assignment2-Q2

March 1, 2022

1 Q2) Gradient Descent Algorithm

[35]:
```
import random
import sys
from math import floor, log10, sqrt
import csv
```

i) Consider the last 4 digits of your mobile number (Note : In case there is a 0 in one of the digits replace it by 3). Let it be n1n2n3n4. Generate a random matrix A of size n1n2 × n3n4. For example, if the last four digits are 2311, generate a random matrix of size 23 × 11. Write a code to calculate the l∞ norm of this matrix.

`Deliverable(s) : The code that generates the results. (0.5)`

---

[1]:
```
# Significant digit conversion
'''
This method convert a digit into significant digit
x: digit
sig: significant digit form ex: 5 = r.dddd
'''
def tidy(x, sig):
    y = abs(x)
    if y <= sys.float_info.min:
        return 0.0000
    return round(x, sig-int(floor(log10(y)))-1)
```

[2]:
```
# Matrix generator of size m x n
def generate_matrix(m, n, low_num=0.10000, up_num=9.9999, sig=5):
    return [[tidy(random.uniform(low_num, up_num), sig) for i in range(n)] for␣
 ↪j in range(m)]
```

[14]:
```
# Calculate infinity norm of matrix m
'''
This method calculate infinity norm of a matrix
m: input matrix
sig: significant digit form ex: 5 = r.dddd
'''
```

```
def norm_infinity(m, sig=5):
    rowmax = 0
    for r in range(len(m)):
        temp = 0
        for c in range(len(m[r])):
            temp = tidy(temp + abs(m[r][c]), sig)
        if temp > rowmax:
            rowmax = temp
    return rowmax
```

[15]:
```
'''
Main entry point method for the Q2 part i answer, to calculate infinity norm␣
 ↪and generate n1n2xn3n4 size matrix
last_4_mob_no_digits: last 4 mobile number digit in string
'''
def run_infinity_norm_calc(last_4_mob_no_digits="0029"):
    # Get Matrix Size m x n
    last_4_digits = ""
    for digit in last_4_mob_no_digits:
        if digit == "0":
            last_4_digits = last_4_digits + "3"
        else:
            last_4_digits = last_4_digits + digit
    m, n = int(last_4_digits[:2]), int(last_4_digits[2:])

    # Generate Matrix
    matrix = generate_matrix(m, n)

    # Calculate infinity norm
    infi_norm = norm_infinity(matrix)
    print(f"\nInfinity Norm: {infi_norm}")
```

[16]:
```
run_infinity_norm_calc()
```

```
Infinity Norm: 170.74
```

ii) Generate a random vector b of size n1n2 × 1 and consider the function f (x) = 1/2 Ax − b ^22 where · 2 is the vector 2 norm. Its gradient is given to be f (x) = A Ax − A b. Write a code to find the local minima of this function by using the gradient descent algorithm (by using the gradient expression given to you). The step size in the iteration xk+1 = xk − f (xk) should be chosen by the formula = gTk gkgTk ATAgk where gk = f (xk) = A Axk − A b. The algorithm should execute until xk − xk−1 2 < 10−4.

Deliverable(s) : The code that finds the minimum of the given function and the expression for . The values of xk and f (xk) should be stored in a file. (1)

---

[21]:
```
'''
This method calculate multiplication of two input matrices
mat1: input matrix a
mat1: input matrix b
d: significant digit form ex: 5 = r.dddd
'''
def get_matrix_multiplication(mat1, mat2, d=5):
    result = [[tidy(sum(tidy(a * b, d) for a, b in zip(mat1_row, mat2_col)), d)␣
 ↪for mat2_col in zip(*mat2)] for mat1_row in mat1]
    return result
```

[24]:
```
'''
This method calculate subtraction of two input matrices
mat1: input matrix a
mat1: input matrix b
sig: significant digit form ex: 5 = r.dddd
'''
def get_matrix_subtraction(mat1, mat2, d=5):
    result = [[tidy(mat1[m][n] - mat2[m][n], d) for n in range(len(mat1[0]))]␣
 ↪for m in range(len(mat1))]
    return result
```

[26]:
```
'''
This method calculate transpose of given input matrix
m: input matrix
'''
def get_matrix_transpose(m):
    result = [[m[j][i] for j in range(len(m))] for i in range(len(m[0]))]
    return result
```

[27]:
```
'''
This method calculate scalar product of a matrix and scalar
mat: input matrix
k: input scalar
d: significant digit form ex: 5 = r.dddd
'''
def get_scalar_product_matrix(mat, k, d=5):
    for i in range(len(mat)):
        for j in range(len(mat[0])):
            mat[i][j] = tidy(mat[i][j] * k, d)
    return mat
```

[28]:
```
'''
This method calculate norm-2 of a vector
v: input vector
d: significant digit form ex: 5 = r.dddd
'''
```

```
def norm_2(v, d=5):
    sqr_sum = 0
    for elm in v:
        elm = elm[0]
        abs_elm = abs(elm)
        sqr_sum = tidy(sqr_sum + tidy(abs_elm ** 2, d), d)
    result = tidy(sqrt(sqr_sum), d)
    return result
```

[29]:
```
'''
This method calculate tau
'''
def get_tau(gk, gkT, ATA, d=5):
    gkTgk = get_matrix_multiplication(gkT, gk)
    gkTATA = get_matrix_multiplication(gkT, ATA)
    gkTATAgk = get_matrix_multiplication(gkTATA, gk)
    gkTgk_div_gkTATAgk = tidy(gkTgk[0][0] / gkTATAgk[0][0], d)
    return gkTgk_div_gkTATAgk
```

[39]:
```
'''
Main entry point method for the Q2 part ii answer, to calculate minimum of f(x)␣
 ↪and tau
last_4_mob_no_digits: last 4 mobile number digit in string
max_iteration: maximum number of iteration to run
filepath: output csv file path to save values of iteration, xk and f(xk)
d: significant digit
'''
def run_optimizer(last_4_mob_no_digits="0029", max_iteration = 5000,␣
 ↪filepath="gradient_descent_results.csv", d=5):
    # Get Matrix Size m x n
    last_4_digits = ""
    for digit in last_4_mob_no_digits:
        if digit == "0":
            last_4_digits = last_4_digits + "3"
        else:
            last_4_digits = last_4_digits + digit
    m, n = int(last_4_digits[:2]), int(last_4_digits[2:])

    # Generate Matrix A and Vector b
    A = generate_matrix(m, n)
    b = generate_matrix(m, 1)

    # Initial points
    x = [[tidy(0.5, d)] for j in range(n)]

    # Create CSV file
    csvfile = open(filepath, "w")
```

```
    csvwriter = csv.writer(csvfile)
    csvwriter.writerow(["Iteration", "Xk", "FXk"])

    # Optimizer
    itr = 0
    while True:
        Ax = get_matrix_multiplication(A, x)
        Ax_minus_b = get_matrix_subtraction(Ax, b)
        Ax_minus_b_n2 = norm_2(Ax_minus_b)
        fx = tidy((tidy(Ax_minus_b_n2 ** 2, d)) / 2, d)
        # Write results to CSV
        csvwriter.writerow([itr, [elm[0] for elm in x], fx])
        AT = get_matrix_transpose(A)
        ATA = get_matrix_multiplication(AT, A)
        ATAx = get_matrix_multiplication(ATA, x)
        ATb = get_matrix_multiplication(AT, b)
        del_fx = get_matrix_subtraction(ATAx, ATb)
        gkT = get_matrix_transpose(del_fx)
        tau = get_tau(del_fx, gkT, ATA)
        tau_del_fx = get_scalar_product_matrix(del_fx, tau)
        x_new = get_matrix_subtraction(x, tau_del_fx)
        error = get_matrix_subtraction(x_new, x)
        error_n2 = norm_2(error)
        x = x_new
        itr = itr + 1
        if itr == max_iteration or error_n2 < 0.0001:
            print(f"\nLast Iteration No: {itr}")
            print(f"Tau Value: {tau}\nMinimum f(x) Value: {fx}\n||Xk - Xk-1||2␣
↪Value: {error_n2}")
            break
    csvfile.close()
```

[40]:
```
run_optimizer(last_4_mob_no_digits="0029", max_iteration=4000,␣
↪filepath="gradient_descent_results.csv")
```

```
Last Iteration No: 1128
Tau Value: 4.6486e-05
Minimum f(x) Value: 17.061
||Xk - Xk-1||2 Value: 9.6021e-05
```