ΕΛΛΗΝΙΚΟ
ΑΝΟΙΚΤΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ

DAMA 50

Written Assignment IV

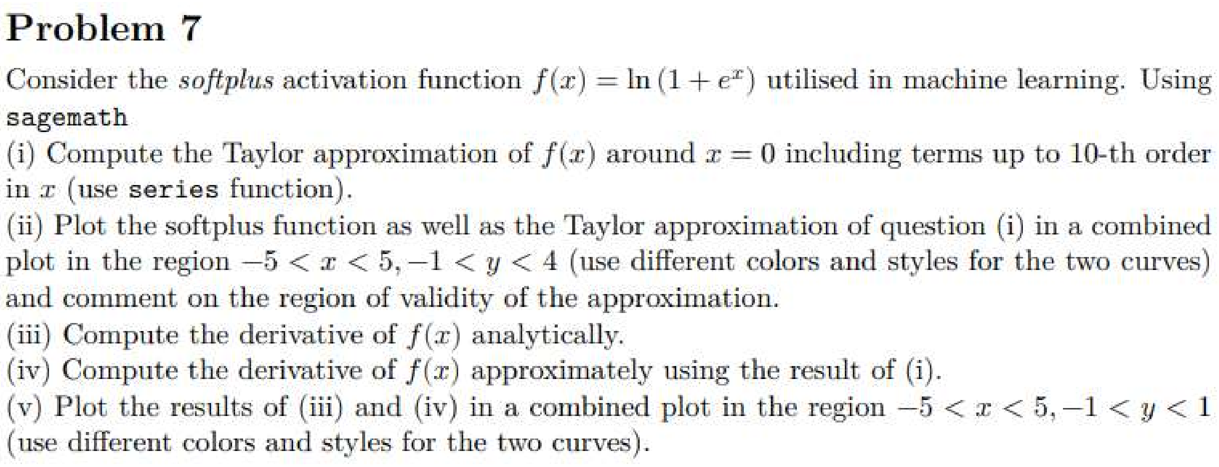
Submitted by:

Panagiotis Paltsokas

ID: std163861

[DAMA 50] Written Assignment 4

Panagiotis Paltsokas - std163861



Problem 7

Consider the *softplus* activation function $f(x) = \ln(1 + e^x)$ utilised in machine learning. Using `sagemath`
(i) Compute the Taylor approximation of $f(x)$ around $x = 0$ including terms up to 10-th order in x (use `series` function).
(ii) Plot the softplus function as well as the Taylor approximation of question (i) in a combined plot in the region $-5 < x < 5, -1 < y < 4$ (use different colors and styles for the two curves) and comment on the region of validity of the approximation.
(iii) Compute the derivative of $f(x)$ analytically.
(iv) Compute the derivative of $f(x)$ approximately using the result of (i).
(v) Plot the results of (iii) and (iv) in a combined plot in the region $-5 < x < 5, -1 < y < 1$ (use different colors and styles for the two curves).

```
In [1]: restore()
```

```
In [2]: %display latex
```

Define the variable x

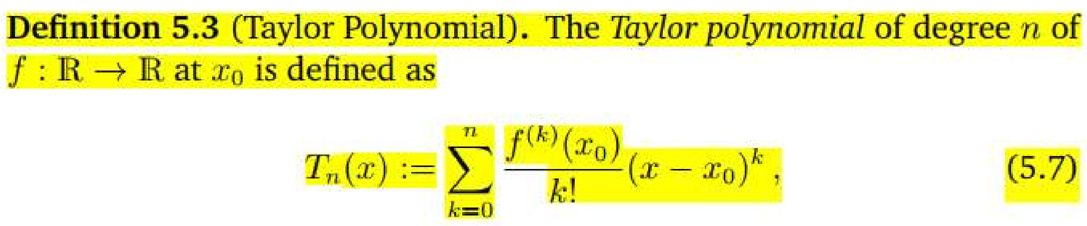
```
In [3]: var('x')
```

Out[3]: x

Define the Softplus Function

```
In [4]: f=ln(1+e^x);f(x)
```

Out[4]: $\log(e^x + 1)$



Definition 5.3 (Taylor Polynomial). The *Taylor polynomial* of degree n of $f : \mathbb{R} \to \mathbb{R}$ at x_0 is defined as

$$T_n(x) := \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k , \qquad (5.7)$$

(i)

Compute the Taylor approximation of the softplus function, around $x = 0$ including terms up to 10-th order in x

```
In [5]: taylor_series = f.series(x==0, 12);taylor_series
```

Out[5]:

$$(\log(2)) + \frac{1}{2}x + \frac{1}{8}x^2 + (-\frac{1}{192})x^4 + \frac{1}{2880}x^6 + (-\frac{17}{645120})x^8 + \frac{31}{14515200}x^{10} + \mathcal{O}\left(x^{12}\right)$$

Truncate to limit the series to a certain number of terms

```
In [6]: ft=taylor_series.truncate();ft
```

Out[6]:

$$\frac{31}{14515200}\,x^{10} - \frac{17}{645120}\,x^8 + \frac{1}{2880}\,x^6 - \frac{1}{192}\,x^4 + \frac{1}{8}\,x^2 + \frac{1}{2}\,x + \log(2)$$

(ii)

```
In [7]: from sage.plot.plot import plot
```

Plot the two functions seperately

```
In [8]: softplus_graph = plot(f, (x, -5, 5), color='darkblue', legend_label='Softplus')
        taylor_series_graph = plot(ft, (x, -5, 5), color='crimson', linestyle='-.', legend_label='Taylor
        (10th Order)')
```

Create a combined plot

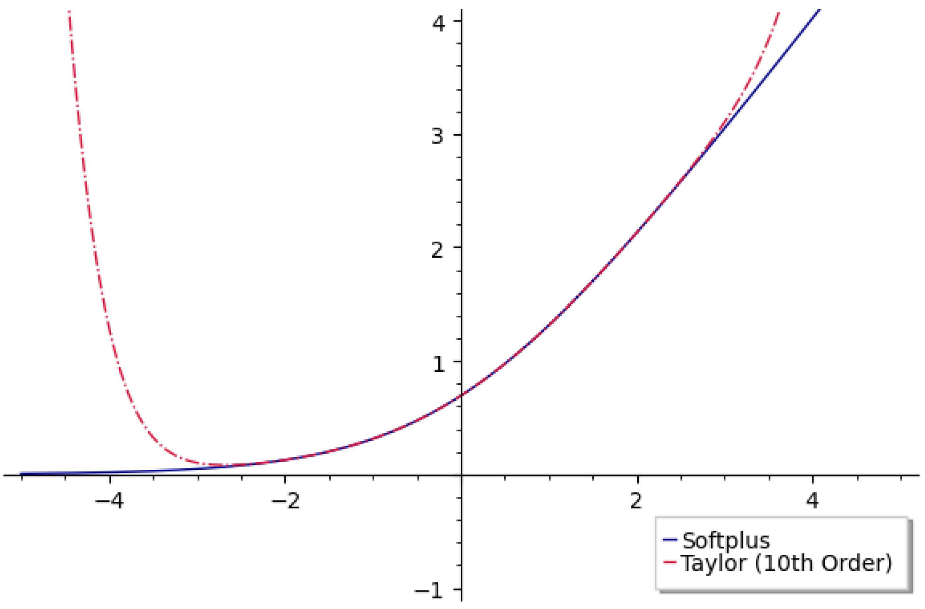
```
In [9]: final_plot = softplus_graph + taylor_series_graph
```

Define the axes range

```
In [10]: final_plot.axes_range(xmin=-5, xmax=5, ymin=-1, ymax=4)
```

Show the plot

In [11]:
```
final_plot.show(legend_loc='lower right')
```



Comment on the region of the approximation :

Near $x = 0$, the approximation (dashed line) closely follows the actual Softplus function (solid line), indicating that the approximation is very good around the point of expansion. This is expected as Taylor series provide the best approximation close to the expansion point. As x moves away from 0, particularly for $x < -2.5$ and $x > 2.5$, the approximation starts to diverge from the actual function. This divergence becomes more pronounced the further x is from 0. The region where the approximation is valid (i.e., where the Taylor series closely matches the Softplus function) appears to be approximately between $x = -2.5$ and $x = 2.5$. Outside of this interval, the error between the Taylor series approximation and the Softplus function increases significantly.

(iii)

Compute the derifative of f analytically

In [12]:
```
analytical_derivative = f.diff(x);analytical_derivative
```

Out[12]:
$$\frac{e^x}{e^x + 1}$$

(iv)

Compute the derivative of f approximately using the Taylor series approximation from (i)

In [13]:
```
approx_derivative = ft.diff(x);approx_derivative
```

Out[13]:
$$\frac{31}{1451520}\,x^9 - \frac{17}{80640}\,x^7 + \frac{1}{480}\,x^5 - \frac{1}{48}\,x^3 + \frac{1}{4}\,x + \frac{1}{2}$$

(v)

Plot the derivatives seperately

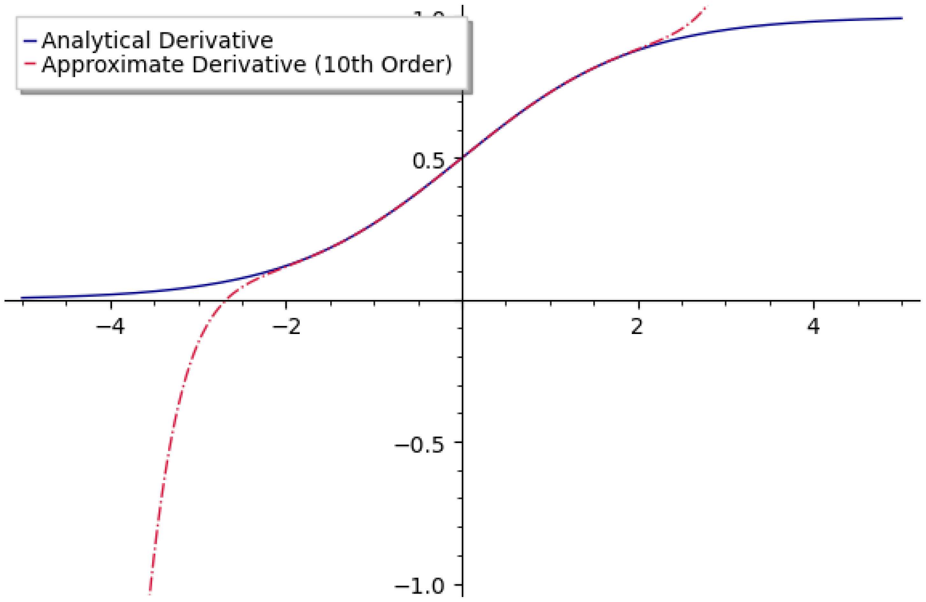
```
In [14]: analytical_plot = plot(analytical_derivative, (x, -5, 5), color='darkblue', legend_label='Analytic
         al Derivative')
         approx_plot = plot(approx_derivative, (x, -5, 5), color='crimson', linestyle='-.', legend_label='A
         pproximate Derivative (10th Order)')# $(iii)$
```

Create a combined plot

```
In [15]: combined_plot = analytical_plot + approx_plot
```

Set the axes range and show the plot

```
In [16]: combined_plot.set_axes_range(-5, 5, -1, 1)
         combined_plot.show()
```



[DAMA 50] Written Assignment 4

Panagiotis Paltsokas - std163861

Problem 8

Consider the *sigmoid* function

$$f(w_0, w_1, w_2, x_1, x_2) = \frac{1}{1+e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \tag{2.1}$$

(a) Install the automatic differentiation library **autograd** in **sagemath** and all additional libraries required for its functionality.
(b) Define f as a python function and utilise **autograd** to define a function named **augradf** that automatically computes the numerical value of the gradient of f with respect to w_0, w_1, w_2, x_0, x_1.
(c) Define a **sagemath** function **angradf** that computes analytically the gradient of f with respect to w_0, w_1, w_2, x_0, x_1.
(d) Compare the results of the functions **augradf** and **angradf** for 10 points $(w_0, w_1, w_2, x_0, x_1)$ choosed randomly in the range $(0, 1)$, and comment on any discrepancies.
(e) Compare the execution time of the two methods of computation of the gradient for a specific point (chosen randomly).

```
In [17]: restore()
```

```
In [18]: %display latex
```

(a)

Install the autograd package if not installed already.

```
In [19]: try :
             import autograd
             print ("Autograd is already installed!")
         except :
             print ("Installing autograd.")
             !pip install autograd
```

```
Autograd is already installed!
```

Import grad and numpy from autograd.

```
In [20]: import autograd.numpy as np
         from autograd import grad
```

(b)

Define the function $f(w_0, w_1, w_2, x_0, x_1) = \frac{1}{1+e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$

```
In [21]: def myf(w):
             w0, w1, w2, x0, x1 = w
             z = w0 * x0 + w1 * x1 + w2
             return 1 / (1 + np.exp(-z))
```

Define augradf using autograd

```
In [22]: augradf = grad(myf)
```

```
In [23]: #test to see if it produces results
         w_values = np.array([0.5, 0.7, -0.2, 1.0, 2.0])
         gradient = augradf(w_values)
         print("Numerical Gradient:", gradient)
```

```
Numerical Gradient: [0.13060575 0.26121149 0.13060575 0.06530287 0.09142402]
```

(c)

Define the symbolic variables

```
In [24]: w0, w1, w2, x0, x1 = var('w0 w1 w2 x0 x1')
         varlist = w0, w1, w2, x0, x1
```

Define the symoblic representation of the sigmoid function for analysis

```
In [25]: f=1/(1+exp(-(w0 * x0 + w1 * x1 + w2)))
```

Create an empty list to store the gradients

```
In [26]: gradients = []
```

Compute the gradients analytically w.r.t. each variable and store them in the empty list we created earlier

```
In [27]: # Run only once otherwise you append the same derivatives multiple times
         for var in varlist:
             gradient = diff(f, var)
             gradients.append(gradient)
```

```
In [28]: gradients
```

Out[28]:

$$\left[\frac{x_0 e^{(-w_0x_0-w_1x_1-w_2)}}{\left(e^{(-w_0x_0-w_1x_1-w_2)}+1\right)^2}, \frac{x_1 e^{(-w_0x_0-w_1x_1-w_2)}}{\left(e^{(-w_0x_0-w_1x_1-w_2)}+1\right)^2}, \frac{e^{(-w_0x_0-w_1x_1-w_2)}}{\left(e^{(-w_0x_0-w_1x_1-w_2)}+1\right)^2}, \frac{w_0 e^{(-w_0x_0-w_1x_1-w_2)}}{\left(e^{(-w_0x_0-w_1x_1-w_2)}+1\right)^2}, \frac{w_1 e}{(e^{(-w}} \right.$$

Define the angradf function to return the gradient as a list

```
In [29]: def angradf(w):
             coord = {w0: w[0], w1: w[1], w2: w[2], x0: w[3], x1: w[4]}
             gradient = [gradients[0].subs(coord), gradients[1].subs(coord), gradients[2].subs(coord), grad
         ients[3].subs(coord), gradients[4].subs(coord)]
             return gradient
```

```
In [30]: #test to see if it produces results
         w_values = [0.5, 0.7, -0.2, 1.0, 2.0]
         gradient_analytical = angradf(w_values)
         print("Analytical Gradient:", gradient_analytical)
```

```
Analytical Gradient: [0.130605746966208, 0.261211493932416, 0.130605746966208, 0.0653028734831040,
0.0914240228763456]
```

(d)

Create empty lists to store 10 randomly generated points, their autograd computed gradients, their analytically computed gradients and their absolute differences.

In [31]:
```
random_points=[]
```

In [32]:
```
auto_val=[]
```

In [33]:
```
an_val=[]
```

In [34]:
```
differences=[]
```

1.Initialize an empty list for each point,

2.Generate a random coordinate,

3.Append the coordinate to the point,

4.Compute the autograd,

5.Append the autograd to the list of autograd values,

6.Compute the analytical grad,

7.Append the gradient to the list of analytical values,

8.Append the point to the list of points

In [35]:
```
for _ in range(10):
    point = []  # 1
    for _ in range(5):
        coordinate = random()  # 2
        point.append(coordinate)  # 3
    autogradval = augradf(point) # 4
    auto_val.append(autogradval) # 5
    angradval = angradf(point) # 6
    an_val.append(angradval) # 7
    random_points.append(point)  # 8
```

In [36]:
```
auto_val
```

Out[36]:
```
[[0.19536985776832724, 0.07267391737936668, 0.20066335295312995, 0.10704879800002969, 0.0355674422
 [0.03364737631635972, 0.14836773801571201, 0.15696956225822842, 0.05737060885016759, 0.0764968466
[0.12409064039920212, 0.22533711761773742, 0.23408370494289893, 0.06759199750928128, 0.0184267436
 [0.12296451373443411, 0.11871262923923777, 0.16022264798583488, 0.10920498745504416, 0.1024339551
[0.17965973156216905, 0.13166941627701317, 0.19530683610361582, 0.05343650565639067, 0.0242566530
[0.06880901828117138, 0.09267509692498975, 0.15428614623338566, 0.004842541333780795, 0.112206277
[0.14866969589314158, 0.13283115546690816, 0.23249643482676527, 0.060573474889726524, 0.080626540
[0.008580847233106623, 0.05223851883739353, 0.20312772291525907, 0.17797395995035942, 0.074568955
    [0.11541413486366654, 0.123031323843702, 0.15808455120816728, 0.0595083637364853, 0.15700323327
 [0.09324395338441338, 0.08206279825677044, 0.19733584571312987, 0.07861873157983465, 0.1903335782
```

In [37]:
```
an_val
```

Out[37]:
```
[[0.19536985776832722, 0.0726739173793668, 0.20066335295312993, 0.10704879800002967, 0.0355674422
 [0.033647376316359716, 0.148367738015712, 0.1569695622582284, 0.05737060885016758, 0.07649684662
 [0.12409064039920212, 0.22533711761773742, 0.23408370494289893, 0.0675919975092813, 0.01842674368
 [0.12296451373443411, 0.11871262923923777, 0.16022264798583488, 0.10920498745504417, 0.1024339551
 [0.17965973156216905, 0.13166941627701317, 0.19530683610361582, 0.05343650565639067, 0.0242566530
 [0.06880901828117139, 0.09267509692498975, 0.15428614623338568, 0.004842541333780796, 0.112206277
 [0.1486696958931416, 0.13283115546690816, 0.2324964348267653, 0.06057347488972654, 0.08062654055
 [0.008580847233106622, 0.05223851883739352, 0.20312772291525905, 0.1779739599503594, 0.0745689557
 [0.11541413486366654, 0.12303132384370201, 0.15808455120816728, 0.0595083637364853, 0.1570032332
 [0.09324395338441338, 0.08206279825677044, 0.19733584571312987, 0.07861873157983465, 0.1903335782
```

Calculate the Euclidean distance between each point

In [38]:
```
for i in range(10):
    print("point",i+1,"       :",random_points[i])
    print("Automatic Grad:",[float(val) for val in auto_val[i]])
    print("Analytic Grad:",an_val[i])
    print("Distance=",\
         sqrt((auto_val[i][0]-an_val[i][0])^2+
             (auto_val[i][1]-an_val[i][1])^2+
             (auto_val[i][2]-an_val[i][2])^2+
             (auto_val[i][3]-an_val[i][3])^2+
             (auto_val[i][4]-an_val[i][4])^2))
    print("")
```

```
point 1        : [0.5334745803088103, 0.17724931718220782, 0.3713990760784198, 0.9736200202632956,
0.3621683596423392]
Automatic Grad: [0.19536985776832724, 0.0726739173793668, 0.20066335295312995, 0.1070487980000296
9, 0.03556744229443465]
Analytic Grad: [0.19536985776832722, 0.0726739173793668, 0.20066335295312993, 0.10704879800002967,
0.03556744229443465]
Distance= 4.163336342344337e-17

point 2        : [0.36548874842237256, 0.48733554151245884, 0.8789248920906776, 0.2143560562461587
2, 0.9452006865613497]
Automatic Grad: [0.03364737631635972, 0.14836773801571201, 0.15696956225822842, 0.0573706088501675
9, 0.07649684662408737]
Analytic Grad: [0.033647376316359716, 0.148367738015712, 0.1569695622582284, 0.05737060885016758,
0.07649684662408734]
Distance= 5.0515910130503314e-17

point 3        : [0.28875139995656385, 0.07871860919005003, 0.2869292483133098, 0.530112253774657
6, 0.9626347877256338]
Automatic Grad: [0.12409064039920212, 0.22533711761773742, 0.23408370494289893, 0.0675919975092812
8, 0.018426743687159044]
Analytic Grad: [0.12409064039920212, 0.22533711761773742, 0.23408370494289893, 0.0675919975092813,
0.018426743687159044]
Distance= 1.3877787807814457e-17

point 4        : [0.6815827152269938, 0.6393225704378606, 0.38719891259911665, 0.7674602515950508,
0.7409229015471834]
Automatic Grad: [0.12296451373443411, 0.11871262923923777, 0.16022264798583488, 0.1092049874550441
6, 0.10243395515266446]
Analytic Grad: [0.12296451373443411, 0.11871262923923777, 0.16022264798583488, 0.1092049874550441
7, 0.10243395515266446]
Distance= 1.3877787807814457e-17

point 5        : [0.27360284321047057, 0.12419766503077068, 0.6789119767116651, 0.919884501466474
4, 0.6741669616068063]
Automatic Grad: [0.17965973156216905, 0.13166941627701317, 0.19530683610361582, 0.0534365056563906
7, 0.024256653008616506]
Analytic Grad: [0.17965973156216905, 0.13166941627701317, 0.19530683610361582, 0.0534365056563906
7, 0.024256653008616506]
Distance= 0

point 6        : [0.031386754105942716, 0.7272608706626652, 0.9951223598670883, 0.4459831291468341
3, 0.60067024284087]
Automatic Grad: [0.06880901828117138, 0.09267509692498975, 0.15428614623338566, 0.0048425413337807
95, 0.11220627704087933]
Analytic Grad: [0.06880901828117139, 0.09267509692498975, 0.15428614623338568, 0.00484254133378079
6, 0.11220627704087936]
Distance= 4.16423974613979e-17

point 7        : [0.2605350698597174, 0.34678613722918905, 0.1773742831782078, 0.6394493575951666,
0.5713255584580537]
Automatic Grad: [0.14866969589314158, 0.13283115546690816, 0.23249643482676527, 0.0605734748897265
24, 0.08062654055313183]
Analytic Grad: [0.1486696958931416, 0.13283115546690816, 0.2324964348267653, 0.06057347488972654,
0.08062654055313184]
Distance= 4.3885418357208765e-17

point 8        : [0.8761677500052846, 0.36710378437091395, 0.7957455743397468, 0.0422436047131113
9, 0.25717079917833974]
Automatic Grad: [0.008580847233106623, 0.05223851883739353, 0.20312772291525907, 0.177973959950359
42, 0.07456895579283802]
Analytic Grad: [0.008580847233106622, 0.05223851883739352, 0.20312772291525905, 0.177973959950359
4, 0.07456895579283801]
Distance= 4.224327714674015e-17

point 9        : [0.3764337709263197, 0.9931598759974334, 0.3584958693785515, 0.7300785179931218,
0.7782627897756634]
Automatic Grad: [0.11541413486366654, 0.123031323843702, 0.15808455120816728, 0.0595083637364853,
0.1570032332750133]
Analytic Grad: [0.11541413486366654, 0.12303132384370201, 0.15808455120816728, 0.0595083637364853,
0.15700323327501334]
Distance= 3.1031676915590914e-17

point 10        : [0.3984006620577384, 0.9645159884933786, 0.40267273968951844, 0.472514018157469
2, 0.4158534804470668]
Automatic Grad: [0.09324395338441338, 0.08206279825677044, 0.19733584571312987, 0.0786187315798346
5, 0.19033357829317632]
```

```
Analytic Grad: [0.09324395338441338, 0.08206279825677044, 0.19733584571312987, 0.0786187315798346
5, 0.19033357829317632]
Distance= 0
```

Comment on any discrepancies

In comparing the automatic and analytical gradients of the sigmoid function across ten randomly chosen points, the results exhibit negligible discrepancies, with the maximum distance between the computed gradients on the order of 10^{-17}. Given the consistency of the results and the insignificant magnitude of the differences, we can conclude that both methods provide virtually identical outcomes for the gradients of the sigmoid function.

(e)

Choose a specific point (e.g., the first point)

```
In [39]: specific_point = [random() for _ in range (5)];specific_point
```

Out[39]: $[0.24509017508487807, 0.5065153201842187, 0.8714626618654139, 0.2616047566344879, 0.86055181497882$

Measure the execution time for autograd method using %timeit

```
In [40]: auto_time = %timeit -n 1000 -o augradf(specific_point)
```

```
384 µs ± 6.63 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

Measure the execution time for analytical method using %timeit

```
In [41]: an_time = %timeit -n 1000 -o angradf(specific_point)
```

```
540 µs ± 27.7 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

Print the execution times

```
In [42]: print("Execution Time (Autograd):", auto_time.best, "seconds")
         print("Execution Time (Analytical):", an_time.best, "seconds")
```

```
Execution Time (Autograd): 0.0003759757999996509 seconds
Execution Time (Analytical): 0.000505597299999863 seconds
```

The execution time of the Autograd method is slightly less than that of the Analytical method.

Problem 9

Using by-hand calculation, find an approximation of the two variable function $f(x,y) = e^x \ln(1+y)$ that comprises up to second-order terms in x, y in the Taylor series expansion around the point $(x_0, y_0) = (0,0)$.

We will use the Taylor expansion

$$f(\vec{x}) = f(\vec{x}_0) + \vec{\nabla} f(\vec{x}_0)(\vec{x}-\vec{x}_0) + \frac{1}{2}(\vec{x}-\vec{x}_0)^T H(\vec{x}_0)(\vec{x}-\vec{x}_0) + \dots \qquad (9.1)$$

around the point $\vec{x}_0 = (0,0)$, where H is the Hessian matrix

$$H = \begin{pmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{pmatrix} \qquad (9.2)$$

$f(x,y) = e^x \ln(1+y) \quad , x \in \mathbb{R}\ ,\ y \in (-1, +\infty)$

$$f(0,0) = e^0 \ln(1+0) = 1 \cdot \ln 1 = 0 \qquad (9.3)$$

<u>First order derivatives</u>

$$(\vec{\nabla} f) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right) = \left(e^x \ln(1+y), \frac{e^x}{1+y}\right) \Rightarrow \vec{\nabla} f(\vec{x}_0) = (0,1) \qquad (9.4)$$

therefore $\vec{\nabla} f(\vec{x}_0)(\vec{x}-\vec{x}_0) = [0 \quad 1] \cdot \begin{bmatrix} x-0 \\ y-0 \end{bmatrix} = [0 \quad 1] \cdot \begin{bmatrix} x \\ y \end{bmatrix} = y \qquad (9.5)$

<u>Second order derivatives</u>

$$\frac{\partial^2 f}{\partial x^2} = \frac{\partial}{\partial x}\left(e^x \ln(1+y)\right) = e^x \ln(1+y) \quad \Rightarrow \quad \frac{\partial^2 f}{\partial x^2}(0,0) = 0 \qquad (9.6)$$

$$\frac{\partial^2 f}{\partial x \partial y} = \frac{\partial}{\partial x}\left(\frac{e^x}{1+y}\right) = \frac{e^x}{1+y} \quad \Rightarrow \quad \frac{\partial^2 f}{\partial x \partial y}(0,0) = 1 \qquad (9.7)$$

$$\frac{\partial^2 f}{\partial y \partial x} = \frac{\partial}{\partial y}\left(e^x \ln(1+y)\right) = \frac{e^x}{1+y} \quad \Rightarrow \quad \frac{\partial^2 f}{\partial y \partial x}(0,0) = 1 \qquad (9.8)$$

$$\frac{\partial^2 f}{\partial y^2} = \frac{\partial}{\partial y}\left(\frac{e^x}{1+y}\right) = e^x \cdot \left(-\frac{1}{(1+y)^2}\right) \cdot \frac{\partial(1+y)}{\partial y} = -\frac{e^x}{1+y} \quad \Rightarrow \quad \frac{\partial^2 f}{\partial y^2}(0,0) = -1$$

(9.9)

Thus

$$H(\vec{x}_0) = \begin{pmatrix} 0 & 1 \\ 1 & -1 \end{pmatrix} \tag{9.10}$$

and

$$\frac{1}{2}(\vec{x}-\vec{x}_0)^T H(\vec{x}_0)(\vec{x}-\vec{x}_0) = \frac{1}{2}[x \quad y] \cdot \begin{bmatrix} 0 & 1 \\ 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \frac{1}{2}[y \quad x-y] \cdot \begin{bmatrix} x \\ y \end{bmatrix} =$$

$$= \frac{1}{2}\left(xy + xy - y^2\right) = xy - \frac{y^2}{2} \tag{9.11}$$

Substituting (9.3), (9.5) and (9.11) into (9.1) we get

$$f(x,y) = 0 + y + xy - \frac{y^2}{2} + ... = y + xy - \frac{y^2}{2} + \mathcal{O}(x,y)^3$$

[DAMA 50] Written Assignment 4

Panagiotis Paltsokas - std163861

Problem 10

Consider the function $f(x, y) = \ln(1 + x(x + y^2))$.
(i) Draw its computational graph.
(ii) Compute its gradient at $(x, y) = (2, 1)$ using the results of (i) and backpropagation.

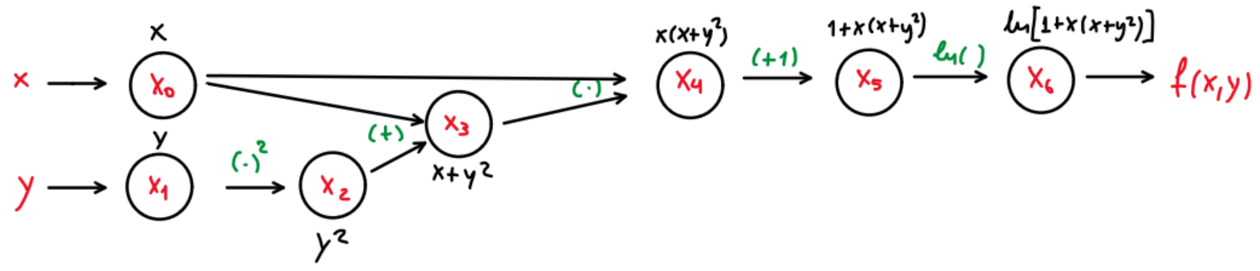
----

i.

We introduce the necessary intermediate variables following the concept of automatic differentiation

$$
\begin{aligned}
x_0 &= x & \qquad x_4 &= x_0 \cdot x_3 \\
x_1 &= y & \qquad x_5 &= x_4 + 1 \\
x_2 &= x_1^{\,2} & \qquad x_6 &= \ln(x_5) \\
x_3 &= x_0 + x_2 & &
\end{aligned}
$$

which allows us to draw the computational graph, as follows:



ii.

We will now compute the backward derivative trace. First, we substitute the values of our parameters for the point $(x, y) = (2,1)$

- $x_0 = 2$
- $x_1 = 1$
- $x_2 = 1$
- $x_3 = 3$
- $x_4 = 6$
- $x_5 = 7$
- $x_6 = \ln 7 = 1.95$

Then we calculate the gradients in reverse mode:

- $\frac{\partial f}{\partial x_6} = 1$
- $\frac{\partial f}{\partial x_5} = \frac{\partial f}{\partial x_6}\frac{\partial x_6}{\partial x_5} = 1 \cdot \frac{1}{x_5} = \frac{1}{7}$
- $\frac{\partial f}{\partial x_4} = \frac{\partial f}{\partial x_5}\frac{\partial x_5}{\partial x_4} = \frac{1}{7} \cdot 1 = \frac{1}{7}$
- $\frac{\partial f}{\partial x_3} = \frac{\partial f}{\partial x_4}\frac{\partial x_4}{\partial x_3} = \frac{1}{7} \cdot x_0 = \frac{2}{7}$
- $\frac{\partial f}{\partial x_2} = \frac{\partial f}{\partial x_3}\frac{\partial x_3}{\partial x_2} = \frac{2}{7} \cdot 1 = \frac{2}{7}$
- $\frac{\partial f}{\partial x_1} = \frac{\partial f}{\partial x_2}\frac{\partial x_2}{\partial x_1} = \frac{2}{7} \cdot 2x_1 = \frac{4}{7} = \frac{\partial f(x,y)}{\partial y}$
- $\frac{\partial f}{\partial x_0} = \frac{\partial f}{\partial x_3}\frac{\partial x_3}{\partial x_0} + \frac{\partial f}{\partial x_4}\frac{\partial x_4}{\partial x_0} = \frac{2}{7} \cdot 1 + \frac{1}{7} \cdot x_3 = \frac{2}{7} + \frac{3}{7} = \frac{5}{7} = \frac{\partial f(x,y)}{\partial x}$

Thus

$$\nabla f(x,y) = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} & \frac{\partial f(x,y)}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{5}{7} & \frac{4}{7} \end{bmatrix}$$