MASD assignment 3

Anton (ckf216), Simon (wgh762) og Kirstine (zbf480)

September 2020

Exercise 1

Igennem de to første opgaver bliver der gjort brug af geogebra's afledningsfunktion. I dette skriver man den originale funktion, og som det næste input skriver "f'(x)" hvorefter den både viser grafen og selve værdierne for den afledte funktion. Igennem de to sidste opgaver bliver der gjort brug af "Second Derivative Test" som set i "James Stewart: Calculus, early transcendentals, 9th edition (metric version)" p.1010

a)

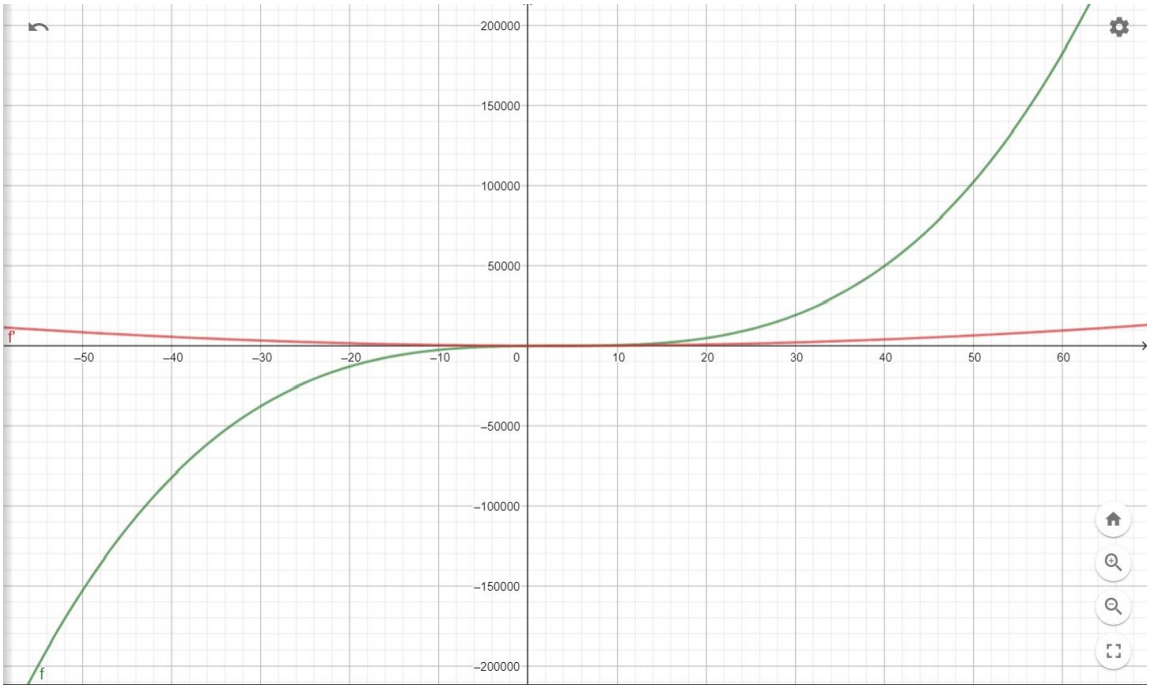

Figure 1: The function (green) and its derivative (red)

Det kan ses på billedet at der er et inflektionspunkt, men dette vil også bevises igennem den afledte funktion:

$f(x) = 2x^2 + (x-4)^3 \Rightarrow x^3 - 10x^2 + 48x - 64$

$\frac{d}{dx}\ x^3 - 10x^2 + 48x - 64 = 3x^2 - 20x + 48$

På grund af x^3, så vil forsymbolet ændrer sig, et eksempel: $-3^3 = -27 \wedge 3^3 = 27$
Dette gør det til et inflektionspunkt rundt om $x = 0$.

grunden til det kun er et inflektionspunkt, og ikke et saddelpunkt.

Dette er fordi der ikke er nogle stationære punkter overhovedet. Dette kan ses ved at der ikke er nogen værdi hvorved den udlede funktion når 0.

Det tætteste punkt til nul i den udledede funktion ville være 0, men hvis det plottes ind i funktionen ville der returneres 48.

b)

Figure 2: The function (green) and its derivative (red)

Endnu engang kan det ses ud fra billedet at: Der er et minimumspunkt ved $x = 0,6$, dette vil

dog også bevises igennem den afledte funktion.

$\frac{d}{dx}x^2ln(x) = 2xln(x) + x$

$2 * 0,6 * ln(0,6) + 0,6 = 0$

Siden denne afledte funktion er lig 0, så ville det på funktionen være et lokalt minimum/maksimum, for at tjekke at det er et lokalt minimum indsætter vi blot værdien og værdien+0,1, og ser forskellen mellem de to.

$0,6^2ln(0,6) = -0,18 \wedge 0,7^2ln(0,7) = -0,17$

Derved kan det ses at det er et lokalt minimum.

Der ville også i teori, fra graferne kunne se ud til at være et stationært punkt i origo, altså $f'(0)$, dette er dog ikke tilfældet, da ln(0) er udefineret. Grunden til det er udefineret er, at ln(y) i bund og grund finder x værdien i følgende udtryk: $e^x = y$. Og og der ikke er nogen strikt værdi som en værdi kan opløftes til, for at få 0.

c)

Der partialdifferentieres:

$\frac{d}{dx_1}x_1^2 + 2x_1x_2 + 3x_2^2 \Rightarrow 2x_1 + 2x_2$

$\frac{d}{dx_2}x_1^2 + 2x_1x_2 + 3x_2^2 \Rightarrow 6x_2 + 2x_1$

De to sættes lig 0:

$2x_1 + 2x_2 = 0$
$6x_2 + 2x_1 = 0$

x_2 isoleres i den ene:

$2x_1 + 2x_2 = 0 \Rightarrow x_2 = -x_1$

Og indsættes i den anden ligning:

$6(-x_1) + 2x_1 = 0 \Rightarrow -4x_1 = 0$

$x_2 = -x_1 = 0$

Der gøres brug af "Second Derivative Test":

$D = f_{x_1x_1}(x_1, x_2) * f_{x_2x_2}(x_1, x_2) - (f_{x_1x_2}(x_1, x_2))^2 = 2 * 6 - 2^2 = 8$

Siden $D = 8 \wedge f_{x_1x_1} = 2$ (altså begge er større en nul) så er det et lokalt minimumspunkt.

d)

Først udvides udtrykket:

$$(x_1 - x_2)^2 \Rightarrow x_1^2 + x_2^2 - 2x_1x_2$$

Derfra gennemgås samme proces som i (c).

$$2x_1 - 2x_2 = 0$$
$$2x_2 - 2x_1 = 0$$

x_2 isoleres i første ligning:

$$2x_1 - 2x_2 = 0 \Rightarrow x_2 = x_1$$

indsættes i anden ligning:

$$2(x_1) - 2x_1 = 0 \Rightarrow 0 = 0$$

Derved kan det ses at ved ethvert punkt hvor $x_1 = x_2$ er der et lokalt extremum. For at afgøre hvilket slags extremum det er, kigges der på funktionsligningen:

$$f(x_1, x_2) = (x_1 - x_2)^2$$

Da funktionsligningen er i andenpotens, vil den aldrig kunne nå en negativ værdi, på denne måde ville alle værdier hvor $x_1 = x_2$ føre til den mindste værdi, 0. Ergo er alle extremumspunkterne lokale minimumspunkter.

Exercise 2

a)

modellen gør det, at den finder to matricer, A og B, vis produkt er det tætteste på M. Da argmin betyder at den finder de argumenter for A og B der gør vektor-normen så lille som muligt.

En måde at fortolke A og B matricerne kunne være som en række parametre, hvor en række i A ville være en persons mening om de parametre, og en kolonne i B ville være hvor meget en film holder sig til de parametrer. Hvor så mængden af kolonner i A og rækker i B er mængden af parametrer der undersøges.

b)

Til at starte med vil det vises hvordan $AB = a_{i1}b_{1j} + a_{i2}b_{2j}$:

En måde det ovenstående udtryk kan læses på er: "summen af produkterne af enkelte søjleværdier og enkelte rækkeværdier." eller, yderligere simplificeret "summen af rækker af A gange søjler af B". Dette kan vises med to $2x2$ matrix:

$$A = \begin{pmatrix} 4 & 6 \\ 3 & 7 \end{pmatrix}$$
$$B = \begin{pmatrix} 6 & 2 \\ 1 & 5 \end{pmatrix}$$
$$AB = \begin{pmatrix} A_{11} * B_{11} + A_{12} * B_{21} & A_{11} * B_{12} + A_{12} * B_{22} \\ A_{21} * B_{11} + A_{22} * B_{21} & A_{21} * B_{12} + A_{22} * B_{22} \end{pmatrix} = \begin{pmatrix} 4*6+6*1 & 4*2+6*5 \\ 3*6+7*1 & 3*2+7*5 \end{pmatrix} = \begin{pmatrix} 30 & 38 \\ 25 & 41 \end{pmatrix}$$

Derved, igennem eksempel kan det ses at udtrykket AB ville blive til $a_{i1}b_{1j} + a_{i2}b_{2j}$ hvor i og j er variable værdier der respektivt svarer til A's antal af rækker og B's antal af søjler.

Som det næste, grunden til at $\odot$ i sumudtrykket bliver til almindelig gange, er fordi at $\odot$ er elementvis mulitplikation, og i den samme tankegang ville det, med specifikke koordinater, være et almindeligt gangetegn.

Grunden til de dobbelte sum-tegn er på grund af det at det er matrix-normen der findes. Altså, alle værdier i matricen konkatineret, dette ville svare til alle elementerne summet sammen.

Derigennem kan det ses at udtrykket:

$$||I \odot (M - AB)||^2 = \sum_{i=1}^{6} \sum_{j=1}^{10} I_{ij} * (M_{ij} - (a_{i1}b_{1j} + a_{i2}b_{2j}))^2$$

er sandt.

Exercise 4

a)

python code

```
def netflix_gradient(M, A, B):
   # list variables
    I = np.array([[1,1,1,0,0,1,1,1,1,1],[0,0,1,1,1,1,1,0,0,1],
    [1,1,0,1,0,0,0,1,1,1],[1,0,1,0,0,1,1,1,0,0],[0,1,1,0,1,0,1,0,1,0]
    ,[1,1,0,0,1,1,0,1,0,1]])
    t = []; p = []; n = []; tt = []; pp = []; nn = []
    # calculating A gradient
    for k in range(0,6):
        for m in range(0,2):
            for j in range(0, 10):
                h = ((-M[k,j]*B[m][j]+A[k][0]*B[0][j]
                *B[m][j]+A[k][1]*B[1][j]*B[m][j])*I[k][j])
                t.append(h)
```

```
            s = np.sum([t]) * 2
            t = []
            x = s
            p.append(x)
        n.append(p)
        p = []
    grad_A = np.array(n)
    # calculating B gradient
    for m in range(0,2):
        for l in range(0,10):
            for i in range(0, 6):
                hh = ((-M[i,l]*A[i][m]+A[i][0]*A[i][m]
                *B[0][l]+A[i][1]*A[i][m]*B[1][l])*I[i][l])
                tt.append(hh)
            ss = np.sum([tt]) * 2
            tt = []
            xx = ss
            pp.append(xx)
        nn.append(pp)
        pp = []
    grad_B = np.array(nn)
    return grad_A, grad_B



def netflix(M):
    # A template function for performing gradient descent over A and B to minimize E
    # Input: The matrix M to be approximated
    people,movies = M.shape

    # Initialize at random
    np.random.seed(1) # Fixed seed for reproducibility; please don't change
    A = np.random.randn(people,2)
    B = np.random.randn(2,movies)


    # Set learning rate (you may want to play with this)
    learningrate = 0.0001

    # Setting parameters for convergence check
    num_iter = 1                    # This is the variable that will keep track of the n
    convergence = 0                 # This is the variable that will keep track of wheth
    max_iter = 10000                # We stop the algorithm after this many iterations
    tolerance = 0.01                # We conclude convergence when the magnitude of the
                                    # is less than the tolerance
```

```
# gradient descent
while convergence == 0:
    grad_AB = netflix_gradient(M, A, B)
    grad_A = grad_AB[0]
    grad_B = grad_AB[1]
    learnGrad_A = grad_A * learningrate
    learnGrad_B = grad_B * learningrate
    A = A - learnGrad_A
    B = B - learnGrad_B

    # Check for convergence
    num_iter = num_iter + 1
    cur_grad_norm = np.sqrt(np.linalg.norm(grad_A**2) + np.linalg.norm(grad_B**2))
    print("cur_grad_norm, num_iter", cur_grad_norm, num_iter)


    if cur_grad_norm < tolerance:
        convergence = 1
        print('converged')
    elif num_iter > max_iter:
        convergence = 1
        print('reached maximum nr of iterations')
print(np.matmul(A, B))

return A, B
```

(short implementation description):
in the netflix-gradient function did we calculate the partial derivatives using some for-loops to get the gradient of A and B matrices.
in the netflix function should we calculate gradient descent over A and B to minimize the error function. The setup in that function was build in from start. So it was primarily the calculations in the while loop that has to be written. First we called our netflix-gradient function, so we could use our gradients, then we muliply them by learning rate, to get very small numbers. Then we took the difference between these random matrices and the gradient times learning rate. The $cur - grad - norm$ is equal to square root of the gradients merged. The while loop is running while cur-grad-norm is bigger than tolerance or it has been running more times than max-iterations.

(the three matrices (A and B: rounded to 2 decimals) (M: rounded to nearest integer)):

$$A = \begin{bmatrix} 0.68 & -3.45 \\ 0.34 & -3.71 \\ -0.37 & -3.65 \\ 3.11 & -1.03 \\ 3.49 & -1.00 \\ 3.23 & -0.93 \end{bmatrix} \quad B = \begin{bmatrix} -0.33 & -0.40 & -0.39 & -0.28 & -0.33 & 2.97 & 2.38 & 2.61 & 1.87 & 2.25 \\ -2.42 & -2.41 & -2.70 & -2.30 & -2.71 & -0.01 & -0.63 & -0.48 & -0.48 & -1.41 \end{bmatrix}$$

$$M' = \begin{bmatrix} 8 & 8 & 9 & 8 & 9 & 2 & 4 & 3 & 3 & 6 \\ 9 & 9 & 10 & 8 & 10 & 1 & 3 & 3 & 2 & 6 \\ 9 & 9 & 10 & 9 & 10 & -1 & 1 & 1 & 1 & 4 \\ 1 & 1 & 2 & 2 & 2 & 9 & 8 & 9 & 6 & 8 \\ 1 & 1 & 1 & 1 & 2 & 10 & 9 & 10 & 7 & 9 \\ 1 & 1 & 1 & 1 & 1 & 10 & 8 & 9 & 7 & 9 \end{bmatrix}$$

we can see that M' is close to the original M, just with some oscillations by 1. So we can conclude that this method is not 100 percent accurate to predict future data. But it can be used to predict some data with almost none fluctuations. And maybe we could have even fewer fluctuations with a smaller number as learning rate and tolerance

b)

a strength with this approach to predict these future Netflix data is that we get a sensation of what movies these people like. A weakness is as we also saw in a) that this approach is not 100 percent accurate because we have some fluctuations in our already discovered data.

A way this matrix factorization could be improve, would be if we have even more iterations and a smaller learning rate. Then we could come even closer to a slope equal to 0.