(a)(i)

First of all, define symbol:

```
x = sympy.symbols('x', real=True)
```

and original function of $y(x) = x^4$:

```
f=x**4
```

then to use the diff function to fetch the expression for the derivative of the original function,

```
dfdx = sympy.diff(f,x)
```

print the expression of the derivative function:

```
4*x**3
```

Which means $\frac{dy}{dx}(x) = 4x^3$

(ii)

First of all, use the the derivative expression from (i):

```
x = sympy.symbols('x', real=True)
f = x ** 4
dfdx = sympy.diff(f, x)
```

then, use function 'lambdify' to allow variable 'dfdx'can be calculated with the input parameters.

```
dfdx = sympy.lambdify(x, dfdx)
```

set up three arrays to store data for plotting graph.

```
xx = []
d = []
ed = []
while x < 0.1:
    xx.append(x)
    d.append(dfdx(x))
    ed.append(((x + delta) ** 4 - x ** 4) / delta)
    x = x + 0.01
```

calculate the actual derivative values:

```
dfdx(x)
```

calculate the estimate derivative values according to the formular of finite difference:

$\frac{f(x'+\delta)-f(x')}{\delta}$

```
ed.append(((x + delta) ** 4 - x ** 4) / delta)
```

'delta' means δ, '(x + delta) ** 4' means primary function $f(x' + \delta)^4$

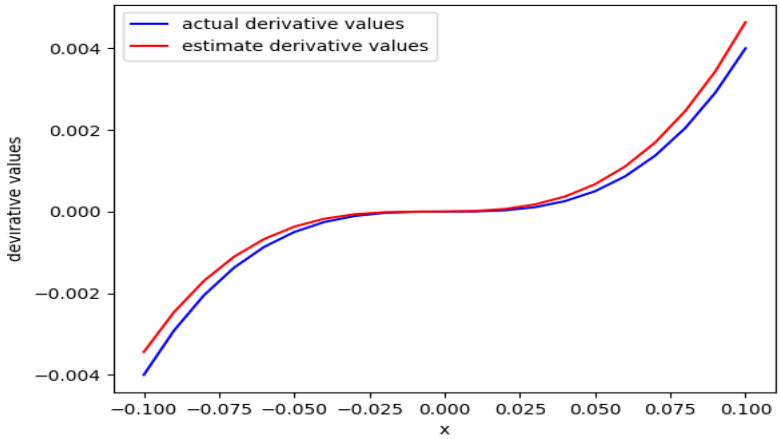

Figure 1 comparison of actual derivative values of $y(x) = x^4$ and estimates derivative values by the finite difference when δ = 0.01

as we can see from Figure 1, the curve of the estimated derivative values and the curve of actual derivative values are very close.

(iii)

Set up 4 perturbation δ:

```
delta0 = 1
delta1 = 0.1
delta2 = 0.01
delta3 = 0.001
```

use the same code in (ii) to get different estimate derivative values:

```
ed0.append(((x + delta0) ** 4 - x ** 4) / delta0)
ed1.append(((x + delta1) ** 4 - x ** 4) / delta1)
ed2.append(((x + delta2) ** 4 - x ** 4) / delta2)
ed3.append(((x + delta3) ** 4 - x ** 4) / delta3)
```

then plot the graph:

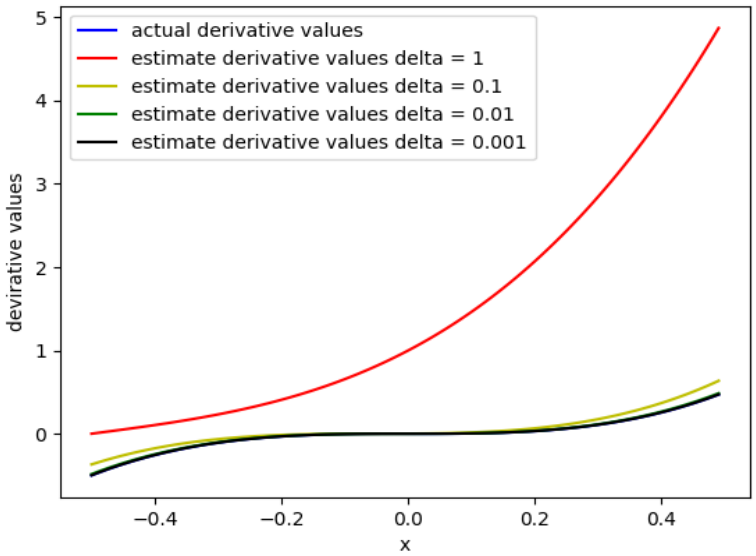

Figure 2 comparison of actual derivative values of $y(x) = x^4$ and estimate derivative values by the finite difference whit different δ(delta)

We can see from Figure2 as the perturbation δ get bigger and bigger, the estimated derivative value gets more inaccuracy. because if the perturbation δ is smaller and smaller, then the estimated point will get more close to the original point according to the finite difference function $\frac{f(x'+\delta)-f(x')}{\delta}$. The distance between the original point and the estimated point is smaller. It is obvious that the derivative value is closer, which means more accuracy.

(b)(i)

First of all use the lambdify function in sympy to define f and dfdx, so that we can get the result right away when entering the x.

```
f = sympy.lambdify(x, f)
dfdx = sympy.lambdify(x, dfdx)
```

then define functions for f and dfdx:

```
def getF(x):
    return f(x)
```

```
def getDf(x):
    return dfdx(x)
```
finally, define a function for descent:
```
def descent(startX, alpha, maxIters):
    x = startX
    i = 0
    while i < maxIters:
        step = alpha*getDf(x)
        x = x-step
        i = i + 1
```
this function can input 3 parameters, ‘startX’ is the x value of starting iterate, ‘alpha’ is the learning rate and ‘maxIters’ is the maximum iterate times.
In the while loop, first get the derivative value according to current x, then multiple learning rate to get the distance to move next, then use current x to minus this distance to get new x. The loop will stop when i>= maximum iterate times.

(ii)

Set function $y(x) = x^4$, start x=1, alpha = 0.1 and maximum iterate times = 50
```
f = x ** 4

def descent(startX=1, alpha=0.1, maxIters=50):
```
then get figure3, figure4 and figure5:

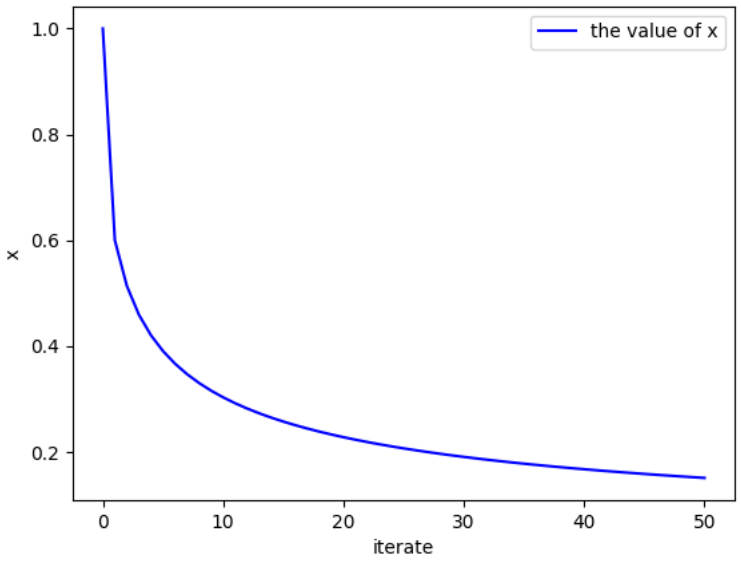

Figure 3 x varies as the number of iterations increased

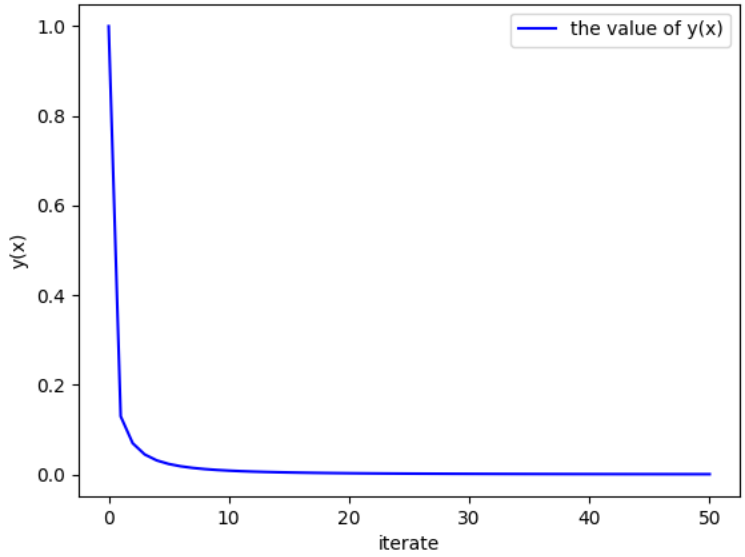

Figure 4 y(x) varies as the number of iterations increased

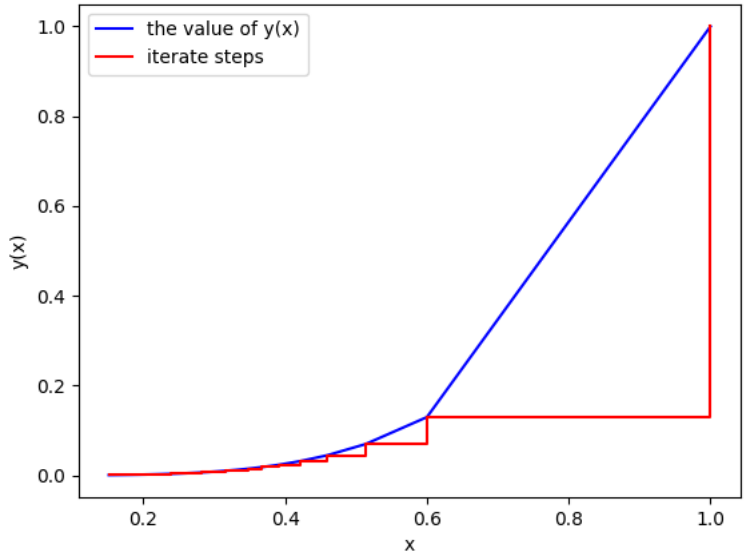

Figure 5 value of y(x) varies as

We can see from Figure3 and Figure4, both x and y(x) decreased as the number of iterations increased, then approaching 0. Because in the derivative function of $y(x) = x^4$, the derivative value always keeps positive when x>0. So in the calculation x always subtracts a small positive number, and then gradually approaches 0. We can see from figure5, the value of y(x) and x decreased very large at the beginning, then get smaller.

(iii)

Choose x=1 and α=0.2 compare to x=1 and α=0.1, use the code from (i)

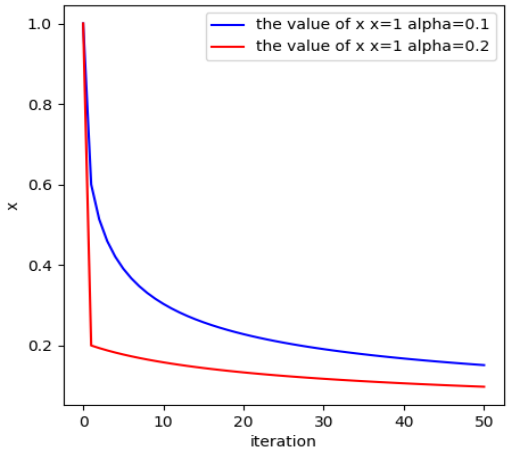

figure 6 x varies comparison as iteration increased when x=1 , alpha=0.1 and alpha=0.2

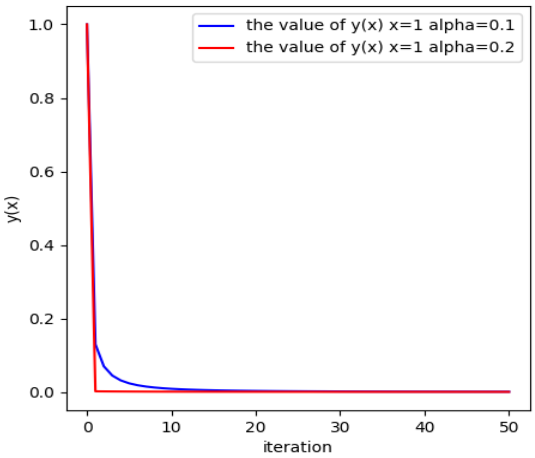

figure 7 y(x) varies comparison as iteration increased when x=1, alpha=0.1 and alpha=0.2

Choose x=1 and α=0.5, fluctuates between 1 and -1 all the time.

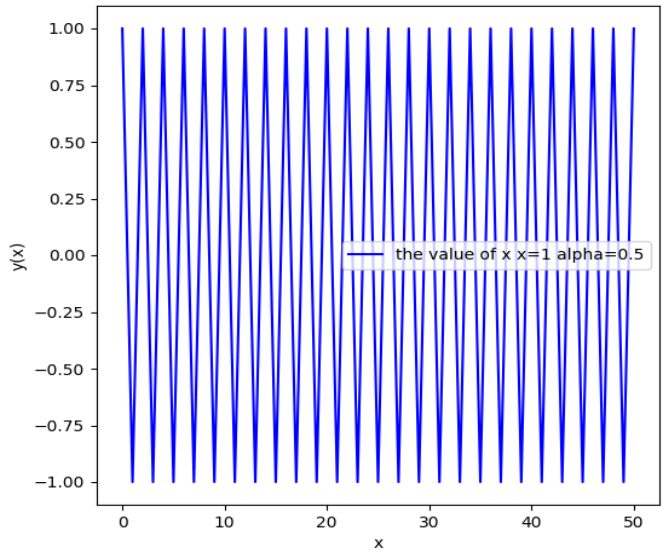

Figure 8 value of x varies as iteration increased when x=1 alpha=0.5

Choose x=2.3 and α=0.1, x=1 and α=0.6. There were cases where they could not converge use the code from (i)
Iterate 5 times get the x and y(x) as follow:

x	1	-1.4	5.19	-329.48	85839710	-1.51e+24
y(x)	1	3.84	723.10	11784329821	5.42e+31	5.31e+96

Table 1 x and y(x) values for each iteration when x=1 and α=0.6

x	2.3	-2.57	4.2	-25.3	6520.89	-1.11e+10
y(x)	27.98	43.41	310.49	415508.25	1.81e+14	1.51e+44

Table 2 x and y(x) values for each iteration when x=2.3 and α=0.1

We can see from Table1 and Table both x and y(x) get bigger and bigger and cause non-converge. Because in the code, I use alpha time derivative value to get the result that is subtracted by x. if you choose too large x or alpha, it will cause a too big result(bigger than x), then x and y(x) get bigger and bigger. As a result, it can't converge.

Choose x=2 and α=0.02, x=1 and α=0.1, x=1 and α=0.2

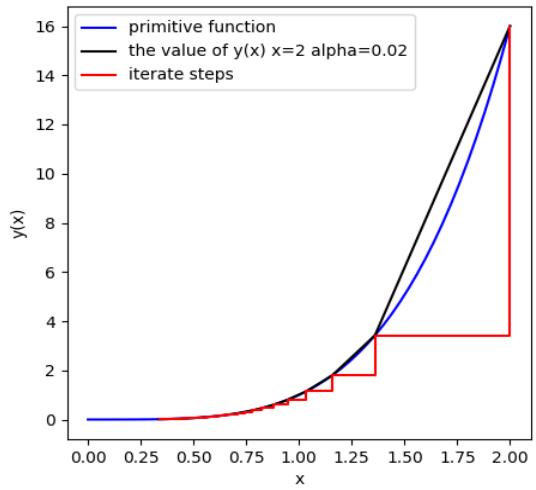

figure 9 y(x) varies as x decreased when the setting is x=2,alpha =0.02

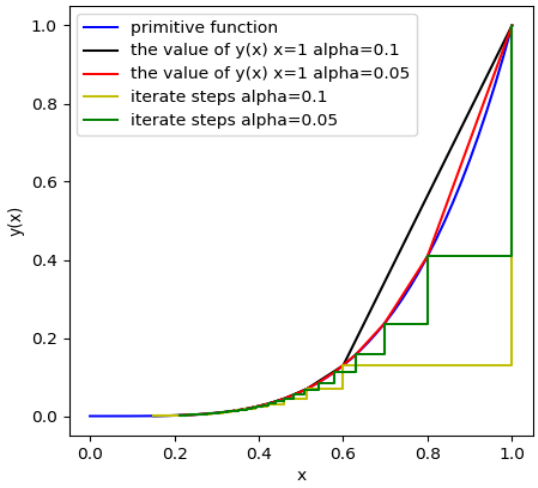

figure 10 y(x) varies as x decreased when the setting is x=1,alpha =0.1 and alpha =0.2 comparison

We can see from figure10 that as the alpha value becomes smaller, the distance of gradient descent becomes smaller, and more iterations are required, which increases the time. In addition, we can also see that as the alpha value changes The smaller the value, the closer the curve is to the curve of the original function, and the more accurate it is. As you can see from figure 9, as the value of x gets larger, we also need more iterations, which also increases the time. This is because I use the strategy of subtracting the current point derivative value by the alpha from the starting point x. The larger the x value, the more times it needs to be updated, and the smaller the alpha value, the slower the update. Sometimes the value of alpha can also cause the function to fluctuate and fail to converge, which takes longer.

(c)(i)

First set up parameters of descent function: start x=2 alpha=0.1, iteration times = 50:

```
descent1(2.0, 0.1, 50)
```

And three expressions of functions use gamma=0.5, gamma=1 and gamma=2 separately

```
f1 = 0.5 * x ** 2
f2 = x ** 2
f3 = 2 * x ** 2
```

then use the program from b(i) plot graphs as follow:

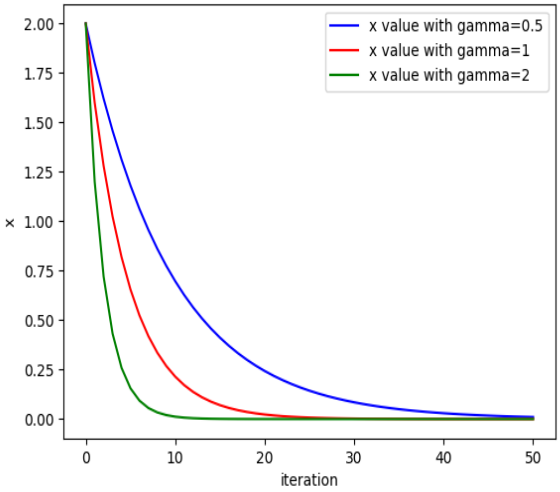

figure 11 x varies in different gamma $y(x) = \gamma x^2$

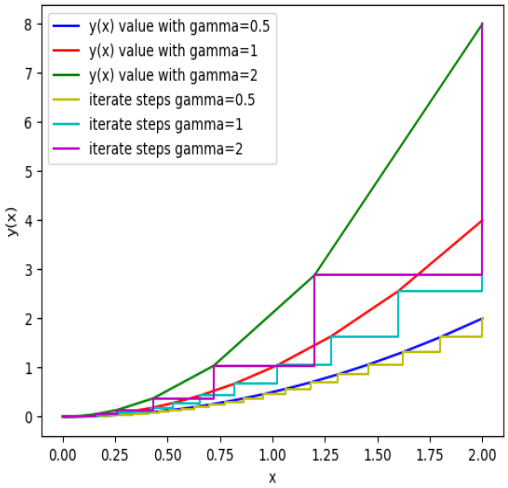

figure 12 y(x) varies as x changed with $y(x) = \gamma x^2$

We can see from figure11 the value of x decreased more sharply with a bigger gamma value. We also can see from the figure12 the iterate step gets bigger with a bigger value of gamma in function.

This is the derivative value of the current x that should be subtracted from x in the main step, and we use dfdx function in sympy to print out the derivatives of the three functions:

```
0.5*x**2 ----> 1.0*x
x**2 ----> 2*x
2*x**2 ----> 4*x
```

We can see that as the gamma value becomes larger, the coefficient of the derivative function is also larger, which leads to a larger derivative value, which causes the above situation to occur.

(ii)

First set up parameters of descent function: start x=2 alpha=0.1, iteration times = 50.
Set up three expressions of functions use gamma=0.5, gamma=1 and gamma=2 separately

```
f1 = 0.5 * abs(x)
f2 = abs(x)
f3 = 2 * abs(x)
```

then use the program from b(i) plot graphs as follow:

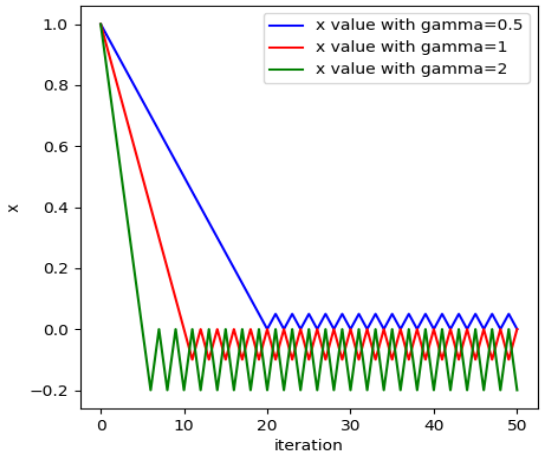

figure 13 x varies with iteration increased and y(x) = γ|x|

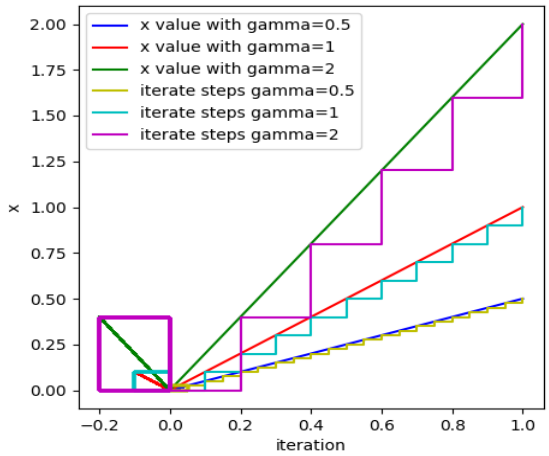

figure 14 y(x) varies as x changed with y(x) = γ|x|

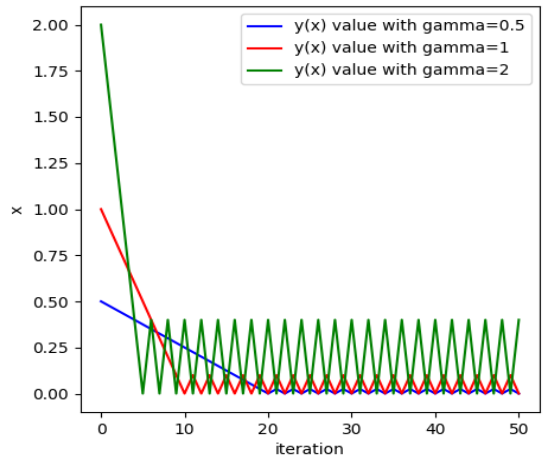

figure 15y(x) varies with iteration increased and y(x) = γ|x|

We can see from figure14 that as the gamma is larger, the span of the drop is larger, but they all keep the same constant and fluctuate all the time when they are close to 0. We can see from figure13 and figure15 that y(x) and x is the same.

This is because the derivative of a function is always a constant:

```
0.5*Abs(x) ----> 0.5*sign(x)
Abs(x) ----> sign(x)
2*Abs(x) ----> 2*sign(x)
```

Therefore, the larger he is, the greater the speed of descent, but it is always a constant, and alpha can appropriately control the size of the descent.

Appendix

Code

a_i:

```
import sympy
x = sympy.symbols('x', real=True)
f=x**4
dfdx = sympy.diff(f,x)
print(dfdx)
```

a_ii:

```
import sympy
from matplotlib import pyplot as plt

x = sympy.symbols('x', real=True)
f = x ** 4
dfdx = sympy.diff(f, x)
f = sympy.lambdify(x, f)
dfdx = sympy.lambdify(x, dfdx)
x = -0.1
delta = 0.01

xx = []
d = []
ed = []
while x < 0.1:
    xx.append(x)
    d.append(dfdx(x))
    ed.append(((x + delta) ** 4 - x ** 4) / delta)
    x = x + 0.01

plt.plot(xx, d, color='b', label='actual derivative values')
plt.plot(xx, ed, color='r', label='estimate derivative values')
plt.xlabel('x')
plt.ylabel('devirative values')
plt.legend()
plt.show()
```

a_iii:

```
import sympy
from matplotlib import pyplot as plt

x = sympy.symbols('x', real=True)
f = x ** 4
dfdx = sympy.diff(f, x)
f = sympy.lambdify(x, f)
dfdx = sympy.lambdify(x, dfdx)
x = -0.5
delta0 = 1
delta1 = 0.1
delta2 = 0.01
delta3 = 0.001

xx = []
d = []
ed0 = []
ed1 = []
ed2 = []
```

```
ed3 = []
while x < 0.5:
    xx.append(x)
    d.append(dfdx(x))
    ed0.append(((x + delta0) ** 4 - x ** 4) / delta0)
    ed1.append(((x + delta1) ** 4 - x ** 4) / delta1)
    ed2.append(((x + delta2) ** 4 - x ** 4) / delta2)
    ed3.append(((x + delta3) ** 4 - x ** 4) / delta3)
    x = x + 0.01

plt.plot(xx, d, color='b', label='actual derivative values')
plt.plot(xx, ed0, color='r', label='estimate derivative values delta = 1')
plt.plot(xx, ed1, color='y', label='estimate derivative values delta = 0.1')
plt.plot(xx, ed2, color='g', label='estimate derivative values delta = 0.01')
plt.plot(xx, ed3, color='black', label='estimate derivative values delta = 0.001')
plt.xlabel('x')
plt.ylabel('devirative values')
plt.legend()
plt.show()
```

b_i and b_ii:

```
import math

import sympy
from matplotlib import pyplot as plt

x = sympy.symbols('x', real=True)
f = x ** 4
dfdx = sympy.diff(f, x)
f = sympy.lambdify(x, f)
dfdx = sympy.lambdify(x, dfdx)
ii = []
ff = []
xx = []

def getF(x):
    print(f(x))
    return f(x)

def getDf(x):
    return dfdx(x)


def descent(startX=1, alpha=0.1, maxIters=50):
    x = startX
    i = 0
    ii.append(i)
    xx.append(x)
    ff.append(getF(x))
    while i < maxIters:
        step = alpha * getDf(x)
        x = x - step
        i = i + 1
        ii.append(i)
        xx.append(x)
        ff.append(getF(x))
        # ff.append(math.log10(getF(x)))


descent(1.0, 0.1, 50)
plt.figure(figsize=(5, 5))
plt.plot(xx, ff, color='b', label='the value of y(x)')
plt.step(xx, ff, color='r', label='')
```

```
plt.xlabel('x')
plt.ylabel('y(x)')
plt.legend()
plt.show()
```

b_iii:

```
import math

import sympy
from matplotlib import pyplot as plt

x = sympy.symbols('x', real=True)
f = x ** 4
dfdx = sympy.diff(f, x)
f = sympy.lambdify(x, f)
dfdx = sympy.lambdify(x, dfdx)
ii = []
ff = []
xx = []
ff0 = []
xx0 = []
ii1 = []
ff1 = []
xx1 = []
ii2 = []
ff2 = []
xx2 = []


def getF(x):
    # print(f(x))
    return f(x)


def getDf(x):
    return dfdx(x)


def original(maxIters):
    s = 1
    x = s;
    xx0.append(x)
    ff0.append(getF(x))
    i = 0
    while i < maxIters:
        x = x - (s / 50)
        xx0.append(x)
        ff0.append(getF(x))
        i = i + 1


def descent2(startX, alpha, maxIters):
    x = startX
    i = 0
    ii2.append(i)
    xx2.append(x)
    ff2.append(getF(x))
    while i < maxIters:
        step = alpha * getDf(x)
        x = x - step
        i = i + 1
        ii2.append(i)
        xx2.append(x)
        ff2.append(getF(x))


```

```
def descent1(startX, alpha, maxIters):
    x = startX
    i = 0
    ii1.append(i)
    xx1.append(x)
    ff1.append(getF(x))
    while i < maxIters:
        step = alpha * getDf(x)
        x = x - step
        i = i + 1
        ii1.append(i)
        xx1.append(x)
        ff1.append(getF(x))


def descent(startX, alpha, maxIters):
    x = startX
    i = 0
    ii.append(i)
    xx.append(x)
    ff.append(getF(x))
    while i < maxIters:
        step = alpha * getDf(x)
        x = x - step

        i = i + 1
        ii.append(i)
        xx.append(x)
        ff.append(getF(x))
        # ff.append(math.log10(getF(x)))


descent(1.0, 0.1, 50)
descent1(2.0, 0.02, 50)
descent2(1.0, 0.05, 50)
original(50)
plt.figure(figsize=(5, 5))
plt.plot(xx0, ff0, color='b', label='primitive function')
plt.plot(xx, ff, color='black', label='the value of y(x) x=1 alpha=0.1')
plt.plot(xx2, ff2, color='r', label='the value of y(x) x=1 alpha=0.05')
plt.step(xx, ff, color='y', label='iterate steps alpha=0.1')
plt.step(xx2, ff2, color='g', label='iterate steps alpha=0.05')
plt.xlabel('x')
plt.ylabel('y(x)')
plt.legend()
plt.show()
```

c_i:

```
import math

import sympy
from matplotlib import pyplot as plt

x = sympy.symbols('x', real=True)
f1 = 0.5 * x ** 2
f2 = x ** 2
f3 = 2 * x ** 2
dfdx1 = sympy.diff(f1, x)
dfdx2 = sympy.diff(f2, x)
dfdx3 = sympy.diff(f3, x)
print(f1,"---->",dfdx1)
print(f2,"---->",dfdx2)
print(f3,"---->",dfdx3)

f1 = sympy.lambdify(x, f1)
f2 = sympy.lambdify(x, f2)
```

```
f3 = sympy.lambdify(x, f3)
dfdx1 = sympy.lambdify(x, dfdx1)
dfdx2 = sympy.lambdify(x, dfdx2)
dfdx3 = sympy.lambdify(x, dfdx3)
ii1 = []
ff1 = []
xx1 = []
ii2 = []
ff2 = []
xx2 = []
ii3 = []
ff3 = []
xx3 = []

def getF1(x):
    return f1(x)

def getF2(x):
    return f2(x)

def getF3(x):
    return f3(x)


def getDf1(x):
    return dfdx1(x)

def getDf2(x):
    return dfdx2(x)

def getDf3(x):
    return dfdx3(x)


def descent1(startX, alpha, maxIters):
    x = startX
    i = 0
    ii1.append(i)
    xx1.append(x)
    ff1.append(getF1(x))
    while i < maxIters:
        step = alpha * getDf1(x)
        x = x - step
        i = i + 1
        ii1.append(i)
        xx1.append(x)
        ff1.append(getF1(x))

def descent2(startX, alpha, maxIters):
    x = startX
    i = 0
    ii2.append(i)
    xx2.append(x)
    ff2.append(getF2(x))
    while i < maxIters:
        step = alpha * getDf2(x)
        x = x - step
        i = i + 1
        ii2.append(i)
        xx2.append(x)
        ff2.append(getF2(x))

def descent3(startX, alpha, maxIters):
    x = startX
    i = 0
    ii3.append(i)
```

```
    xx3.append(x)
    ff3.append(getF3(x))
    while i < maxIters:
        step = alpha * getDf3(x)
        x = x - step
        i = i + 1
        ii3.append(i)
        xx3.append(x)
        ff3.append(getF3(x))


descent1(2.0, 0.1, 50)
descent2(2.0, 0.1, 50)
descent3(2.0, 0.1, 50)
plt.plot(xx1, ff1, color='b', label='y(x) value with gamma=0.5')
plt.plot(xx2, ff2, color='r', label='y(x) value with gamma=1')
plt.plot(xx3, ff3, color='g', label='y(x) value with gamma=2')
plt.step(xx1, ff1, color='y', label='iterate steps gamma=0.5')
plt.step(xx2, ff2, color='c', label='iterate steps gamma=1')
plt.step(xx3, ff3, color='m', label='iterate steps gamma=2')
plt.xlabel('x')
plt.ylabel('y(x)')
plt.legend()
plt.show()
```

c_ii:

```
import math

import sympy
from matplotlib import pyplot as plt

x = sympy.symbols('x', real=True)
f1 = 0.5 * abs(x)
f2 = abs(x)
f3 = 2 * abs(x)
dfdx1 = sympy.diff(f1, x)
dfdx2 = sympy.diff(f2, x)
dfdx3 = sympy.diff(f3, x)
print(f1,"---->",dfdx1)
print(f2,"---->",dfdx2)
print(f3,"---->",dfdx3)

f1 = sympy.lambdify(x, f1)
f2 = sympy.lambdify(x, f2)
f3 = sympy.lambdify(x, f3)
dfdx1 = sympy.lambdify(x, dfdx1)
dfdx2 = sympy.lambdify(x, dfdx2)
dfdx3 = sympy.lambdify(x, dfdx3)
print((dfdx1(1)))
ii1 = []
ff1 = []
xx1 = []
ii2 = []
ff2 = []
xx2 = []
ii3 = []
ff3 = []
xx3 = []

def getF1(x):
    return f1(x)

def getF2(x):
    return f2(x)

def getF3(x):
```

```
    return f3(x)


def getDf1(x):
    return dfdx1(x)

def getDf2(x):
    return dfdx2(x)

def getDf3(x):
    return dfdx3(x)


def descent1(startX, alpha, maxIters):
    x = startX
    i = 0
    ii1.append(i)
    xx1.append(x)
    ff1.append(getF1(x))
    while i < maxIters:
        step = alpha * getDf1(x)
        x = x - step
        i = i + 1
        ii1.append(i)
        xx1.append(x)
        ff1.append(getF1(x))

def descent2(startX, alpha, maxIters):
    x = startX
    i = 0
    ii2.append(i)
    xx2.append(x)
    ff2.append(getF2(x))
    while i < maxIters:
        step = alpha * getDf2(x)
        x = x - step
        i = i + 1
        ii2.append(i)
        xx2.append(x)
        ff2.append(getF2(x))

def descent3(startX, alpha, maxIters):
    x = startX
    i = 0
    ii3.append(i)
    xx3.append(x)
    ff3.append(getF3(x))
    while i < maxIters:
        step = alpha * getDf3(x)
        x = x - step
        i = i + 1
        ii3.append(i)
        xx3.append(x)
        ff3.append(getF3(x))


descent1(1.0, 0.1, 50)
descent2(1.0, 0.1, 50)
descent3(1.0, 0.1, 50)
plt.figure(figsize=(5, 5))
plt.plot(ii1, ff1, color='b', label='y(x) value with gamma=0.5')
plt.plot(ii2, ff2, color='r', label='y(x) value with gamma=1')
plt.plot(ii3, ff3, color='g', label='y(x) value with gamma=2')
# plt.step(xx1, ff1, color='y', label='iterate steps gamma=0.5')
# plt.step(xx2, ff2, color='c', label='iterate steps gamma=1')
# plt.step(xx3, ff3, color='m', label='iterate steps gamma=2')
```

```
107.plt.xlabel('iteration')
108.plt.ylabel('x')
109.plt.legend()
110.plt.show()
```