Demonstration of The Taylor Approximation

Name: Dahye Kim (김다혜)
ID: 20153712
Date: 27 September 2018
Project Github: [https://github.com/joyfuldahye/MFCVML_assignment02 (https://github.com/joyfuldahye/MFCVML_assignment02)](https://github.com/joyfuldahye/MFCVML_assignment02)

---

1. The first order approximation of a polynomial function of degree 2

1.1 Import packages for plotting graphs and manipulation data

In [2]:

```
import numpy as np
import matplotlib.pyplot as plt
```

1.2 Define a polynomial function of degree 2: $f(x) = x^2 + 5x + 2$

In [3]:

```
def myDeTwoFunction(x):
    Tf = x**2+5*x+2
    return Tf
```

1.3 Define the derivative of the function: $f'(x) = 2x + 5$

In [4]:

```
def myDerivativeFunction(x):
    Df = 2*x+5
    return Df
```

1.4 Define the first-order Taylor approximation of the function $\hat{f(x)} = f(z) + f'(z)(x - z)$

In [5]:

```
def firstTaylor(z):
    Ft = myDeTwoFunction(z) + myDerivativeFunction(z)*(x-z)
    return Ft
```

1.5 Define the domain of the function $x = [-100 : 0.1 : 100]$

In [6]:

```
x = np.arange(-100, 100, 0.1)
```

1.6 Compute the graph at example points

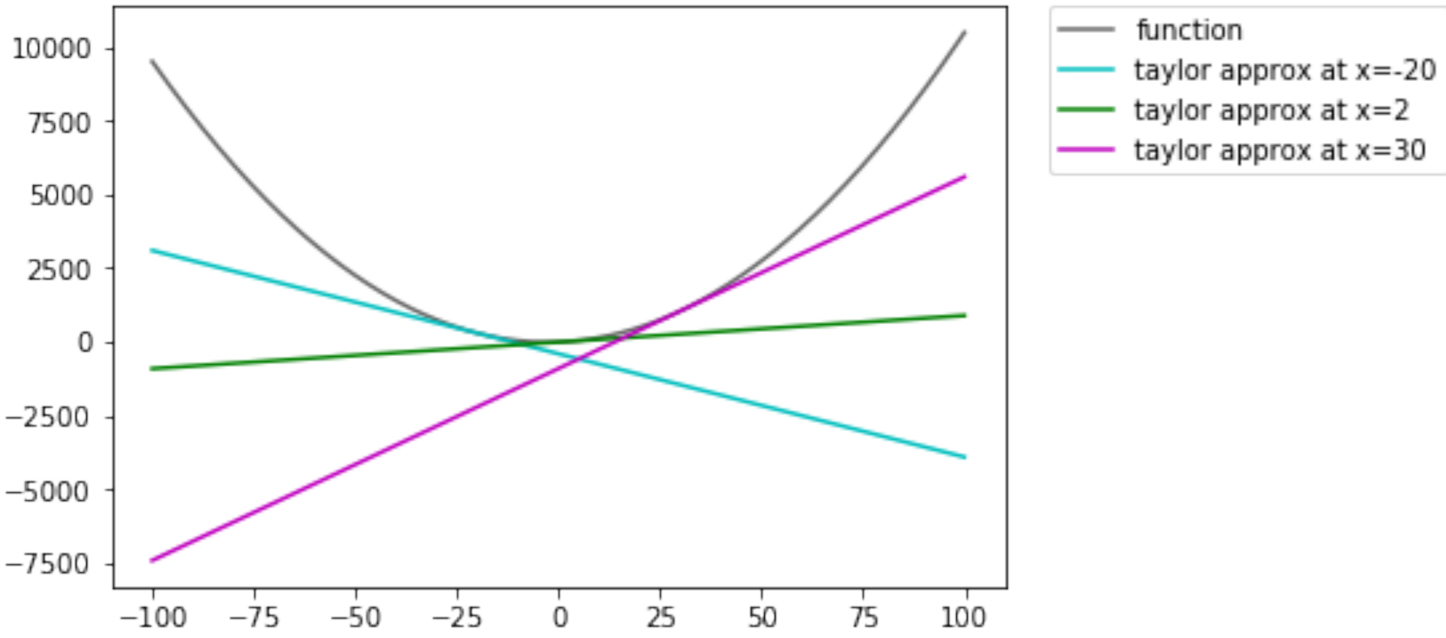
In [8]:

```
z1 = -20
z2 = 2
z3 = 30
Tf = myDeTwoFunction(x)
Df = myDerivativeFunction(x)
Ft1 = firstTaylor(z1)
Ft2 = firstTaylor(z2)
Ft3 = firstTaylor(z3)
```

1.7 Plot the graph for the function and its approximation

In [9]:

```
plt.figure(1)
plt.plot(x, Tf, 'dimgrey', label="function")
plt.plot(x, Ft1, 'c', label="taylor approx at x=-20")
plt.plot(x, Ft2, 'g', label="taylor approx at x=2")
plt.plot(x, Ft3, 'm', label="taylor approx at x=30")
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.show()
```



2. The first, second and third order approximations of a polynomial function of degree 4

2.1 Define a polynomial function of degree 4:

$$f(x) = x^4 - 5x^3 + 5x^2 + 5x - 6$$

In [10]:

```
def myFourDeFunction(x):
    Ff = x ** 4 - 5 * x ** 3 + 5 * x ** 2 + 5 * x - 6
    return Ff
```

2.2 Define the derivatives of the function for each order:

- First derivative: $f'(x) = 4x^3 - 15x^2 + 10x + 5$
- Second derivative: $f''(x) = 12x^2 - 30x + 10$
- Third derivative: $f'''(x) = 24x - 30$
- Third derivative: $f''''(x) = 24$

In [11]:

```
def my_first_derivative(x):
    return 4 * x ** 3 - 15 * x ** 2 + 10 * x + 5

def my_second_derivative(x):
    return 12 * x ** 2 - 30 * x + 10

def my_third_derivative(x):
    return 24 * x - 30

def my_fourth_derivative(x):
    return 24
```

2.3 Compute the derivatives at a certain point

In [12]:

```
def derivatives(z):
    dervs = [myFourDeFunction(z), my_first_derivative(z), my_second_derivative
(z), my_third_derivative(z), my_fourth_derivative(z)]
    return dervs
```

2.4 Set a factorial function which is used to compute the Taylor approximation:

- First order approximation: $\hat{f(x)} = f(z) + \frac{f'(z)}{1!}(x - z)$
- Second order approximation: $\hat{f(x)} = f(z) + \frac{f'(z)}{1!}(x - z) + \frac{f''(z)}{2!}(x - z)^2$
- Third order approximation: $\hat{f(x)} = f(z) + \frac{f'(z)}{1!}(x - z) + \frac{f''(z)}{2!}(x - z)^2 + \frac{f'''(z)}{3!}(x - z)^3$

In [13]:

```
def factorial(n):
    if n <= 0:
        return 1
    else:
        return n * factorial(n - 1)
```

2.5 Define the Taylor approximation of the function

In [14]:

```
def taylorApprox(z, k):
    der = derivatives(z)
    t = 0
    i = 0
    while i <= k:
        t = t + der[i] * (x-z)**i / factorial(i)
        i += 1
    return t
```

2.6 Define the domain of the function $x = [-2 : 0.1 : 4]$

In [15]:

```
x = np.arange(-2, 4, 0.1)
```

2.7 Compute the graphs at example points

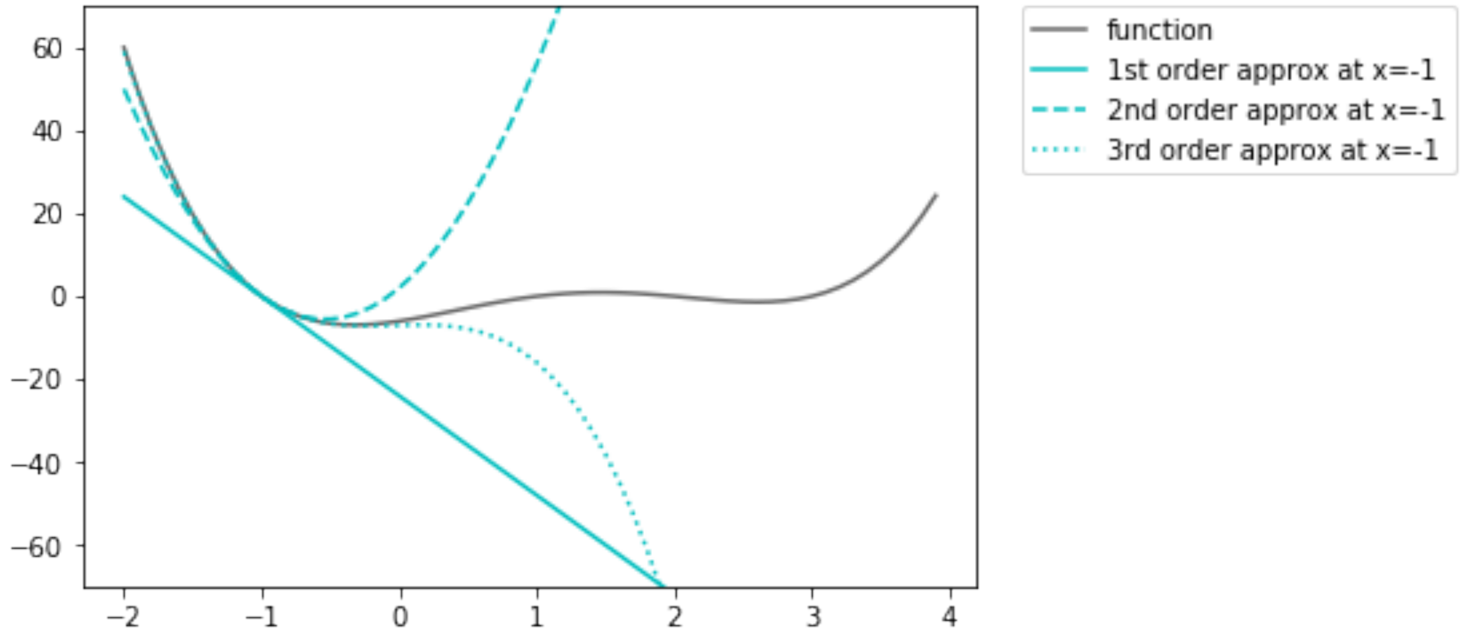
In [16]:

```
z1 = -1
z2 = 1
z3 = 3
f = myFourDeFunction(x)
ta11 = taylorApprox(z1, 1)
ta12 = taylorApprox(z1, 2)
ta13 = taylorApprox(z1, 3)
#ta14 = taylorApprox(z1, 4) // fourth order approx
ta21 = taylorApprox(z2, 1)
ta22 = taylorApprox(z2, 2)
ta23 = taylorApprox(z2, 3)
#ta24 = taylorApprox(z2, 4) // fourth order approx
ta31 = taylorApprox(z3, 1)
ta32 = taylorApprox(z3, 2)
ta33 = taylorApprox(z3, 3)
#ta34 = taylorApprox(z3, 4) // fourth order approx
```

2.8 Plot the graph for the function and its approximations at $x = -1$

In [17]:

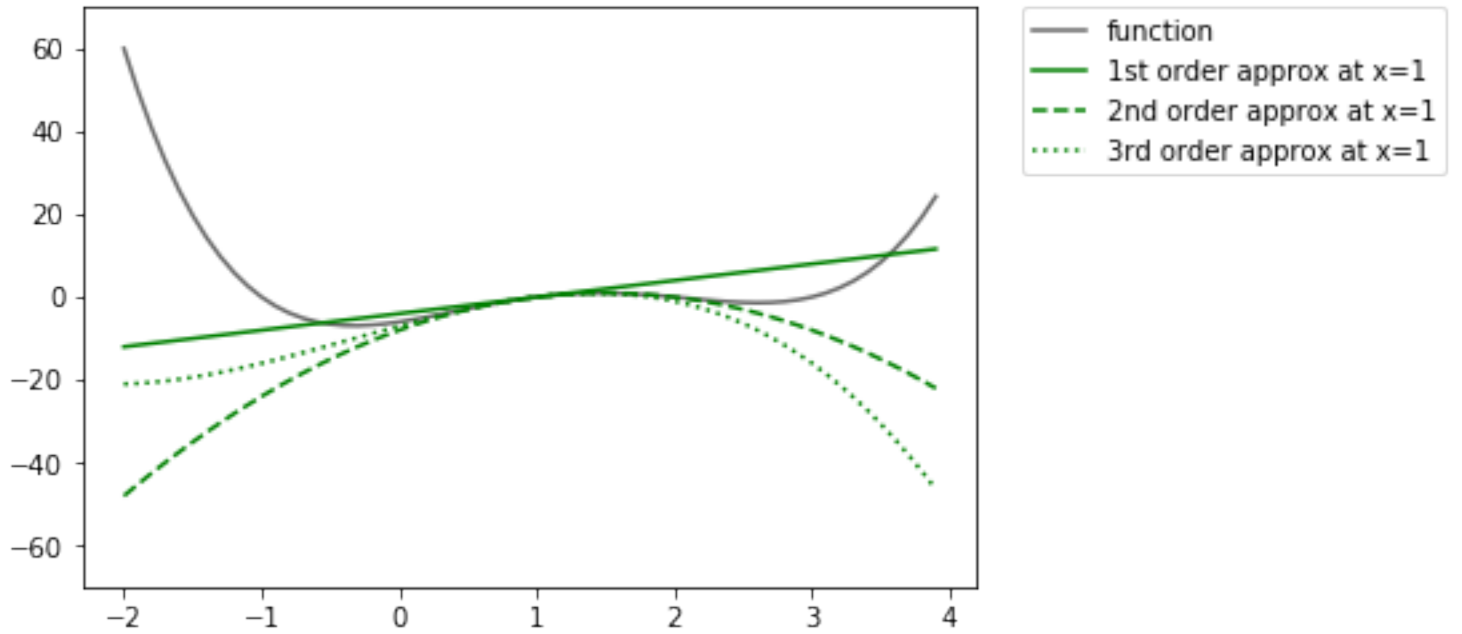
```
plt.figure(1)
plt.plot(x, f, 'dimgrey', label="function")
plt.plot(x, ta11, '-c', label="1st order approx at x=-1")
plt.plot(x, ta12, '--c', label="2nd order approx at x=-1")
plt.plot(x, ta13, ':c', label="3rd order approx at x=-1")
plt.ylim((-70, 70))
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.show()
```



2.9 Plot the graph for the function and its approximations at $x = 1$

In [18]:

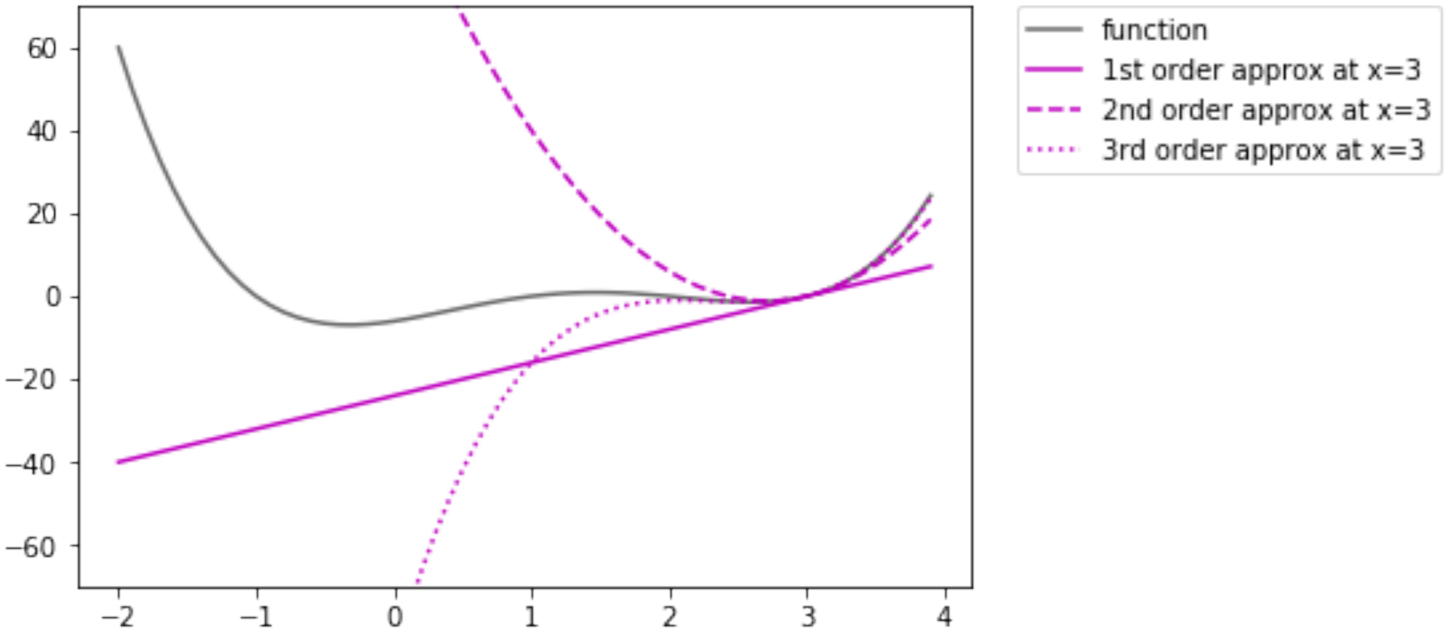
```
plt.figure(1)
plt.plot(x, f, 'dimgrey', label="function")
plt.plot(x, ta21, '-g', label="1st order approx at x=1")
plt.plot(x, ta22, '--g', label="2nd order approx at x=1")
plt.plot(x, ta23, ':g', label="3rd order approx at x=1")
plt.ylim((-70, 70))
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.show()
```



2.10 Plot the graph for the function and its approximations at $x = 3$

In [19]:

```
plt.figure(1)
plt.plot(x, f, 'dimgrey', label="function")
plt.plot(x, ta31, '-m', label="1st order approx at x=3")
plt.plot(x, ta32, '--m', label="2nd order approx at x=3")
plt.plot(x, ta33, ':m', label="3rd order approx at x=3")
plt.ylim((-70, 70))
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.show()
```



2.11 Plot the integrated graph of the approximations at $x = -1, x = 1, x = 3$

In [20]:

```
plt.figure(1)
plt.plot(x, f, 'dimgrey', label="function")
plt.plot(x, ta11, '-c', label="1st order approx at x=-1")
plt.plot(x, ta12, '--c', label="2nd order approx at x=-1")
plt.plot(x, ta13, ':c', label="3rd order approx at x=-1")
plt.plot(x, ta21, '-g', label="1st order approx at x=1")
plt.plot(x, ta22, '--g', label="2nd order approx at x=1")
plt.plot(x, ta23, ':g', label="3rd order approx at x=1")
plt.plot(x, ta31, '-m', label="1st order approx at x=3")
plt.plot(x, ta32, '--m', label="2nd order approx at x=3")
plt.plot(x, ta33, ':m', label="3rd order approx at x=3")
plt.ylim((-70, 70))
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.show()
```