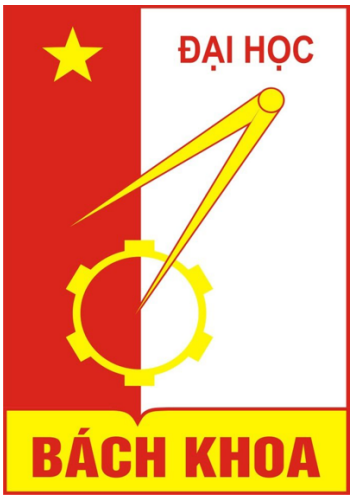
Trường Đại học Bách khoa Hà Nội
Viện Toán ứng dụng và Tin học



Báo cáo bộ môn phương pháp số
Phương pháp nội suy ngược

Giảng viên hướng dẫn	TS. Hà Thị Ngọc Yến	
Sinh viên thực hiện	Trần Xuân Hiếu	20182354
	Nguyễn Thị Xuân Hồng	20185364

Hà Nội, ngày 05 tháng 01 năm 2020

Mục lục

1. Giới thiệu bài toán

Ta xét đến bài toán: Tìm giá trị gần đúng của hàm f(x) tại điểm $\bar{x}$ nào đó không có trong bảng số. Bây giờ, ta thử xét bài toán ngược lại, nghĩa là từ bảng số đã cho trong dạng y=f(x) và một giá trị $\bar{y}$ cho trước trong khoảng giá trị đã cho trong bảng số, tìm giá trị $\bar{x}$ tương ứng. Hay có thể phát biểu là giải bài toán f(x)=y với $y = \bar{y} \in (y_0, y_n)$ là giá trị cho trước, và hàm y=f(x) được cho dưới dạng bảng số.

2. Ý tưởng giải quyết bài toán

2.1. Đưa bài toán giải phương trình $f(x) = \bar{y}$ về giải phương trình $f(x) - \bar{y} = 0$ hay $g(x) = 0, g(x) = f(x) - \bar{y}$

Để giải phương trình g(x)=0 có thể sử dụng phương pháp đã học ở môn giải tích số: phương pháp dây cung, phương pháp tiếp tuyến, phương pháp chia đôi,… để giải.

Tuy nhiên, hàm g(x) được cho dưới dạng bảng số, do đó, ta khó có thể đánh giá chính xác để tìm khoảng cách ly nghiệm của phương trình g(x)=0. Trước hết ta phải tìm được những khoảng mà bảng số đơn điệu, nếu giá trị $\bar{y}$ nằm trong khoảng đó, ta sẽ tìm nghiệm gần đúng của $f(x) = \bar{y}$ trên khoảng đó.

2.2. Đưa về bài toán tìm hàm ngược

Ta có thể thấy ở đây cũng có 1 bảng số gồm các giá trị của x và giá trị của y=f(x) tương ứng, ta đổi ngược lại, tức là, coi đây là bảng số gồm các giá trị của y và các giá trị của $x = \varphi(y)$ và sử dụng các phương pháp xấp xỉ hàm số đã học trong các bài trước để xấp xỉ hàm $\varphi(y)$

Như ta đã biết, hàm số $y = f(x)$ là một ánh xạ đơn ánh từ $\mathbb{R} \longrightarrow \mathbb{R}$ nhưng hàm ngược của nó thì không đơn ánh, do đó để có thể sử dụng các phương pháp xấp xỉ hàm số đã biết ta cần xem xét bảng số đã cho có đủ điều kiện để tồn tại hàm ngược hay không?

Hàm ngược $x = \varphi(y)$ (nếu có) là đơn ánh, do đó hàm $y = f(x)$ phải đơn điệu trên khoảng đang xét để tìm hàm ngược.

Như vậy, nếu bảng số đã cho là đơn điệu thì ta có thể áp dụng ngay các công thức xấp xỉ hàm số đã biết, còn nếu không, ta sẽ phải tìm cách xử lí bảng số này, trong trường hợp này, ta sẽ tìm các khoảng đơn điệu trên bảng số và tách ra để xấp xỉ trên từng khoảng đơn điệu đó. Tuy nhiên, nếu khoảng đơn điệu đó chỉ có 2 mốc thì việc xấp xỉ hàm số mà chỉ có 2 mốc

sẽ cho sai lệch khá lớn và không đáng tin cậy. Do đó nếu không cần thiết ta có thể ngắt bớt 1 mốc nào đó.

3. Giải quyết bài toán bằng cách xấp xỉ hàm số

3.1. Sử dụng đa thức nội suy Lagrange

3.1.1. Công thức xấp xỉ

Coi $y_i(i = \overline{0,n})$ là các mốc nội suy,nói chung, thông thường thì các mốc $y_i(i = \overline{0,n})$ không cách đều nhau. Ta sẽ sử dụng đã thức nội suy Lagrange để xấp xỉ hàm $x = \varphi(y) \approx P_n(y)$. Để tìm được đa thức $P_n(y)$ sinh ra từ bảng số

$x_i = \varphi(y_i)(i = \overline{0,n}), a \equiv y_0 < y_1 < \cdots < y_n \equiv b$,thỏa mãn điều kiện: $x_i = P_n(y_i)(i = \overline{0,n})$, ta sử dụng công thức:

$P_n(y) = \sum_{j=0}^{n} L_j(y)x_i \ (i = \overline{0,n}), L_j(y)$ là đa thức cơ sở bậc n và

$$L_j(y_i) = \begin{cases} 0 \ với \ i \neq j \\ 1 \ với \ i = j \end{cases}$$

3.1.2. Công thức đánh giá sai số

Với giá trị $\bar{y}$ cho trước ($\bar{y} \neq y_i, i = \overline{0,n}$), đặt $R_n(\bar{y}) = \varphi(\bar{y}) - P_n(\bar{y})$ là sai số tại điểm $\bar{y}$, ta có công thức đánh giá sai số là:

$$|R_n(\bar{y})| \leq \frac{M}{(n+1)!}|\omega_{n+1}(\bar{y})|$$

Với:

$$M = \sup_{y \in [a,b]} \left|\varphi^{(n+1)}(y)\right|, \ \omega_{n+1}(y) = (x - x_0)(x - x_1) \ldots (x - x_n) = \prod_{k=0}^{n}(x - x_k)$$

3.1.3. Thuật toán

1. Read n, y
2. for i = 1 to (n + 1) in steps of 1 do Read x[i], f[i] end for
3. sum ← 0
4. for i = 1 to (n + 1) in steps of 1 do
5. values ←1
6. for j = 1 to (n + 1) in steps of 1 do
7. If (j ≠ i) then

Values ← $values * (y - y_j)/(y_i - y_j)$
End for

8. sum ← sum + x[i] * values
9. Write y, sum
10. Stop

3.2. Sử dụng đa thức nội suy Newton

3.2.1. Công thức xấp xỉ hàm số

Xấp xỉ hàm $x = \varphi(y) \approx P_n(y)$ theo công thức nội suy Newton:

Newton tiến:

$P_n(y) = x_0 + (y - y_0)\varphi[y_0, y_1] + \cdots + (y - y_0)(y - y_1) \dots (y - y_{n-1})\varphi[y_0, y_1, \dots, y_n]$(1)

Newton lùi:

$P_n(y) = x_n + (y - y_n)\varphi[y_n, y_{n-1}] + \cdots + (y - y_n)(y - y_{n-1}) \dots (y - y_1)\varphi[y_n, y_{n-1}, \dots, y_0]$(2)

3.2.2. Công thức đánh giá sai số

Ta có $R_n(y) = \varphi(y) - P_n(y) = (y - y_0)(y - y_1) \dots (y - y_n)\varphi[y, y_0, y_1, \dots, y_n]$

$= \omega_{n+1}(y)\varphi[y, y_0, y_1, \dots, y_n]$

là công thức đánh giá sai số chung cho cả công thức (1) và (2).

3.2.3. Thuật toán

Sử dụng đa thức nội suy Newton rồi thực hiện lặp:

Ta cho thêm 2 biến esp, max:

- Esp: độ lệch giữa hai nghiệm trong quá tình lặp đủ nhỏ để kết luận nghiệm
- Max: số vòng lặp tối đa thực hiện để tránh trường hợp không hội tụ về nghiệm (lặp vô hạn)

Thuật toán:

1. Read x[i], y[i]
   Input esp, Max
2. Tìm khoảng phân ly nghiệm
3. $y = P_n(x)$
4. $x = g(x)$
5. Lặp $x_{0i} = x_i + \dfrac{\bar{y} - y_i}{f[x_{i+1}, x_i]}$

5. x
6. End

4. Lập trình phương pháp

4.1. Kiểm tra và xử lý dữ liệu đầu vào

- Chương trình sắp xếp lại tập dữ liệu

```
def dataSort(data):
    lenght = len(data)
    for i in range(lenght-1):
        for j in range(i, lenght):
            if data[i][0] > data[j][0]:
                data[i][0], data[j][0] = data[j][0], data[i][0]
                data[i][1], data[j][1] = data[j][1], data[i][1]
    return data
```

- Chương trình kiểm tra các điểm dữ liệu có thỏa mãn điều kiện nội suy ngược hay không:

```
def InterpolationConditions(data):
    lenght = len(data)
    for i in range(lenght-1):
        for j in range(i+1,lenght):
            if data[i][0] == data[j][0] and data[i][1] == data[j][1]:
                return False
            if data[i][0] == data[j][0]:
                return False
    return True
```

- Chương trình kiểm tra các mốc dữ liệu là cách đều hay ngẫu nhiên:

```
def x_equidistant(data, index):
    i = index[0]
    while i < (index[1]-1):
        delta_x1 = data[i][0] - data[i+1][0]
        delta_x2 = data[i+1][0] - data[i+2][0]
        i += 1
        if abs(delta_x1 - delta_x2) >= 10e-5:
            print("x khong cach deu")
            return False
    print("x cach deu")
    return True
```

- Chương trình cắt mảng dữ liệu thành từng khoảng đơn điệu:

```
def FindMonotonousInterval(data):
    indexs = []
    lenght = len(data)
```

```
    index_head = 0
    index_tail = 1
    for i in range(lenght-2):
        delta_y1 = data[i][1] - data[i+1][1]
        delta_y2 = data[i+1][1] - data[i+2][1]
        if delta_y1 * delta_y2 > 0:
            index_tail += 1
        else:
            indexs.append([index_head, index_tail])
            index_head = index_tail
            index_tail += 1
    indexs.append([index_head, index_tail])
    return np.array(indexs)
```

- Qua quá trình làm bài và phân tích, nhóm em nhận thấy với số lượng mốc nội suy là 4 mốc thì sẽ đạt được sự tối ưu về sai số, khối lượng tính toán và tốc độ xử lí. Đây là chương trình chọn ra số mốc nội xung quanh điểm nội suy, trả về tối đa 4 mốc nội suy:

```
# 4 moc noi suy
def choosePoint(data, index, y):
    if data[index[0]][1] < data[index[0]+1][1]:
        data = -data
        y = -y
    flat_y = index[0]
    while y < data[flat_y][1]:
        flat_y += 1

    if flat_y == 1:
        index[1] = index[0] + 3
    elif flat_y == len(data)-1:
        index[0] = index[1] - 3
    else:
        index[0] = flat_y - 2
        index[1] = flat_y + 1
    return index
```

4.2. Lựa chọn phương pháp nội suy phù hợp với từng khoảng đơn điệu

- Chương trình lựa chọn phương pháp nội suy:

```
def ConsiderApprox(data, indexs, y):
    lenght = len(indexs) - 1
    Newton = []
    Lagrange = []
    while lenght >= 0:
        if data[indexs[lenght][0]][1] > data[indexs[lenght][1]][1]:
            # Xét trên đơn điệu giảm
            if (data[indexs[lenght][0]][1]<y or y<data[indexs[lenght][1]][1]):
```

```
                indexs = np.delete(indexs, indexs[lenght])
            else:
                print("Mang: {} => Su dung phuong phap: ".format(indexs[lenght
]), end='')
                thu = Lagrange_Newton(-data, indexs[lenght], y)
                x_equidistant(data, indexs[lenght])
                if thu == True :
                    Newton.append(indexs[lenght])
                else:
                    Lagrange.append(indexs[lenght])
        else:
            # xét trên đơn điệu tăng
            if data[indexs[lenght][0]][1]>y or y>data[indexs[lenght][1]][1]:
                indexs = np.delete(indexs,indexs[lenght])
            else:
                print("Mang: {} => Su dung phuong phap: ".format(indexs[lenght
]), end='')
                thu = Lagrange_Newton(data, indexs[lenght], y)
                x_equidistant(data, indexs[lenght])
                if thu == True :
                    Newton.append(indexs[lenght])
                else:
                    Lagrange.append(indexs[lenght])
        lenght -= 1
    return np.array(Newton), np.array(Lagrange)
# Đơn điệu tăng
def Lagrange_Newton(data, index, y):
    delta_y = []
    flat =  True
    if index[1] - index[0] == 1:
        print("Newton voi khoang 2 moc")
    else:
        if data[index[0]+1][1] > y and data[index[0]][1] < y:
            print("Newton Tien")
        elif data[index[1]-1][1] < y and data[index[1]][1] > y:
            print("Newton Lui")
        else:
            for i in range(index[1]-index[0]):
                d_y = abs(data[index[0]+i][1] - data[index[0]+i+1][1])
                d_x = abs(data[index[0]+i][0] - data[index[0]+i+1][0])
                delta = d_y/d_x
                delta_y.append(delta)

            ty_hieu_1 = delta_y[0]/delta_y[-1]

            if (ty_hieu_1 <= 1.5 and ty_hieu_1 >= 0.67):
                print('Larange')
                flat = False
```

```
            else:
                print('Newton')
    return flat
```

4.3. Nội suy ngược bằng phương pháp nội suy Newton

- Chương trình xây dựng bảng sai phân:

```
def __Bangsaiphan(self):
        m = len(self.yData)
        a = np.zeros((m,m))
        for k in range(m):
            a[k][0] = self.yData[k]

        for i in range(1,m):
            for j in range(1,i+1):
                a[i][j] = a[i][j-1] - a[i-1][j-1]
        return a
```

- Chương trình giải phương trình bằng phương pháp lặp đơn:

```
def __g0(self, t0, t, heso):
        t1 = t0 - heso[2]/2*t*(t-1) - heso[3]/(2*3)*t*(t-1)*(t-2)
        return t1
    def __g1(self, t0, t, heso):
        t1 = t0 - heso[2]/2*t*(t+1) - heso[3]/(2*3)*t*(t+1)*(t+2)
        return t1
    def __g2(self, t0, t, heso):
        t1 = t0 + t*(t-1)*(heso[3]-heso[1])/4
        return t1


def __Lapdon(self ,t0, heso, pp, x0):
        esp = 10e-9
        max = 1000
        delta = 100
        flag = 1
        i = 1
        t = t0 # xap xi dau
        x = t*self.h + x0
        # print(t0)
        while (delta > esp):
            t1 = t
            x1 = x
            if (pp == 0):
                t =  self.__g0(t0, t1, heso)
            elif (pp == 1):
                t = self.__g1(t0, t1, heso)
            elif (pp == 2):
                t = self.__g2(t0, t1, heso)
```

```
            x = t*self.h + x0
            print("Lan lap thu", i, "co nghiem la", x)
            delta = math.fabs(x-x1)
            # x1 = x
            i += 1
            if (i > max):
                flag = 0
                break
        if (flag == 1):
            print("Ket qua:\t", x,"\n")
        else:
            print("Khong hoi tu hoac hoi tu cham!")
        return x
```

- Chương trình nội suy bằng đa thức nội suy Newton:

```
def Newtontien(self):
        a = self.__Bangsaiphan()
        pp = 0 # xác định công thức lặp
        heso = []
        for i in range(4):
            heso.append(a[i][i]/a[1][1])

        t0 = (self.y-a[0][0])/a[1][1]
        x = self.__Lapdon(t0, heso, pp, self.xData[0])
        return x

    def Newtonlui(self):
        a = self.__Bangsaiphan()
        pp = 1  # xác định công thức lặp
        heso = []
        for i in range(4):
            heso.append(a[3][i]/a[3][1])

        t0 = (self.y-a[3][0])/a[3][1]
        x = self.__Lapdon(t0, heso, pp, self.xData[3])
        return x

    def Betxen(self):
        a = self.__Bangsaiphan()
        pp = 2
        heso = []
        for i in range(4):
            heso.append(a[i][1]/a[2][1])

        t0 = (self.y - a[1][0])/a[2][1]
        x = self.__Lapdon(t0, heso, pp, self.xData[1])
        return x
```

4.4. Nội suy ngược bằng phương pháp nội suy Lagrange

- Chương trình phương pháp nội suy ngược bằng đa thức Lagrange:

```
def multiPoly(A, B):
    prod = [0]*(len(A) + len(B) - 1)
    for i in range(0, len(A), 1):
        for j in range(0, len(B), 1):
            prod[i + j] += A[i] * B[j]
    return prod

def Lagrange(x, y, F, L):
    L = [[0] * len(x)] * len(x)
    poly = [[0, 0]]*len(x)  # mảng cho các đa thức đơn vị
    temp = [1]*len(x)
    tempPoly = [[]]*len(x)  # đa thức cơ sở ( 1 + 0*x)
    for i in range(len(x)):
        poly[i] = [-x[i], 1]
        # hàm tạo các đa thức nhỏ từ mảng x
    for i in range(len(x)):
        if i == 0:
            tempPoly[i] = poly[1]
        else:
            tempPoly[i] = poly[0]
        for j in range(len(x)):
            if (j != 0 and j != i and i != 0) or (i == 0 and (j > 1)):
                # if i != j:
                tempPoly[i] = multiPoly(tempPoly[i], poly[j])
                # tinh tu so L[i]
            if j != i:
                temp[i] *= (-poly[i][0] + poly[j][0])
                # tính mẫu số L[i]
    for i in range(len(tempPoly)):
        L[i] = [tempPoly[i][j]/temp[i] for j in range(len(tempPoly[i]))]
        # tính các đa thức Lagrange cơ bản

    for i in range(len(x)):
        for j in range(len(x)):
            F[i] += L[j][i] * y[j]
            # tính đa thức Lagrange
    return L, F

def printPoly(F):
    print("F = ", end='')
    for i in range(len(F)):
        if i != len(F)-1:
            print(f'{F[i]}*x^{i} + ', end='')
        else:
```

```
        print(f'{F[i]}*x^{i}  ', end='')
```

5. Các ví dụ minh hoạ

5.1. Nội suy ngược tại một điểm nằm trên một tập dữ liệu cách đều

Xét hàm số:

$$y = \frac{1}{x} + \frac{\sin(x)}{2}$$

- Ta có bảng dữ liệu (x,y) của hàm số như sau:

X	Y
-3	-0.4038933
-2.9	-0.4644523
-2.8	-0.5246369
-2.7	-0.5840603
-2.6	-0.6423661
-2.5	-0.6992361
-2.4	-0.7543983
-2.3	-0.8076352
-2.2	-0.8587937
-2.1	-0.9077952
-2	-0.9546487
-1.9	-0.9994658
-1.8	-1.0424794
-1.7	-1.0840677
-1.6	-1.1247868
-1.5	-1.1654142
-1.4	-1.2070106
-1.3	-1.2510099
-1.2	-1.2993529
-1.1	-1.3546946
-1	-1.4207355
-0.9	-1.5027746
-0.8	-1.608678
-0.7	-1.7506803
-0.6	-1.9489879
-0.5	-2.2397128
-0.4	-2.6947092
-0.3	-3.4810934
-0.2	-5.0993347
-0.1	-10.049917

- Thực hiện nội suy ngược tại vị trí $y_0 = -6$. Xét thấy -6 là một điểm nằm trong khoảng sát điểm cuối cùng trong tập hợp mốc dữ liệu, nên sẽ được xử lí bằng phương pháp Newton lùi:

```
nhap gia tri y0: -6
Mang: [ 0 29] => Su dung phuong phap: Newton
x cach deu
Newton:  [[ 0 29]]
Lagrange:  []

Noi suy Newton doan [ 0 29]
Cac moc noi suy:
 [[ -0.4         -0.3         -0.2         -0.1        ]
 [ -2.69470917  -3.48109344  -5.09933467 -10.04991671]]
Dung Newton lui:
Lan lap thu 1 co nghiem la -0.17531694267936093
Lan lap thu 2 co nghiem la -0.173598781554206
Lan lap thu 3 co nghiem la -0.17319960074162277
Lan lap thu 4 co nghiem la -0.1731102577310487
Lan lap thu 5 co nghiem la -0.17309043766824206
Lan lap thu 6 co nghiem la -0.17308604948897127
Lan lap thu 7 co nghiem la -0.17308507837194329
Lan lap thu 8 co nghiem la -0.17308486348193158
Lan lap thu 9 co nghiem la -0.17308481593182734
Lan lap thu 10 co nghiem la -0.17308480541015836
Lan lap thu 11 co nghiem la -0.17308480308197421
Ket qua:         -0.17308480308197421

Ket qua x0:  [-0.17308480308197421]
```

Thử lại: với x = -0.17308480308197421 thì y = -5.86362567088365

=>Ta có thể thấy sai số này rất lớn vì ta nội suy tại một điểm rất gần với điểm ngoặt của hàm. Tại vùng này, tốc độ biến thiên của x và y chênh lệch nhau rất lớn nên sai số của phép nội suy cũng rất lớn.

- Thực hiện nội suy ngược tại vị trí $y_0 = -1.2$, xét thấy giá trị -1,2 rất gần với một giá trị nằm ở khoảng giữa trong tập hợp mốc dữ liệu, nên sẽ được xử lí bằng phương pháp nội suy Betxen (Newton trung tâm):

```
nhap gia tri y0: -1.2
Mang: [ 0 29] => Su dung phuong phap: Newton
x cach deu
Newton:  [[ 0 29]]
Lagrange:  []

Noi suy Newton doan [ 0 29]
Cac moc noi suy:
 [[-1.6        -1.5        -1.4        -1.3       ]
 [-1.1247868  -1.16541416 -1.20701058 -1.25100986]]
Dung Newton trung tam:
Lan lap thu 1 co nghiem la -1.4171377941654548
Lan lap thu 2 co nghiem la -1.4171415931128826
Lan lap thu 3 co nghiem la -1.417141643710114
Lan lap thu 4 co nghiem la -1.4171416443839664
Ket qua:         -1.4171416443839664

Ket qua x0:  [-1.4171416443839664]
```

Thử lại: với x = -1.4171416443839664 thì y = -1.19975492860797.
=>Ta có thể thấy sai số đã nhỏ có thể chấp nhận được. Lí do là vì điểm nội suy nằm xa điểm ngoặt của đồ thị hàm số, tại khoảng này tốc độ biến thiên của x và y không chênh nhau quá nhiều nên sai số đã giảm nhiều.

- Thực hiện nội suy tại điểm y_0 = -0.41. Xét thấy -0.41 là một điểm nằm trong khoảng sát điểm đầu trong tập hợp mốc dữ liệu, nên sẽ được xử lí bằng phương pháp Newton tiến:

```
nhap gia tri y0: -0.41
Mang: [ 0 29] => Su dung phuong phap: Newton
x cach deu
Newton:  [[ 0 29]]
Lagrange:  []

Noi suy Newton doan [ 0 29]
Cac moc noi suy:
 [[-3.         -2.9        -2.8        -2.7       ]
 [-0.40389334 -0.46445225 -0.52463693 -0.58406031]]
Dung Newton tien:
Lan lap thu 1 co nghiem la -2.9899258338280226
Lan lap thu 2 co nghiem la -2.989925824657599
Ket qua:         -2.989925824657599

Ket qua x0:  [-2.989925824657599]
```

Thử lại: với x = -2.989925824657599 thì y = -0.409999475844878.

- Tương tự với trường hợp bên trên, điểm nội suy nằm trên khoảng mà x với y có tốc độ biến thiên như nhau nên sai số khá nhỏ.

Kết luận: Có thể dễ dàng thấy được rằng tập dữ liệu có rất nhiều điểm nội suy và các điểm nội suy cách đều nhau nên bài toán sẽ được ưu tiên giải bằng đa thức nội suy Newton.

5.2. Nội suy trên tập dữ liệu bất kì

Xét hàm số:
$y = x^2 – 4x + 3.75$
có giá trị cho theo bảng sau:

X	Y
0.1	3.36
0.19	3.0261
0.28	2.7084
0.42	2.2464
0.53	1.9109
0.67	1.5189
0.7	1.44
0.8	1.19
0.91	0.9381
1.03	0.6909
1.09	0.5781
1.22	0.3584
1.29	0.2541
1.43	0.0749
1.5	0
1.6	-0.09
1.7	-0.16
1.8	-0.21
1.9	-0.24
2	-0.25
2.1	-0.24
2.2	-0.21
2.3	-0.16
2.4	-0.09
2.5	0
2.6	0.11
2.7	0.24

2.8	0.39
2.9	0.56
3	0.75

Xét thấy tập dữ liệu có 2 khoảng đơn điệu. Tiến hành nội suy tại điểm $y_0 = 0.4$, thu được kết quả như sau:

```
nhap gia tri y0: 0.4
Mang: [19 29] => Su dung phuong phap: Newton
x cach deu
Mang: [ 0 19] => Su dung phuong phap: Newton
x khong cach deu
Newton:  [[19 29]
 [ 0 19]]
Lagrange:  []

Noi suy Newton doan [19 29]
Cac moc noi suy:
 [[2.7  2.8  2.9  3.  ]
 [0.24 0.39 0.56 0.75]]
Dung Newton lui:
Lan lap thu 1 co nghiem la 2.8076250182242317
Lan lap thu 2 co nghiem la 2.806436507665448
Lan lap thu 3 co nghiem la 2.806257644036162
Lan lap thu 4 co nghiem la 2.806230597422088
Lan lap thu 5 co nghiem la 2.806226504674943
Lan lap thu 6 co nghiem la 2.806225885285272
Lan lap thu 7 co nghiem la 2.8062257915463147
Lan lap thu 8 co nghiem la 2.8062257773597463
Lan lap thu 9 co nghiem la 2.8062257752127326
Ket qua:         2.8062257752127326


Noi suy Newton doan [ 0 19]
Cac moc noi suy:
 [[1.03   1.09   1.22   1.29  ]
 [0.6909 0.5781 0.3584 0.2541]]
Dung Newton trung tam:
Lan lap thu 1 co nghiem la 1.1387281326505296
Lan lap thu 2 co nghiem la 1.1387275960571823
Lan lap thu 3 co nghiem la 1.1387275992971537
Ket qua:         1.1387275992971537

Ket qua x0:  [2.8062257752127326, 1.1387275992971537]
```

Tiến hành thử lại với:

- $x_1 = 2.806225775212736$ thì $y_1 = 0.400000000617368$
- $x_2 = 1.1387275992971537$ thì $y_2 = 0.491790148212451$

Dễ dàng nhận thấy giá trị nội suy của x_2 mang lại sai số quá lớn, nguyên nhân là vì x_2 nằm trong khoảng không cách đều của tập dữ liệu.
Trong trường hợp các mốc dữ liệu không cách đều, ta tiến hành nội suy bằng phương pháp đa thức nội suy Lagrange với cùng số mốc nội suy như khi sử dụng nội suy bằng phương pháp Newton, đó là dung 4 mốc quanh điểm nội suy.
Ta thu được kết quả như sau:

```
PS C:\Users\HIEU.TX185354\Desktop\PPS_Hồng> python -u "c:\Users\HIEU.TX185354\Desk
top\PPS_H?ng\LarangeReverseInterpolation-2.py"
[1.03, 1.09, 1.22, 1.29]
[0.6909, 0.5781, 0.3584, 0.2541]
F = 1.4907304337338427*x^0 + -0.8878232099989987*x^1 + 0.4230906683405351*x^2 + -0
.14946404571506378*x^3
 kết quả
1.193729957742965
PS C:\Users\HIEU.TX185354\Desktop\PPS_Hồng> python -u "c:\Users\HIEU.TX185354\Desk
top\PPS_H?ng\LarangeReverseInterpolation-2.py"
[2.7, 2.8, 2.9, 3.0]
[0.24, 0.39, 0.56, 0.75]
F = 2.509545923632608*x^0 + 0.885061919504686*x^1 + -0.4156633413599309*x^2 + 0.14
333218667586323*x^3
 kết quả
2.8062378167641486
PS C:\Users\HIEU.TX185354\Desktop\PPS_Hồng>
```

Tiến hành thử lại với:

- $x_1 = 2.8062378167641486$ thì $y_1 = 0.400019417180607$
- $x_2 = 1.193729957742965$ thì $y_2 = 0.400071381041169$

Dễ dàng nhận thấy kết quả nội suy của đa thức Lagrange cho kết quả với sai số thấp hơn đa thức nội suy Newton khi các mốc nội suy không cách đều nhau. Tuy nhiên trong khoảng dữ liệu mốc cách đều thì nội suy Newton vẫn cho kết quả tốt hơn (so sánh y_1 ở hai phương pháp).

6. Nhận xét và ứng dụng

- Với mốc nội suy x_i cách đều thì phương pháp nội suy bằng đa thức Newton tốt hơn hẳn so với nội suy bằng đa thức Lagrange
  nhất. Nguyên nhân là do khi đó mốc yi có thể biến đổi bất thường.
- Với mốc nội suy x_i không cách đều thì nội suy bằng đa thức Lagrange có thể tốt hơn
- Nhược điểm của phương pháp nội suy bằng đa thức Newton là khó đánh giá điều kiện hội tự nghiệm và sai số. Sai số của phương pháp này bao gồm sai số tính toán, sai số của phép nội suy, sai số của phép lặp.

- Ứng dụng chính của hai phương pháp đó là giải phương trình, phán đoán kết quả dựa trên các kết quả sẵn có, xây dựng mối quan hệ giữa các dữ liệu trên khoảng quan sát được.

Tài liệu tham khảo

[1] Phạm Kỳ Anh, Giải tích số, NXB Đại học Quốc Gia Hà Nội (1996) 200 trang.
[2] Lê Trọng Vinh, Giải tích số, NXB Khoa học và Kỹ Thuật (2007) 240 trang.
[3] Numerical Methods in Engineering with Python 3
[4] Newton Methodsfor Nonlinear Problems