lambda_1 = 10\ , \quad \lambda_2 = 0\ , \quad \mathbf{v}_1 = \begin{bmatrix} -1 \\ 3 \end{bmatrix}\ , \quad \mathbf{v}_2 = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

(4) The point $\mathbf{x}_0$ is a non-isolated minimum.

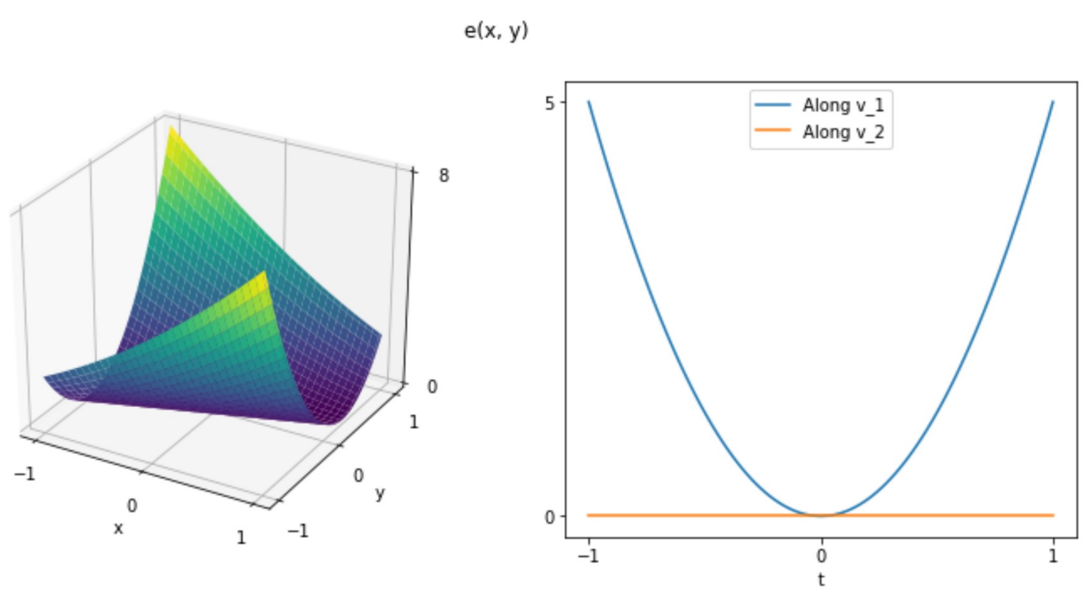
It is a minimum because along $\mathbf{v}_1$, the function e becomes $\frac{(10t)^2}{2}$, with the point $t = 0$, $e = 0$ which is the min of the new function.

But along $\mathbf{v}_2$, e becomes $e = 0$, so that means multiple points have the value $e = 0$, therefore $\mathbf{v}_1$ is not isolated.

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

e_evectors = [np.array([-1.0, 3.0]), np.array([3.0, 1.0])]
answer('e', e, e_evectors)
```

```
(5) Plot of The Function
```



Problem 1.3

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

(1) Gradient and Hessian:

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

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

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

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

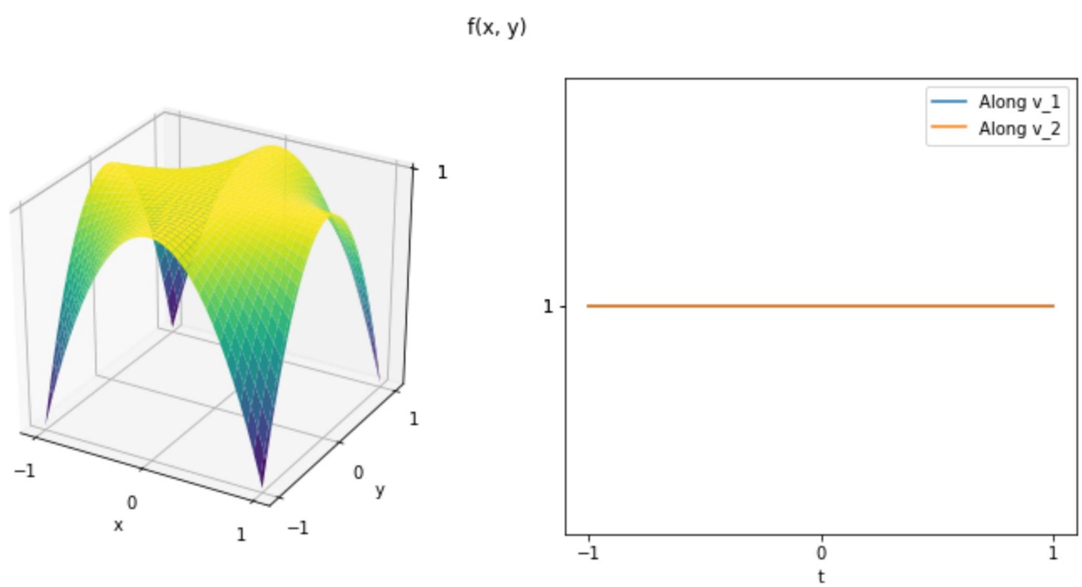
$$\lambda_1 = 0\,, \quad \lambda_2 = 0\,, \quad \mathbf{v}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}\,, \quad \mathbf{v}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

(4) The point $\mathbf{x}_0$ is a non-isolated maximum because along either $\mathbf{v}_1$ or $\mathbf{v}_2$, $f = 1$, so the maximum values is 1, but that is shared by many other points of t, so it is not isolated.

In [8]:
```
def f(x, y):
    return 1-(.5)*np.power(x,2)*np.power(y,2)

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

```
(5) Plot of The Function
```



Problem 1.4

$$g(x, y) = x \sin y$$

(1) Gradient and Hessian:

$$\nabla g(\mathbf{x}, \mathbf{y}) = \begin{bmatrix} \sin y \\ x \cos y \end{bmatrix} \qquad H_g(\mathbf{x}) = \begin{bmatrix} 0 & \cos y \\ \cos y & -x \sin y \end{bmatrix}$$

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

$$\nabla g(\mathbf{x}_0) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad H_g(\mathbf{x}) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

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

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

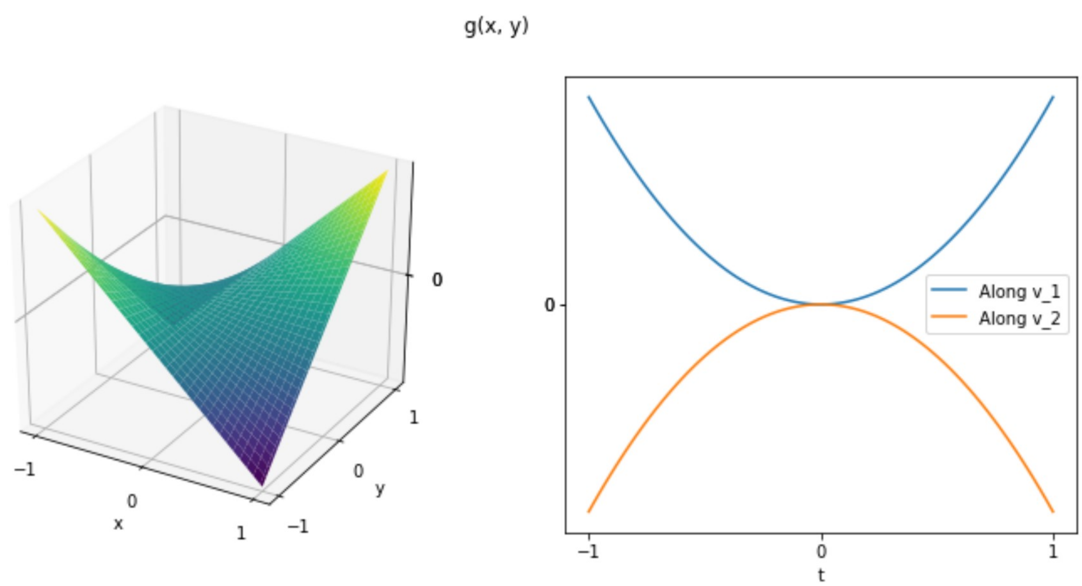
(4) The point $\mathbf{x}_0$ is an isolated saddle point.

It is a saddle point because the eigenvalues of the Hessian are postive and negative at $(0, 0)$. It is isolated because along $\mathbf{v}_1$, g is a postive parabola, so going beyond $t = 0$ in any directions will change g. This also applies if g goes along $\mathbf{v}_2$, but the parabola is negative. This makes it so that the point $(0, 0)$ is the only saddlepoint on g therefore it is isolated.

In [9]:
```
def g(x, y):
    return x * np.sin(y*(np.pi/180))

g_evectors = [np.array([1.0, 1.0]), np.array([-1.0, 1.0])]
answer('g', g, g_evectors)
```

```
(5) Plot of The Function
```



Part 2: Fitting Polynomials

In [10]:
```
import numpy as np
import matplotlib.pyplot as plt


def show_polynomials(T, polynomials=()):
    for key, value in T.items():
        T[key] = np.array(value)
    xs, ys = T.values()
    plt.plot(xs, ys, marker='.', markersize=12, ls='')
    number_of_plot_points = 100
    x_range = [xs - 1, xs + 1] if xs.size == 1 else [np.amin(xs), np.amax(xs)]
    x = np.linspace(x_range[0], x_range[1], number_of_plot_points)
    for polynomial in polynomials:
        number_of_coefficients = len(polynomial)
        y = np.zeros(x.shape)
        x_power = np.ones(x.shape)
        for k in range(number_of_coefficients):
            y += polynomial[k] * x_power
            x_power *= x
        plt.plot(x, y, label='degree ' + str(number_of_coefficients - 1))
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend()
    plt.show()
```

In [11]:
```
def round4(L):
    roundedList = []
    for x in L:
        roundedList.append(round(x,4))
    return roundedList
```

Problem 2.1

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 4 & 16 \end{bmatrix} \qquad b = \begin{bmatrix} 2 \\ -1 \\ 3 \end{bmatrix}$$

Problem 2.2

In [12]:
```
T_exact = {'x': [1, 2, 4], 'y': [2, -1, 3]}
```

In [13]:
```
def fit(T, degree):
    x = T['x']
    y = T['y']
    A_list = []
    for i in range(len(x)):
        n = x[i]
        for k in range(degree+1):
            A_list.append(np.power(n,k))
    A_mat = np.reshape(A_list, (len(x),degree+1))
    b_mat = np.array(y)
    return np.linalg.lstsq(A_mat, b_mat, rcond=None)[0]

c1 = fit(T_exact, 2)
```

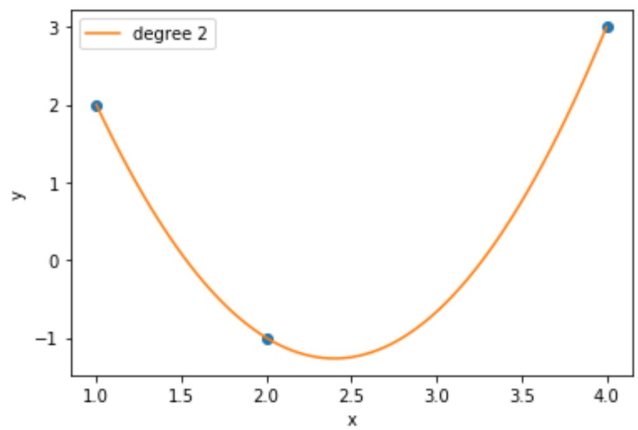
In [14]:
```
c1_rounded = round4(c1)
print("The vector or coefficients: ",c1_rounded)
```

```
The vector or coefficients:  [8.3333, -8.0, 1.6667]
```

In [15]:
```
c1_norm = np.linalg.norm(c1)
print("The Euclidean norm: ", round4([c1_norm]))
```

```
The Euclidean norm:  [11.6714]
```

In [16]:
```
show_polynomials(T_exact, [c1])
```



Problem 2.3

In [17]:
```
T_under = {'x': [1, 4], 'y': [2, 3]}
```

In [18]:
```
c2_linear = fit(T_under, 1)
c2_quad = fit(T_under, 2)
```

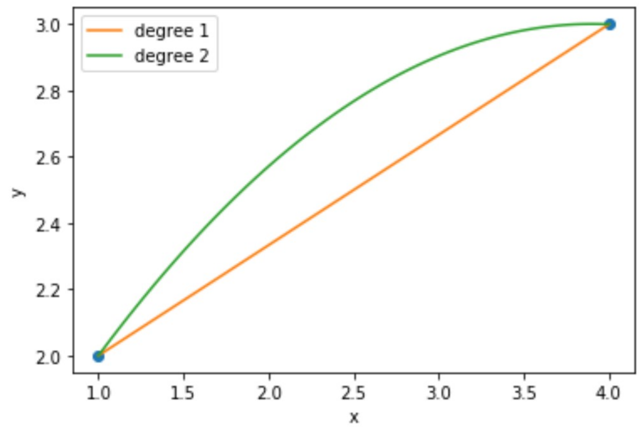
In [19]:
```
c2_linear_rounded = round4(c2_linear)
print("The coefficient vector for the linear fit: ",c2_linear_rounded)
c2_linear_norm = np.linalg.norm(c2_linear)
print("The norm of the linear fit: ", round4([c2_linear_norm]))
```

```
The coefficient vector for the linear fit:  [1.6667, 0.3333]
The norm of the linear fit:  [1.6997]
```

In [20]:
```
c2_quad_rounded = round4(c2_quad)
print("The coefficient vector for the quadratic fit: ",c2_quad_rounded)
c2_quad_norm = np.linalg.norm(c2_quad)
print("The norm of the quadratic fit: ", round4([c2_quad_norm]))
```

```
The coefficient vector for the quadratic fit:  [1.1905, 0.9286, -0.119]
The norm of the quadratic fit:  [1.5145]
```

In [21]:
```
show_polynomials(T_under, [c2_linear, c2_quad])
```



Problem 2.4

Problem 2.5

In [22]:
```
T_over = {'x': [1, 2, 3, 4], 'y': [2, -1, 1, 3]}
```

In [23]:
```
c_over = []
degrees = [0,1,2,3,6]
for x in degrees:
    c_over.append(round4(fit(T_over,x)))
```

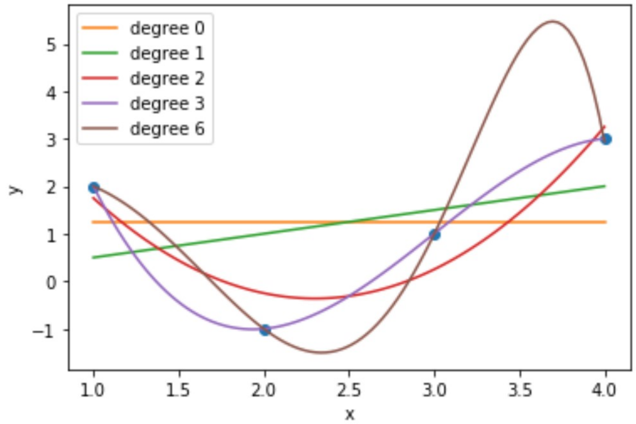
In [24]:
```
for z in range(5):
    print("The coefficient vector for the fit of degree", degrees[z], ": ", c_over[z])
```

```
The coefficient vector for the fit of degree 0 :  [1.25]
The coefficient vector for the fit of degree 1 :  [0.0, 0.5]
The coefficient vector for the fit of degree 2 :  [6.25, -5.75, 1.25]
The coefficient vector for the fit of degree 3 :  [15.0, -19.6667, 7.5, -0.8333]
The coefficient vector for the fit of degree 6 :  [1.6991, 1.1196, 0.2676, -0.6799, -
0.9654, 0.6533, -0.0942]
```

In [25]:
```
for z in range(5):
    print("The norm for the fit of degree", degrees[z], ": ", np.linalg.norm(c_over
[z]))
```

```
The norm for the fit of degree 0 :  1.25
The norm for the fit of degree 1 :  0.5
The norm for the fit of degree 2 :  8.584142356694699
The norm for the fit of degree 3 :  25.859688276930175
The norm for the fit of degree 6 :  2.458046059373176
```

In [26]:
```
show_polynomials(T_over, c_over)
```



Part 3: Eigenvalues and Eigenvectors

Problem 3.1

Let λ_1 and λ_2 be the eigenvalues of A, and $\mathbf{v}_1$ and $\mathbf{v}_2$ be the corresponding eigenvectors.

By definition:

$$A\mathbf{v}_1 = \lambda_1 \mathbf{v}_1$$

If we move over the $\lambda_1 \mathbf{v}_1$ to the left side of the equation, factor out the vector, and expand A, we will get:

$$\begin{bmatrix} a-\lambda_1 & 0 \\ b & c-\lambda_1 \end{bmatrix} \mathbf{v}_1 = 0$$

Since $\mathbf{v}_1$ cannot be the zero vector $A\lambda_1$ must equal zero. Therefore, we can set $a - \lambda_1 = 0$. From this, we can conclude that a is an eigenvalue of A.

Now, lets consider the transpose of A, A^T. By theorem, A^T has the same eigenvalues as A. Using this, we can find the second eigenvalue.

$$\begin{bmatrix} a & b \\ 0 & c \end{bmatrix}$$

Similar to how the first eigenvalue was found, we set $(A^T - \lambda_2)\mathbf{v}_2 = 0$. From this we get:

$$\begin{bmatrix} a-\lambda_2 & b \\ 0 & c-\lambda_2 \end{bmatrix} = 0$$

From this we get, $\lambda_2 = c$.

In conclusion, the eigenvalues of A are a and c. Then the real-valued, unit-norm eigenvectors associated with a and c are the following:

$$\mathbf{v}_1 = \frac{1}{\sqrt{1+(\frac{a-c}{b})^2}} \begin{bmatrix} \frac{a-c}{b} \\ 1 \end{bmatrix} \qquad \mathbf{v}_2 = \frac{1}{\sqrt{1+(\frac{c-a}{b})^2}} \begin{bmatrix} 1 \\ \frac{c-a}{b} \end{bmatrix}$$

Problem 3.2

(a)

Since g is a vertical line that eventaully goes to 0, the x-value for all points on g have to be 0. That makes the rule for the new y, become $y' = qy$. So y has to be any multiple of q.

(b)

For b, since the x-pos is only modified by m, the starting point for x should be a multiple of m.

For the y position, to be a straigt line, the $value$ has to decrease by the same ratio as x decreases. For this to happen it has to start at a multiple of $\frac{q-m}{p}$

(c)

They are the eigenvalues of the transformation that the rules for each step states. m and q are negative as well.