TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA CÔNG NGHỆ THÔNG TIN

ĐỀ THI VÀ BÀI LÀM

Tên học phần: Toán ứng dụng CNTT
Mã học phần: Hình thức thi: *Tự luận có giám sát*
Đề số: **0003** Thời gian làm bài: 90 phút *(không kể thời gian chép/phát đề)*
Được sử dụng tài liệu khi làm bài.

Họ tên:…Trương Minh Phước……**Lớp**:…19TCLC_DT6……**MSSV**:…102190283...

Sinh viên làm bài trực tiếp trên tệp này, lưu tệp với định dạng MSSV_HọTên.pdf và nộp bài thông qua MS Teams.

Câu 1 (*2 điểm*):

a) *(1 điểm)* Cho số nguyên dương n (n>1). Hãy viết hàm phân tích n ra thành tích hữu hạn của các số nguyên tố.

 # Trả lời: Dán code vào bên dưới:

```
def PhanTich(n):
    dictPrime = {}
    for i in range(2, n+1):
        count = 0
        while n % i == 0:
            count += 1
            n = n//i
        if count != 0:
            dictPrime[i] = count
    i = 0
    for key, value in dictPrime.items():
        print(str(key)+"^"+str(value), end="")
        if i != len(dictPrime)-1:
            print(" x ", end="")
        i = i+1
```

n = 26000

PhanTich(n)

Trả lời: Dán kết quả thực thi vào bên dưới, biết rằng n=26000.

```
PS D:\CODE\Python\CS-Math> & "C:/Program Files/Python39/python.exe" d:/CODE/P
2^4 x 5^3 x 13^1
PS D:\CODE\Python\CS-Math>
```

b) *(1 điểm)* Cho hệ đồng dư sau: $\begin{cases} x \equiv 2 \bmod 3 \\ x \equiv 3 \bmod 5 \\ x \equiv 4 \bmod 11 \end{cases}$.

Viết hàm giải hệ phương trình đồng dư trên.

Trả lời: Dán code vào bên dưới:

```
from math import gcd


def ModularInverse(a, m):
    if gcd(a, m) == 1:
        i = 0
        while True:
            if (a*i-1) % m == 0:
                return i
            else:
                i = i+1
    else:
        return None
```

```
def ChineseTheorem(a, m):
    n = len(a)
    M = 1
    for x in m:
        M = M*x
    mi = [0]*n
    y = [0]*n
    x = 0
    for i in range(0, n):
        mi[i] = M//m[i]
        y[i] = ModularInverse(mi[i], m[i])
        x = x+a[i]*mi[i]*y[i]
    return Optimize(x, M)



def Optimize(x, M):
    # Tìm i nhỏ nhất để i*M>x
    i = 0
    while i*M < x:
        i = i+1
    # Trừ x đi một lượng (i-1)*M để nó vẫn > 0
    return x-(i-1)*M, M



a = [2, 3, 4]
m = [3, 5, 11]
x, M = ChineseTheorem(a, m)
print(str(x)+"(mod"+str(M)+")")
```

Trả lời: Dán kết quả thực thi vào bên dưới:

```
PS D:\CODE\Python\CS-Math> & "C:/Program Files/Python39/py
ungHoa.py
158(mod165)
PS D:\CODE\Python\CS-Math>
```

Câu 2 (*3 điểm*): Cho ma trận A. Viết hàm phân rã ma trận A bằng phương pháp Cholesky decomposition

Trả lời: Dán code vào bên dưới (bao gồm điều kiện của ma trận A nếu có):

```
import numpy as np
from numpy.linalg import eigvals


def cholesky_decomposition(A):
    L = np.zeros_like(A)
    n = len(A)
    for j in range(n):
        for i in range(j, n):
            if i == j:
                sumk = 0
                for k in range(j):
                    sumk += L[i, j]**2
                L[i, j] = np.sqrt(A[i, j]-sumk)
            else:
                sumk = 0
                for k in range(j):
                    sumk += L[i, k]*L[j, k]
                L[i, j] = (A[i, j]-sumk)/L[j, j]
    return L
```

```
# Ma trận đối xứng
def is_symmetric_matrix(A):
    n = len(A)
    for i in range(0, n):
        for j in range(0, n):
            if A[i, j] != A[j, i]:
                return False
    return True



# Ma trận xác định dương, nghĩa là mọi giá trị riêng phải dương
def is_positive_definite_matrix(V):
    for x in V:
        if x <= 0:
            return False
    return True



# Kiểm tra điều kiện để phân rã
def matrix_can_use_cholesky(A):
    m, n = A.shape
    if m != n:
        print(">> Ma tran A khong vuong !")
        return False

    if not is_symmetric_matrix(A):
        print(">> Ma tran A khong doi xung !")
        return False
```

```
    V = eigvals(A)
    if not is_positive_definite_matrix(V):
        print(">> Ma tran A khong xac dinh duong !")
        return False
    return True


A = np.array([[5, -2],
              [-2, 7]], dtype=float)
if matrix_can_use_cholesky(A):
    L = cholesky_decomposition(A)
    print("> L : \n", L)
    print("> Kiem tra L.LT : \n", L.dot(L.T))
else:
    print("Matrix A cannot use cholesky decomposition")
```

Trả lời: Dán kết quả thực thi vào bên dưới biết rằng $A = \begin{bmatrix} 5 & -2 \\ -2 & 7 \end{bmatrix}$, sai số $\varepsilon = 10^{-5}$.

```
PS D:\CODE\Python\CS-Math> & "C:/Program Files/Python39/python.exe"
py
> L :
 [[ 2.23606798  0.        ]
 [-0.89442719  2.64575131]]
> Kiem tra L.LT :
 [[ 5.  -2. ]
 [-2.   7.8]]
PS D:\CODE\Python\CS-Math>
```

Câu 3 (2 *điểm*): Cho mười điểm trong không gian Oxy như sau: (3, 4); (5,3); (6,5); (7,6); (8,7); (4,9); (3,8); (4,8); (7,10); (7,4)

a) *(1.0 điểm) Mô tả thuật toán xác định bao lồi và xác định bao lồi*

 # Trả lời: dán sơ đồ khối hoặc ngôn ngữ tự nhiên vào bên dưới:

Để tìm bao lồi, ta thực hiện 6 bước như sau:

Bước 1 : Ta cần sắp xếp các điểm tăng dần theo thứ tự hoành độ x (nếu x bằng nhau thì ưu tiên tung độ y)

Bước 2 : Ta chọn 2 điểm có hoành độ nhỏ nhất và lớn nhất để làm đường phân cách 2 vùng bao

Bước 3: Ta xác định bao trên Lupper bằng cách bổ sung 2 điểm đầu tiên, bổ sung thêm điểm thứ 3 và xoá điểm giữa nếu ba điểm không tạo thành rẽ phải, thực hiện lặp theo chiều kim đồng hồ

Bước 4: Tương tự, ta xác định bao dưới Llower bằng cách bổ sung 2 điểm đầu tiên, bổ sung điểm thứ 3 và xoá điểm giữa nếu ba điểm không tạo thành rẽ phải, thực hiện lặp theo chiều kim đồng hồ

Bước 5 : Ta xoá điểm đầu và điểm cuối trong Llower

Bước 6 : Ta thu được tập hợp điểm thu được theo chiều kim đồng hồ (Lupper hợp Llower)

chính là bao lồi cần tìm.

b) *(1.0 điểm)* Viết hàm xác định bao lồi

 # Trả lời: Dán code bên dưới:

```
import numpy as np


class Point():
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y


def RightTurn(p, q, r):
    return (q.x*r.y+p.x*q.y+p.y*r.x)-(q.x*p.y+q.y*r.x+p.x*r.y) < 0


def ConvexHull(P):
    sorted(P, key=lambda p: p.x)
    L_upper = [P[0], P[1]]
    for i in range(2, len(P)):
        L_upper.append(P[i])
```

```
        while len(L_upper) > 2 and not RightTurn(L_upper[-1], L_upper[-2], L_upper[-3]):
            del L_upper[-2]
    L_lower = [P[-1], P[-2]]
    for i in range(len(P)-3, -1, -1):
        L_lower.append(P[i])
        while len(L_lower) > 2 and not RightTurn(L_lower[-1], L_lower[-2], L_lower[-3]):
            del L_lower[-2]
    del L_lower[0]
    del L_lower[-1]
    L = L_upper + L_lower
    return np.array(L[:-1])


if __name__ == '__main__':
    P = np.array([Point(3, 4),
                  Point(5, 3),
                  Point(6, 5),
                  Point(7, 6),
                  Point(8, 7),
                  Point(4, 9),
                  Point(3, 8),
                  Point(4, 8),
                  Point(7, 10),
                  Point(7, 4)])
    for ch in ConvexHull(P):
        print("(", ch.x, ",", ch.y, ")")
```

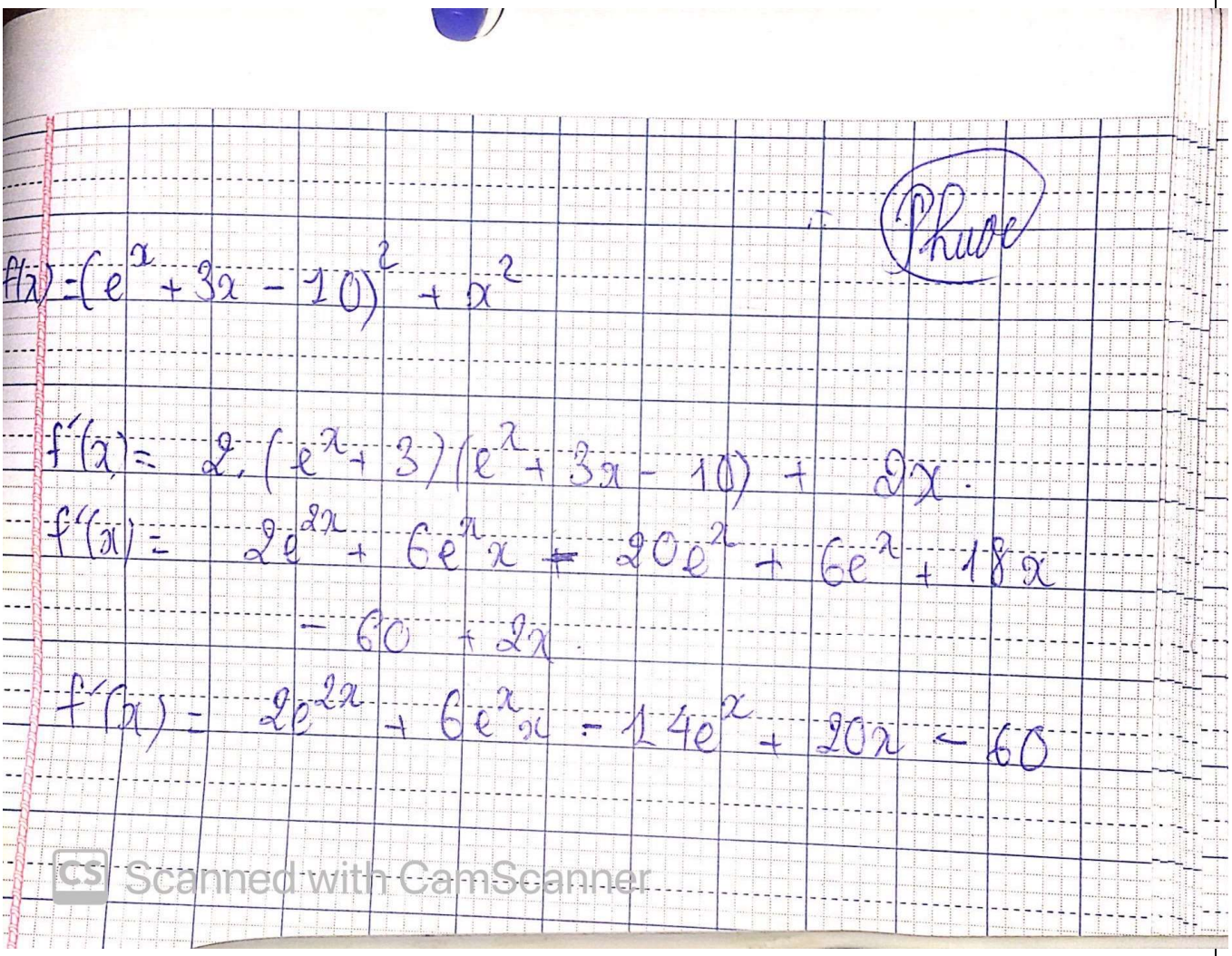
Trả lời: Dán kết quả thực thi vào bên dưới:

```
PS D:\CODE\Python\CS-Math> & "C:/Program Files/Python3
ullClass.py
( 3 , 4 )
( 5 , 3 )
( 8 , 7 )
( 4 , 9 )
( 3 , 8 )
( 7 , 4 )
( 7 , 10 )
PS D:\CODE\Python\CS-Math>
```

Câu 4 (2 *điểm*): Cho hàm số $f(x)=\left(e^x+3x-10\right)^2+x^2$.

a) *(1 điểm) Khai triển đạo hàm cấp 1 của f(x)*

Trả lời: Khai triển đạo hàm:



b) *(1 điểm)* Viết chương trình (có dùng hàm) tính giá trị bé nhất của f(x) sử dụng phương pháp *gradient descent với momentum* với tham số học (learning rate) γ, hệ số động lượng là α, số bước lặp N và sai số ε:

 # Trả lời: Dán code vào bên dưới:

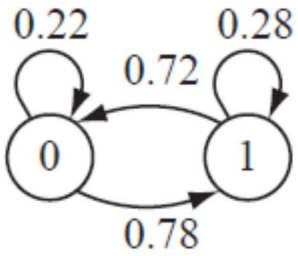
```
from math import exp
import numpy as np


def grad(x):  # f'(x)
    return 2*exp(2*x)+6*exp(x)*x-14*exp(x)+20*x-60


def cost(x):  # f(x)
    return (exp(x)+3*x-10)**2+x**2


def GD_momentum(x_init, gamma=0.1, alpha=0.9, N=1000, esilon=1e-5):
    theta = [x_init]
    v_old = np.zeros_like(x_init)
    for it in range(N):
        v_new = alpha*v_old + gamma*grad(theta[-1])
        theta_new = theta[-1] - v_new
        if np.abs(grad(theta_new)) < esilon:
            break
        theta.append(theta_new)
        v_old = v_new
    return theta, it


if __name__ == '__main__':
    x, it = GD_momentum(1.5, 0.1, alpha=0.9)
    print("Momentum_Solution : ")
    print("x = ", x[-1])
    print("cost = ", cost(x[-1]))
    print("After", it, "iterations")
```

Trả lời: Dán kết quả thực thi với điểm khởi $x = 1.50000$, tham số học học (*learning rate*) $\gamma = 0.1$, hệ số động lượng (*momentum coefficient*) là $\alpha = 0.9$, số bước lặp $N = 1000$ và sai số $\varepsilon = 10^{-5}$:

- Đối với bài này thì x sau mỗi bước lặp lớn dẫn đến hàm tính đạo hàm bị tràn số.
- Em thử tính f'(x) = 0 thì nó vô nghiệm.

```
PS D:\CODE\Python\CS-Math> & "C:/Program Files/Python39/python.exe" d:/CODE/Python/CS-Math/4.ToiU
Traceback (most recent call last):
  File "d:\CODE\Python\CS-Math\4.ToiUu\GradientDescent\3.Momentum.py", line 27, in <module>
    x, it = GD_momentum(1.5, gamma=0.1, alpha=0.9)
  File "d:\CODE\Python\CS-Math\4.ToiUu\GradientDescent\3.Momentum.py", line 19, in GD_momentum
    if np.abs(grad(theta_new)) < esilon:
  File "d:\CODE\Python\CS-Math\4.ToiUu\GradientDescent\3.Momentum.py", line 6, in grad
    return 2*exp(2*x)+6*exp(x)*x-14*exp(x)+20*x-60
OverflowError: math range error
PS D:\CODE\Python\CS-Math>
```

Câu 5 (1 điểm): Một hệ thống có chế độ làm việc ở mỗi giai đoạn vận hành chỉ với hai trạng thái 0 và 1. Chế độ làm việc của hệ thống này được mô tả bằng chuỗi Markov như hình vẽ.



a) (0.5) Xác định ma trận chuyển đổi trạng thái **P** của hệ.

Trả lời: dán kết quả vào bên dưới:

Quy ước: Hàng i , cột j biểu thị cho xác suất chuyển từ trạng thái i sang trạng thái j (i,j = 0..1)

$$P = \begin{bmatrix} 0.22 & 0.78 \\ 0.72 & 0.28 \end{bmatrix}$$

b) (0.5) Tìm xác suất (lớn nhất) khi hệ thống vẫn làm việc ở trạng thái **0** sau hai giai đoạn vận hành biết rằng hệ thống bắt đầu làm việc ở trạng thái **0**.

Trả lời: Dán kết quả tính toán vào bên dưới: %)

Sau 2 giai đoạn thì :

$$P^2 = \begin{bmatrix} 0.61 & 0.39 \\ 0.36 & 0.64 \end{bmatrix}$$

Vậy xác suất lớn nhất cần tìm là 0.61 tương ứng với 61%

Đà Nẵng, ngày 01 tháng 12 năm 2021

GIẢNG VIÊN BIÊN SOẠN ĐỀ THI

KHOA CÔNG NGHỆ THÔNG TIN

(đã duyệt)