MNUM Exam 2017

Telmo Baptista

January 10, 2020

1 Exercise 1

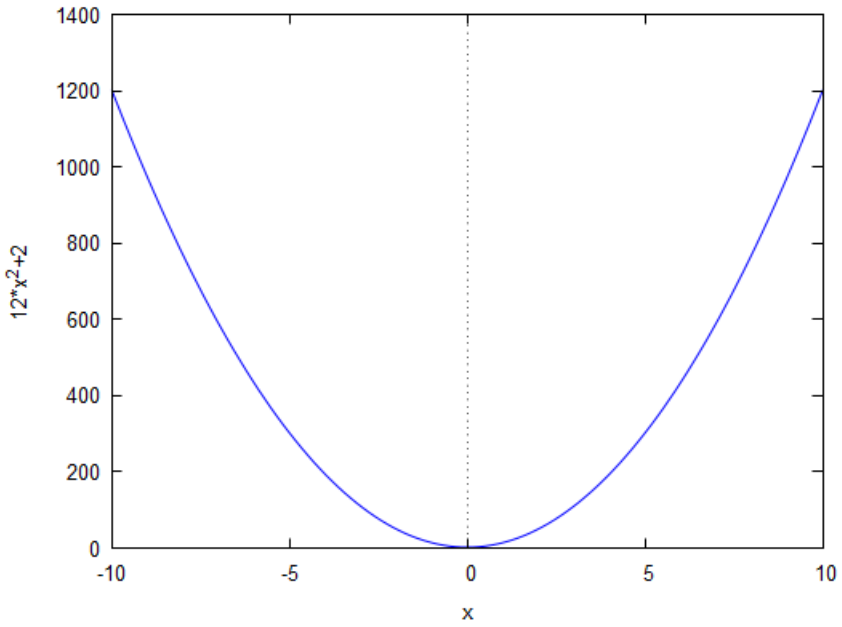
To minimize the function $f(x, a) = (x - a)^2 + x^4$ we can first apply a searching method to search the interval where the minimum is located.
If we study the derivatives of the function, we obtain:

$$\frac{df}{dx} = 4x^3 + 2(x - a) \tag{1}$$

and

$$\frac{d^2 f}{dx^2} = 12x^2 + 2 \tag{2}$$

If we graph the second derivative



We can note that the sign of the second derivative never changes, it's always positive, therefore the concavity of the function is always faced up. That eases the process of searching the interval that contains the minimum of the function, as starting from one point there's only two possibilities, either the function decreases to the right side or decreases to the left side.
By applying an unidimensional search:

```
from math import cos, sin, sqrt

# a = 4
f = lambda x: (x-4)**2 + x**4

# guess
a = 4
fa = f(a)
h = 1 # step

b = a + h
fb = f(b)

if fa < fb:
    h = -h
    b = a + h
    fb = f(b)

c = b + h
fc = f(c)

while (fb > fc):
    a = b
```

```
    fa = fb
    b = c
    fb = c
    c = b + h
    fc = f(c)

if (h < 0):
    a, b = c, a
else:
    b = c
# the extremes of the interval are now stored on a and b
print(f"Interval: [ {a} , {b} ]")
```

Program Output:

```
Interval:  [ 0 , 2 ]
```

After doing a search on the interval we need to work on, we can apply now a reducing method such as the trisection method or the golden rule method.

We'll be applying the golden rule method as this saves more calculations than the trisection method, and we'll stop the method when the length of the interval is less than 10^{-5} to get the 5 digits precision.

The Golden Rule method consists on dividing the interval into specific proportions (proportions that follow the Golden Ratio: $\frac{\sqrt{5}-1}{2}$). Having the interval divided into three equally sized intervals, $[a, c]$, $[c, d]$, $[d, b]$, we can now compare the values of the function in the points c and d to decide which part of the interval will be cut. If $f(c)$ value is less than $f(d)$ then we can cut the interval $[d, b]$ as we can assure the minimum isn't there. On the other hand, if $f(d)$ value is less than $f(c)$ then we can cut the interval $[a, c]$.

```
from math import cos, sin, sqrt

# a = 4
f = lambda x: (x-4)**2 + x**4

# guess
a = -10
fa = f(a)
h = 1 # step

b = a + h
fb = f(b)

if fa < fb:
    h = -h
    b = a + h
    fb = f(b)

c = b + h
fc = f(c)

while (fb > fc):
    a = b
    fa = fb
    b = c
    fb = fc
    c = b + h
    fc = f(c)

if (h < 0):
    a, b = c, a
else:
    b = c
# the extremes of the interval are now stored on a and b
print(f"Interval: [ {a} , {b} ]")

g_ratio = (sqrt(5)-1)/2
```

```
def golden_rule(f, lower, upper):
    a = lower
    b = upper
    c = b - g_ratio * (b-a)
    d = a + g_ratio * (b-a)

    fc = f(c)
    fd = f(d)

    while (abs(b-a) > 1e-5):
        if (fc < fd):
            b = d
            d = c
            fd = fc
            c = b - g_ratio * (b-a)
            fc = f(c)

        else:
            a = c
            c = d
            fc = fd
            d = a + g_ratio * (b-a)
            fd = f(d)

    return a, b

a, b = golden_rule(f, a, b)

print(f"Interval: [ {a:.7f} , {b:.7f} ]")
```

Program Output:

```
Interval:  [ 1.1281714 , 1.1281788 ]
```

As we applied a reducing method we only obtain an interval where the minimum is located, no the value of the minimum, but as we can specify the precision we want we say the minimum has an approximate value of 1.12817.

2 Exercise 2

```
from math import cos, sin, sqrt, exp

def trapeze(f, a, b, h):
    n = int(abs(a-b)/h)

    h1 = h/2
    h2 = h1/2

    S = h/2 * (f(a) + f(b) + sum(2*f(a + i * h) for i in range(1, n)))

    S1 = h1/2 * (f(a) + f(b) + sum(2*f(a + i * h1) for i in range(1, 2*n)))

    S2 = h2/2 * (f(a) + f(b) + sum(2*f(a + i * h2) for i in range(1, 4*n)))

    QC = (S1-S)/(S2-S1)

    error = (S2-S1)/3


    return S, h, S1, h1, S2, h2, QC, error


def simpson(f, a, b, h):
    n = int(abs(a-b)/h)

    h1 = h/2
    h2 = h1/2

    S = h/3 * (f(a) + f(b) + sum(4*f(a + i * h) if i%2 else 2*f(a + i*h) for i in range(1, n)))

    S1 = h1/3 * (f(a) + f(b) + sum(4*f(a + i * h1) if i%2 else 2*f(a + i*h1) for i in range(1, 2*n)))

    S2 = h2/3 * (f(a) + f(b) + sum(4*f(a + i * h2) if i%2 else 2*f(a + i*h2) for i in range(1, 4*n)))

    QC = (S1-S)/(S2-S1)

    error = (S2-S1)/15


    return S, h, S1, h1, S2, h2, QC, error


f = lambda x: sqrt(1+(2.5*exp(2.5*x))**2)

a = 0
b = 1
print("TRAPEZE")
S, h, S1, h1, S2, h2, QC, error = trapeze(f, a, b, 0.125)
print(f"""
h     : {h:.5f}
h'    : {h1:.5f}
h''   : {h2:.5f}
L1    : {S:.5f}
L2    : {S1:.5f}
L3    : {S2:.5f}
QC    : {QC:.5f}
error: {error:.5f}""")
```

```
print("\nSIMPSON")
S, h, S1, h1, S2, h2, QC, error = simpson(f, a, b, 0.125)
print(f"""
h     : {h:.5f}
h'    : {h1:.5f}
h''   : {h2:.5f}
L1    : {S:.5f}
L2    : {S1:.5f}
L3    : {S2:.5f}
QC    : {QC:.5f}
error: {error:.5f}""")
```

Program Output:

$TRAPEZE$

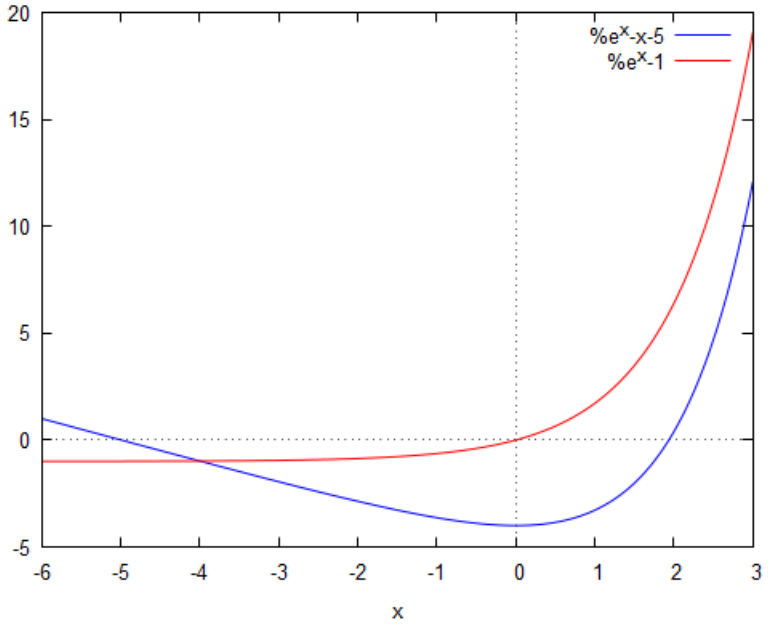
h : 0.12500
h' : 0.06250
h'' : 0.03125
$L1$: 11.34629
$L2$: 11.27778
$L3$: 11.26063
QC : 3.99394
$error$: -0.00572

$SIMPSON$

h : 0.12500
h' : 0.06250
h'' : 0.03125
$L1$: 11.25550
$L2$: 11.25495
$L3$: 11.25491
QC : 15.85798
$error$: -0.00000

3 Exercise 3

We'll use maxima to draw the graph of the function $f(x) = e^x - x - 5$ and its derivative $f'(x) = e^x - 1$.
Function and its first derivative graph



With this we can isolate the roots of the function, let the interval for the first root be $[-6, -4]$ we can prove there's atleast one zero in this interval by comparing the signs of the function in the extreme points of the interval.

$f(-6) \approx 1.0024787521766667$
$f(-4) \approx -0.9816843611112658$

As $f(-6)$ and $f(-4)$ have opposite signs, it's proven that there's atleast one zero on the interval. If we also analyse the derivative on this interval we can observe the sign of the derivative stays the same for the whole interval, which means the function is monotone on the interval and the root will be unique. Similar structure can be applied to the second root observed in the graph, by starting with the interval $[1, 3]$.

$f(1) \approx -3.281718171540955$
$f(3) \approx 12.085536923187668$

And for the same reason as the previous interval. As $f(1)$ and $f(3)$ have opposite signs, it's proven that there's atleast one zero on the interval. If we also analyse the derivative on this interval we can observe the sign of the derivative stays the same for the whole interval, which means the function is monotone on the interval and the root will be unique.
With this the roots of the function are isolated, one resides on the interval $[-6, -4]$ and the other root resides $[1, 3]$.
To test the convergence of the recurrent expressions we must study the first derivative of those expressions. Using Maxima to calculate and graph the expressions on those intervals isolated, we obtain:

```
(%i35)exp1:%e^x-5;
```

(%o35) $e^x - 5$

```
(%i36)exp2:log(x+5);
```

(%o36) $\log(x + 5)$

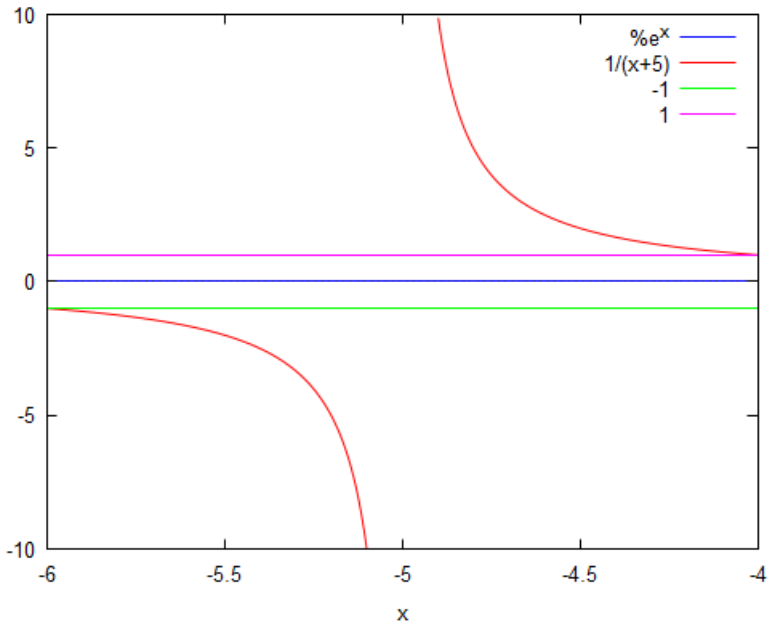
```
(%i37)de1:diff(exp1,x);
```

(%o37) e^x

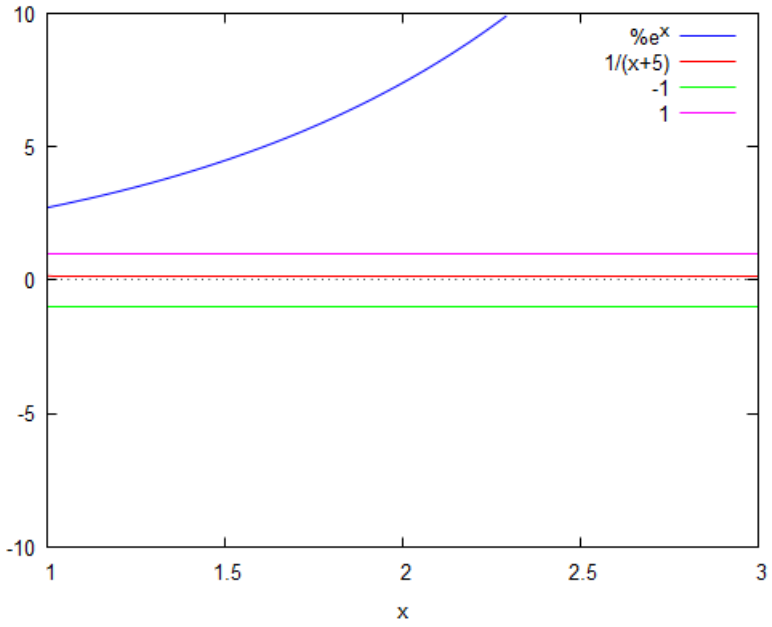
```
(%i38)de2:diff(exp2,x);
```

(%o38) $$\frac{1}{x+5}$$

```
(%i40)wxplot2d([de1,de2,-1,1],[x,-6,-4],[y,-10,10]);
```




```
(%i41)wxplot2d([de1,de2,-1,1],[x,1,3],[y,-10,10]);
```



As we can observe, in the interval $[-6,-4]$ the derivative of the second expression $log(x+5)$, $\frac{1}{x+5}$ isn't contained between the interval $f(x)=-1$ and $f(x)=1$, and so we can't conclude if the expression converges on this interval. On the other hand the derivative of the expression e^x-5, e^x is contained in the interval cited above, and so we can say the expression converges in this interval.
As for the second interval $[1,3]$, it's the opposite, the derivative of the expression $log(x+5)$ is contained on the interval $f(x)=-1$ and $f(x)=1$, and so we can conclude it converges, but the derivative of the expression e^x-5 isn't contained and so we can't conclude if it converges.
Another non-interval method that can be applied is Newton's Method which is a special case of a Picardo-Peano method, in which the recurrent expression $g(x)=x-\frac{f(x)}{f'(x)}$.
Calculating the expression for Newton's Method:

```
(%i54)f:%e^x-x-5;
```

(%o54) $e^x - x - 5$

```
(%i55)df:diff(f,x);
```

(%o55) $e^x - 1$

```
(%i56)g:x-f/df;
```

(%o56) $x - \frac{e^x - x - 5}{e^x - 1}$

```
(%i57)ratsimp(%);
```

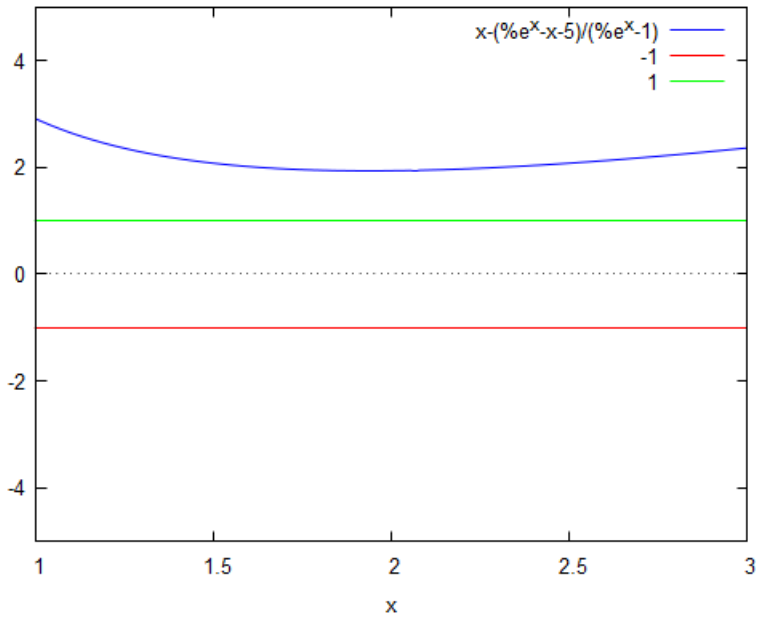
(%o57) $\frac{(x-1)\,e^x + 5}{e^x - 1}$

Studying the convergence of this formula for the interval $[1, 3]$ which contains the highest of the roots.

```
(%i66)diff(g,x);
```

(%o66) $\frac{e^x\,(e^x - x - 5)}{(e^x - 1)^2}$

```
(%i67)wxplot2d([g,-1,1],[x,1,3],[y,-5,5]);
```



The derivative of the recurrent expression isn't contained on the interval $f(x) = -1$ and $f(x) = 1$, so we can't conclude the convergence of the expression.
Now to applying the method for the three expressions with initial conditions: $x_0 = 2$ and the stopping criteria being the difference between successive approximations be less than 10^{-5} ($|x_n - x_{n+1}| \leqslant 10^{-5}$).

```
from math import cos, sin, sqrt, exp, log

f = lambda x: exp(x) - x - 5

exp1 = lambda x: exp(x) - 5

exp2 = lambda x: log(5 + x)

print(f"f(-6) = {f(-6)}")
print(f"f(-4) = {f(-4)}")

print(f"f(1) = {f(1)}")
```

```
print(f"f(3) = {f(3)}")

def find_root(g, x0):
    x = g(x0)
    print(f"Iter 0: X= {x0}")
    i = 1
    while (abs(x-x0) > 1e-5):
        print(f"Iter {i}: X= {x}")
        x0 = x
        x = g(x)
        i += 1

    return x

newton_exp = lambda x: ((x-1)*exp(x)+5)/(exp(x)-1)
try:
    print("EXPRESSION 1")
    root = find_root(exp1, 2)
    print("Root: X =", root)
    print("Value: f(X) =", f(root))
    print()
except:
    print("Diverged\n")

try:
    print("EXPRESSION 2")
    root = find_root(exp2, 2)
    print("Root: X =", root)
    print("Value: f(X) =", f(root))
    print()
except:
    print("Diverged\n")

try:
    print("EXPRESSION NEWTON")
    root = find_root(newton_exp, 2)
    print("Root: X =", root)
    print("Value: f(X) =", f(root))
except:
    print("Diverged\n")
```

Program Output:

$f(-6) = 1.0024787521766667$
$f(-4) = -0.9816843611112658$
$f(1) = -3.281718171540955$
$f(3) = 12.085536923187668$
$EXPRESSION1$
$Iter0 : X = 2$
$Iter1 : X = 2.3890560989306504$
$Iter2 : X = 5.903197544594013$
$Iter3 : X = 361.2065595626377$
$Iter4 : X = 7.41336947232664e + 156$
$Diverged$

$EXPRESSION2$
$Iter0 : X = 2$
$Iter1 : X = 1.9459101490553132$
$Iter2 : X = 1.9381530185997644$
$Iter3 : X = 1.937035603449888$
$Iter4 : X = 1.9368745367887088$
$Iter5 : X = 1.9368513181488938$
$Root : X = 1.9368479710105608$
$Value : f(X) = 3.3471383327210447e - 06$

$EXPRESSIONNEWTON$
$Iter0 : X = 2$
$Iter1 : X = 1.939105856497994$
$Iter2 : X = 1.9368503837145918$
$Root : X = 1.9368474072253945$
$Value : f(X) = 3.072742060794553e - 11$

By comparing these results we can conclude:

- The first expression diverged, and it could be expected to do so, by the conclusions we got when studying the convergence.
- The second expression converged, as expected. It took 5 iterations to converge to a precision of 10^{-5}, and the residue is approximately 3.34714×10^{-6}.
- The newton expression also converged, even if we couldn't conclude anything by studying the convergence, and took less iterations to converge to the same precision and the residue is less than the previous expression, approximately 3.07274×10^{-11}. (approximately 100000 lower than the one obtained by the second expression).

With this we can conclude the newton expression performed better than the other expressions to find the root of this interval.

4 Exercise 4

```
from math import cos, sin, sqrt, exp, log

dC = lambda t, C, T: -exp(-0.5/(T+273))*C

dT = lambda t, C, T: 30 * exp(-0.5/(T+273))*C - 0.5*(T-20)


def euler(f, t, C, T, h):
    return h * f(t, C, T)


def rk4(fC, fT, t, C, T, h):
    C1 = h * fC(t, C, T)
    T1 = h * fT(t, C, T)

    C2 = h * fC(t + h/2, C + C1/2, T + T1/2)
    T2 = h * fT(t + h/2, C + C1/2, T + T1/2)

    C3 = h * fC(t + h/2, C + C2/2, T + T2/2)
    T3 = h * fT(t + h/2, C + C2/2, T + T2/2)

    C4 = h * fC(t + h, C + C3, T + T3)
    T4 = h * fT(t + h, C + C3, T + T3)

    return C1/3 + C2/6 + C3/6 + C4/3, T1/3 + T2/6 + T3/6 + T4/3

t0 = 0
C0 = 2.5
T0 = 25.0
h = 0.25

print("EULER")
t = t0
C = C0
T = T0

print(f"Iteration 0: t = {t:.5f} \t| C = {C:.5f} \t| T = {T:.5f}")

for i in range(2):

    hC = euler(dC, t, C, T, h)
    hT = euler(dT, t, C, T, h)

    C += hC
    T += hT
    t += h

    print(f"Iteration {i}: t = {t:.5f} \t| C = {C:.5f} \t| T = {T:.5f}")

print()
Th = T # saving for later exercise

print("RK4")
t = t0
C = C0
T = T0

print(f"Iteration 0: t = {t:.5f} \t| C = {C:.5f} \t| T = {T:.5f}")
```

```
for i in range(2):

    hC, hT = rk4(dC, dT, t, C, T, h)

    C += hC
    T += hT
    t += h

    print(f"Iteration {i}: t = {t:.5f} \t| C = {C:.5f} \t| T = {T:.5f}")

print()

h1 = h/2
t = t0
C = C0
T = T0


for i in range(4):

    hC = euler(dC, t, C, T, h1)
    hT = euler(dT, t, C, T, h1)

    C += hC
    T += hT
    t += h1

Th1 = T

h2 = h1/2
t = t0
C = C0
T = T0


for i in range(8):

    hC = euler(dC, t, C, T, h2)
    hT = euler(dT, t, C, T, h2)

    C += hC
    T += hT
    t += h2

Th2 = T

QC = (Th1 - Th)/(Th2-Th1)

error = (Th2-Th1)

print("h' = {h1}")
print(f"T' = {Th1}")
print("h'' = {h2}")
print(f"T'' = {Th2}")
print(f"QC = {QC}")
print(f"Error = {error}")
```

Program Output:

$EULER$
$Iteration0 : t = 0.00000|C = 2.50000|T = 25.00000$
$Iteration1 : t = 0.25000|C = 1.87605|T = 43.09357$
$Iteration2 : t = 0.50000|C = 1.40778|T = 54.25499$

$RK4$
$Iteration0 : t = 0.00000|C = 2.50000|T = 25.00000$
$Iteration1 : t = 0.25000|C = 1.94660|T = 40.00542$
$Iteration2 : t = 0.50000|C = 1.51568|T = 49.79557$

$h' = 0.12500$
$T' = 51.77067$
$h'' = 0.06250$
$T'' = 50.69205$
$QC = 2.30325$
$Error = -1.07861$

5 Exercise 5

Calculating the expressions that compose the gradient using Maxima:

```
(%i71)f:(-1.1*x*y)+12*y+7*x^2-8*x;
```

(%o71) $$-1.1\,x\,y+12\,y+7\,x^2-8\,x$$

```
(%i72)diff(f,x);
```

(%o72) $$-1.1\,y+14\,x-8$$

```
(%i73)diff(f,y);
```

(%o73) $$12-1.1\,x$$

```
from math import cos, sin, sqrt, exp, log

w = lambda x, y: -1.1*x*y + 12*y + 7*x**2 - 8*x

x0 = 3
y0 = 1
k  = 0.1

dfx = lambda x, y: -1.1*y + 14*x - 8
dfy = lambda x, y: 12 - 1.1*x

x = x0
y = y0
for i in range(1):

    x, y = x - k * dfx(x, y), y - k * dfy(x, y)

print(f"X = {x:.5f} \t Y = {y:.5f}")
print(f"w(X, Y) = {w(x, y):.5f}")
```

Program Output:

```
X = -0.29000 Y = 0.13000
w(X, Y) = 4.51017
```