**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**VIỆN TOÁN ỨNG DỤNG VÀ TIN HỌC**

-----🙞🙜🕮🙞🙜-----

****

*Báo cáo môn học: Kỹ thuật lập trình*

**Đề tài số 04:**

Xây dựng đa thức nội suy ứng với một bảng số cho trước theo

công thức Newton

\*\*\*

**Giảng viên hướng dẫn: TS. Nguyễn Thị Thanh Huyền**

**Nhóm thực hiện: Nhóm 04**

1.Nguyễn Hữu Thuật MSSV: 20185410

2. Lại Tiến Long MSSV: 20185376

*Hà Nội, 2020*

Bước 0: Tổng quan về Menu

Case 0

Case 8

Case 7

Case 6

Case 5

Case 4

Case 3

Case 2

Case 1

MENU

newton LuiMoc CachDeu ba bangSaiPhanThuan ngSaiPhanThuan

newton TienMoc CachDeu ba bangSaiPhanThuan ngSaiPhanThuan

newton LuiMoc BatKy ba bangSaiPhanThuan ngSaiPhanThuan

Thoát ba bangSaiPhanThuan ngSaiPhanThuan

newton TienMocBatKy ba bangSaiPhanThuan ngSaiPhanThuan

newton LuiMocBatKy ba bangSaiPhanThuan ngSaiPhanThuan

bangSaiPhan ThuanhanThuan ba bangSaiPhanThuan ngSaiPhanThuan

bangtyHieu ThuanhanThuan ba bangSaiPhanThuan ngSaiPhanThuan

Nhap

Bước 1: Mô tả MENU

* In các lựa chọn
* In dòng khuyến nghị thứ tự nhập: 1-2-4-8; 1-3-6-8; 1-5-8, 1-7-8
* CASE 0: Xóa file đã tồn tại. Nếu không xóa các dữ liệu sẽ được ghi tiếp ở cuối file cũ.
* CASE 1: Nhập và kiểm tra các bộ giá trị
* CASE 2: Vẽ bảng tỷ hiệu thuận

Kiểm tra xem người dùng đã nhập bộ giá trị chưa

+ nếu chưa : thông báo yêu cầu nhập, trở về menu

+ nếu đã nhập: Tiến hành vẽ bảng tỷ hiệu thuận => xuất file.

* CASE 3: Vẽ bảng sai phân thuận

Kiểm tra xem người dùng đã nhập bộ giá trị chưa

+ nếu chưa : thông báo yêu cầu nhập, trở về menu

+ nếu đã nhập: Tiến hành kiểm tra độ lệch các mốc (h)

h = |x[1] – x[0]|

for i = 1 to n -2:

if |x[i+1] – x[i]| == h

Tiến hành vẽ bảng sai phân

else: gửi thông báo không vẽ được, gợi ý chọn vẽ bảng tỷ hiểu. Trở về MENU

* CASE 4: Viết đa thức Newton tiến mốc bất kỳ

Kiểm tra xem đã có bảng tỷ hiệu thuận

+ nếu chưa : thông báo yêu cầu vẽ bảng, trở về menu

+ nếu đã nhập: Tiến hành viết đa thức Newton tiến mốc bất kỳ

CASE 5: Viết đa thức Newton lùi mốc bất kỳ

Kiểm tra xem người dùng đã nhập bộ giá trị chưa

+ nếu chưa : thông báo yêu cầu nhập, trở về menu

+ nếu đã nhập:

* Vẽ bảng tỷ hiệu nghịch
* Tiến hành viết đa thức Newton lùi mốc bất kỳ
* CASE 6: Viết đa thức Newton tiến mốc cách đều

Kiểm tra xem đã có bảng sai phân thuận

+ nếu chưa : thông báo yêu cầu vẽ bảng, trở về menu

+ nếu đã nhập: Tiến hành viết đa thức Newton tiến mốc cách đều

* CASE 7: Viết đa thức Newton lùi mốc cách đều

Kiểm tra xem người dùng đã nhập bộ giá trị chưa

+ nếu chưa : thông báo yêu cầu nhập, trở về menu

+ nếu đã nhập:

* Tiến hành kiểm tra độ lệch các mốc (h)

h = |x[1] – x[0]|

for i = 1 to n -2:

if x[i+1] – x[i] == h

+ Tiến hành lập bảng sai phân nghịch

+ Tiến hành viết đa thức Newton lùi mốc bất kỳ

else: gửi thông báo không lập được, gợi ý chọn viết đa thức Newton lùi mốc bất kỳ. Trở về MENU

* CASE 8: Tính giá trị của đa thức tại điểm c nhập từ bàn phím

kiểm tra xem đã có đa thức Newton hay chưa:

* Nếu chưa: yêu cầu thực hiện lập đa thức Newton
* Nếu đã có:

\* Nhập c

\* Tiến hành chia các TH để đưa vào hàm tính (hoocne)

+ đa thức Newton tiến mốc bất kỳ => hệ số Newton tiến mốc bất kỳ => hoocne

+ đa thức Newton lùi mốc bất kỳ => hệ số Newton lùi mốc bất kỳ => hoocne

+ đa thức Newton tiến mốc cách đều => hệ số Newton tiến mốc cách đều => hoocne

+ đa thức Newton tiến lùi cách đều => hệ số Newton lùi mốc cách đều => hoocne

Bước 2: Tổng quan các hàm trong chương trình

NEWTON

Tính giá trị tại 1 điểm

Công thức Newton

Bảng tỉ hiệu và sai phân

Sơ đồ Hoocne

Tìm hệ số đa thức

CT Newton lùi

CT Newton tiến

Bảng sai phân

Bảng tỉ hiệu

Bảng tỉ hiệu thuận

CT Newton tiến mốc cách đều

CT Newton tiến mốc bất kỳ

CT Newton lùi mốc cách đều

CT Newton lùi mốc bất kỳ

Bảng sai phân nghịch

Bảng sai phân thuận

Bảng tỉ hiệu nghịch

Tính giai thừa

Hệ số Newton lùi mốc bất kỳ

Hệ số Newton lùi mốc cách đều

Hệ số Newton tiến mốc bất kỳ

Hệ số Newton tiến mốc cách đều

In mảng

Nhân hệ số

Sao chép mảng

Bước 3:

* Dữ liệu vào: số các mốc (n) , các mốc điểm, điểm cần tính (c).
* Dữ liệu ra: giá trị đa thức tại điểm cần tính đó (P(c))
* Ý tưởng thuật toán:

Lập được đa thức Newton và dựa vào sơ đồ Hoocne để tính giá trị biểu thức tại c.

Hiển thị các bước tính toán ra File.

Bước 4:

Function Newton

* Dữ liệu vào: n, mảng x[] lưu mốc các điểm, mảng y[][] lưu ma trận trong quá trình tính giá trị các bảng, điểm cần tính (c).
* Dữ liệu ra: các bảng (tỷ hiệu, sai phân), các phương trình Newton( Newton tiến, Newton lùi), quá trình tính toán bằng lược đồ Hoocne. Hiển thị dưới dạng File.
* Phân loại dữ liệu đầu vào: Kiểm tra hiệu mốc các điểm để xem xem dữ liệu đầu vào phù hợp với dạng bảng và công thức nào.

Kiểm tra c để đưa gợi ý cho người dùng sử dụng dạng Newton nào hiệu quả hơn.

Bước 5: Hàm nhập và hiển thị bảng mốc các điểm

Function Nhap

* Dữ liệu vào: n, mảng x[], mảng y[][].
* Dữ liệu ra: bảng mốc các điểm.
* Ý tưởng: sử dụng mảng 1 chiều để lưu dãy mốc điểm x. Sử dụng mảng 2 chiều y để lưu các giá trị tương ứng với x và quá trình thao tác trong các bảng.

Bước 6: Lập bảng tỷ hiệu & sai phân

* Dữ liệu vào: 2 mảng x[], y[][].
* Dữ liệu ra: 1 trong 4 bảng: bảng tỷ hiệu thuận, bảng tỷ hiệu nghịch, bảng sai phân thuận, bảng sai phân nghịch.
* Ý tưởng: dựa vào dữ liệu đầu vào phân 4 bảng trên thành 2 TH theo x[] => bảng tỷ hiệu hoặc bảng sai phân. Cho người dùng tự chọn loại bảng muốn hiển thị (thuật toán có gợi ý nếu đã nhập c).

Bước 7:

Funtion bangSaiPhanThuan

* Dữ liệu vào: 2 mảng x[], y[].
* Dữ liệu ra: bảng sai phân thuận.
* Sử dụng công thức hiệu hữu hạn tiến tổng quát :



* Áp dụng vào chương trình bằng cách chạy 2 vòng for lồng nhau:

For i = 0 to n - 1:

For j = 0 to n – i:

y[j][i] = y[j + 1][i - 1] - y[j][i - 1]

Bước 8:

Funtion bangSaiPhanNghich

* Dữ liệu vào: 2 mảng x[], y[][].
* Dữ liệu ra: bảng sai phân nghịch.
* Sử dụng công thức hiệu hữu hạn tiến tổng quát :



* Áp dụng vào chương trình bằng cách chạy 2 vòng for lồng nhau :

For i = 1 to n -1 :

For j = n – 1 down to i – 1:

y[j][i] = y[j][i - 1] - y[j - 1][i - 1]

Bước 9:

Funtion bangTyHieuThuan

* Dữ liệu vào: 2 mảng x[], y[][].
* Dữ liệu ra: bảng tỷ hiệu thuận.
* Sử dụng công thức tổng quát tính tỷ hiệu cấp n :



* Áp dụng vào chương trình bằng cách chạy 2 vòng for lồng nhau :

For i =1 to n-1 :

For j = 0 to n – i:

y[j][i] = (y[j + 1][i - 1] - y[j][i - 1]) / (x[i + j] - x[j])

Bước 10:

Funtion bangTyHieuNghich

* Dữ liệu vào: 2 mảng x[], y[][],
* Dữ liệu ra: bảng tỷ hiệu nghịch.
* Sử dụng công thức tổng quát tính tỷ hiệu cấp n :



* Áp dụng vào chương trình bằng cách chạy 2 vòng for lồng nhau :

For i =1 to n-1 :

For j = n down to n – i:

y[j][i] = (y[j][i - 1] - y[j - 1][i - 1]) / (x[j] - x[j - i])

Bước 11: Hàm tính giai thừa để tính hệ số của đa thức Newton mốc cách đều

Funtion giaiThua(n)

* Dữ liệu vào: n
* Dữ liệu ra: n!
* Sử dụng đệ quy:

If n == 0 or n == 1 return 1

Return n \* giaithua(n-1)

Bước 12 : Viết đa thức Newton

* Dữ liệu vào: x[], y[][], bảng sai phân hoặc tỷ hiệu
* Dữ liệu ra: Đa thức Newton
* Ý tưởng: từ bảng sai phân/ tỷ hiệu đã có bên trên chọn dạng biểu diễn đa thức Newton tương ứng.

Bước 13:

Funtion newtonTienMocBatKy

* Dữ liệu vào: x[], y[], bảng tỷ hiệu thuận.
* Dữ liệu ra: Đa thức Newton tiến mốc bất kỳ.
* Từ đa thức Newton tiến mốc bất kỳ tổng quát:



* Để biểu diễn đa thức, ta chia làm 2 phần: phần hệ số tự do và phần chứa x.

For i = 0 to n – 1:

If i == 0

Print y[0][0];

Else print y[0][i]

For j = 0 to i – 1 :

If x[j] < 0. Print (x + x[j])

Else if x[j] > 0. Print (x - x[j])

Else Print (x)

If i != (n-1)

If y[0][i+1] > = 0

Print ( + )

* Lưu ý: đa thức nội suy Newton tiến xuất phát từ , thường dùng để tính gần đúng f(c) với c gần .

Bước 14:

Funtion newtonLuiMocBatKy

* Dữ liệu vào: x[], y[][], bảng tỷ hiệu thuận.
* Dữ liệu ra: Đa thức Newton lùi mốc bất kỳ.
* Từ đa thức Newton lùi mốc bất kỳ tổng quát:



* Để biểu diễn đa thức, ta chia làm 2 phần: phần hệ số tự do và phần chứa x.

For i = n down to 1:

If i == n

Print y[n-1][0];

Else print y[n-1][n-i]

For j = n down to i + 1 :

If x[j-1] < 0. Print (x + x[j-1])

Else if x[j-1] > 0. Print (x - x[j-1])

Else Print (x)

If i != 1

If y[n-1][i-1] > = 0

Print ( + )

* Lưu ý: đa thức nội suy Newton lùi xuất phát từ , thường dùng để tính gần đúng f(c) với c gần .

Bước 15:

Funtion newtonTienMocCachDeu

* Dữ liệu vào: x[], y[][], bảng tỷ hiệu thuận.
* Dữ liệu ra: Đa thức Newton tiến mốc cách đều.
* Từ đa thức Newton tiến mốc cách đều tổng quát:



* Để biểu diễn đa thức, ta chia làm 2 phần: phần hệ số tự do và phần chứa x.

For i = 0 to n – 1:

If i == 0

Print ((y[0][1]) / (giaiThua(i) \* pow(denta, i)) >= 0);

Else print (y[0][i]) / (giaiThua(i) \* pow(denta, i))

For j = 0 to i – 1 :

If x[j] < 0. Print (x + x[j])

Else if x[j] > 0. Print (x - x[j])

Else Print (x)

If i != (n-1)

If y[0][i+1] > = 0 : Print ( + )

Bước 16:

Funtion newtonLuiMocCachDeu

* Dữ liệu vào: x[], y[][], bảng tỷ hiệu thuận.
* Dữ liệu ra: Đa thức Newton lùi mốc cách đều.
* Từ đa thức Newton lùi mốc cách đều tổng quát:



* Để biểu diễn đa thức, ta chia làm 2 phần: phần hệ số tự do và phần chứa x.

For i = n down to 1:

If i == n

Print ((y[n - 1][1]) / (giaiThua(i) \* pow(denta, i)) >= 0);

Else print printf("%.2lf", (y[n - 1][n - i]) / (giaiThua(n - i) \* pow(denta,(n - i)))

For j = n down to i + 1 :

If :x[j - 1] < 0. Print (x + x[j - 1])

Else if: x[j - 1] > 0. Print (x - x[j - 1])

Else: Print (x)

If i != 1: 

If y[n-1][i-1] > = 0:

Print ( + )

Bước 17: Tìm hệ số của đa thức tổng quát: 

* Dữ liệu vào: n, x[], y[][].
* Dữ liệu ra: hệ số của đa thức ứng với từng bậc.
* Ý tưởng: Tận dụng ưu điểm “tính kế thừa” của đa thức Newton (mà đa thức Lagrange không có), tức tại mỗi đa thức con của đa thức P(x) sẽ có 2 đa thức nhân với nhau (trong đó có 1 đa thức luôn có dạng (x -a)) => Ý tưởng tương tự thuật toán đệ quy.

Bước 18: hàm sao chép mảng

input: x[]

output: y[] = x[]

Funtion cpMang:

for i = 0 to n – 1 : y[i] = x[i]

Bước 19: hàm nhân hệ số vào mảng

input: x[], p là số tự do

output: mảng x’[] với các phần tử của mảng thành x[i]\*p

Funtion nhanHeSo:

for i = 0 to n – 1: x[i] = x[i] \* p

Bước 20 : hàm in mảng

input: x[]

output: hiển thị x[] ra màn hình

Funtion inMangMotChieu:

for i = 0 to n -1 :

print( x[i] ).

Bước 21 : Tìm hệ số của đa thức Newton tiến mốc bất kỳ

Funtion heSoNewtonTienMocBatKy

* Khởi tạo 2 mảng cấp phát động:

mangMu[][]: Ma trận lưu hệ số của từng đa thức con

tmpMangMu[]: Mảng lưu “hệ số” tạm thời của đa thức con trước để sử dụng cho đa thức con tiếp theo.

* Tìm hàng 0 của ma trận mangMu[i][j]

For i = 0 to n - 1:

If i == 0: mangMu[0][0] = y[0][0].

* Tìm hàng 1 của ma trận mangMu[i][j]

For i = 0 to n - 1:

If i == 1:

mangMu[1][0] = -x[0].

mangMu[1][1] = 1.

Sao chép mangMu[i] => tmpMangMu

mangMu[i] = mangMu[i] \* y[0][1]

* Trường hợp tổng quát

For i = 0 to n -1:

For j = 0 to i:

For k = 0 to 1:

If k == 0 : mangMu[i][j + k] = mangMu[i][j + k] – tmpMangMu[j] \* x[i-1]

Else: mangMu[i][j + k] = mangMu[i][j + k] + tmpMangMu[j]

Sao chép mangMu[i] => tmpMangMu

mangMu[i] = mangMu[i] \* y[0][i]

Bước 22 : Tìm hệ số của đa thức Newton tiến mốc cách đều

Funtion heSoNewtonTienMocCachDeu

* Khởi tạo 2 mảng cấp phát động:

mangMu[][]: Ma trận lưu hệ số của từng đa thức con

tmpMangMu[]: Mảng lưu “hệ số” tạm thời của đa thức con trước để sử dụng cho đa thức con tiếp theo.

denta = x[1] – x[0]

* Tìm hàng 0 của ma trận mangMu[i][j]

For i = 0 to n - 1:

If i == 0: mangMu[0][0] = y[0][0].

* Tìm hàng 1 của ma trận mangMu[i][j]

For i = 0 to n - 1:

If i == 1:

mangMu[i][0] = -x[0].

mangMu[i][1] = 1.

Sao chép mangMu[i] => tmpMangMu

mangMu[i] = mangMu[i] \* (y[0][1] / denta)

* Trường hợp tổng quát

For i = 0 to n -1:

For j = 0 to i:

For k = 0 to 1:

If k == 0 : mangMu[i][j + k] = mangMu[i][j + k] – tmpMangMu[j] \* x[i-1]

Else: mangMu[i][j + k] = mangMu[i][j + k] + tmpMangMu[j] \* 1

Sao chép mangMu[i] => tmpMangMu

mangMu[i] = mangMu[i] \* (y[0][i] / (giaithua(i) \* pow(denta , i)

Bước 23 : Tìm hệ số của đa thức Newton lùi mốc bất kỳ

Funtion heSoNewtonLuiMocBatKy

* Khởi tạo 2 mảng cấp phát động:

mangMu[][]: Ma trận lưu hệ số của từng đa thức con

tmpMangMu[]: Mảng lưu “hệ số” tạm thời của đa thức con trước để sử dụng cho đa thức con tiếp theo.

* Tìm hàng 0 của ma trận mangMu[i][j]

For i = n – 1 down to 0 :

If i == n - 1: mangMu[n – i - 1][0] = y[n - 1][0].

* Tìm hàng 1 của ma trận mangMu[i][j]

For i = n – 1 down to 0:

If i == n - 2:

mangMu[n – i - 1][0] = -x[n - 1].

mangMu[n – i - 1][1] = 1.

Sao chép mangMu[i] => tmpMangMu

mangMu[i] = mangMu[i] \* y[n - 1][n – i - 1]

* Trường hợp tổng quát

For i = n – 1 down to 0:

For j = 0 to n – i - 1:

For k = 0 to 1:

If k == 0 : mangMu[n – i - 1 ][j + k] = mangMu[n – i - 1][j + k] – tmpMangMu[j] \* x[I + 1]

Else: mangMu[i][j + k] = mangMu[n – i - 1][j + k] + tmpMangMu[j] \* 1

Sao chép mangMu[i] => tmpMangMu

mangMu[i] = mangMu[n – i -1] \* y[n - 1][n - i - 1]))

Bước 24 : Tìm hệ số của đa thức Newton lùi mốc cách đều

Funtion heSoNewtonLuiMocCachDeu

* Khởi tạo 2 mảng cấp phát động:

mangMu[][]: Ma trận lưu hệ số của từng đa thức con

tmpMangMu[]: Mảng lưu “hệ số” tạm thời của đa thức con trước để sử dụng cho đa thức con tiếp theo.

* Tìm hàng 0 của ma trận mangMu[i][j]

For i = n – 1 down to 0 :

If i == n - 1: mangMu[n – i - 1][0] = y[n - 1][0].

* Tìm hàng 1 của ma trận mangMu[i][j]

For i = n – 1 down to 0:

If i == n - 2:

mangMu[n – i - 1][0] = -x[n - 1].

mangMu[n – i - 1][1] = 1.

Sao chép mangMu[i] => tmpMangMu

mangMu[i] = mangMu[i] \* (y[n - 1][n – i - 1] / denta)

* Trường hợp tổng quát

For i = n – 1 down to 0:

For j = 0 to n – i - 1:

For k = 0 to 1:

If k == 0 : mangMu[n – i - 1 ][j + k] = mangMu[n – i - 1][j + k] – tmpMangMu[j] \* x[I + 1]

Else: mangMu[i][j + k] = mangMu[n – i - 1][j + k] + tmpMangMu[j] \* 1

Sao chép mangMu[i] => tmpMangMu

mangMu[i] = mangMu[n – i -1] \* y[n - 1][n - i - 1] / (giaiThua(n-i-1) \* pow(denta,(n – i -1))

Bước 25 : Sử dụng sơ đồ Hoocne để tính giá trị đa thức tại 1 điểm xác định.

* Dữ liệu vào: n, c,  (i=1,n).
* Dữ liệu ra : P(c).
* Ý tưởng: Đặt 

Lặp i = 1 to n : p = p \* c + 

Bước 26 :

Funtion hoocne(arr[], n)

Input c;

p= arr[0]

for i = 0 to n :

p = p\*c + arr[i];

return p;

Bước 27: lấy p chữ số thâp phân sau dấu phẩy

input: số cần hiển thị num, số chứ số sau dấu phẩy p (vd: num = 123.45678, p= 2)

output: số num với phần thập phân có p chữ số (num = 123.45)

Ý tưởng: Chia 1 số thành 2 phần:

Phần 1: Phần số nguyên + dấu “ . ”

Phần 2: Phần số thập phân. Ta chú trọng xử lý phần này.

Bước 28:

Funtion hienThiSoTP

curren\_digit : biến lưu số thập phân đang xét

next\_digit : biến lưu số thập phân ngay sau số thập phân đang xét

if (num < 0 && num > -1 ) :

print(“ -0. “)

else :

print( (int)num. ) //in phần nguyên của num và dấu “ . ”

//xử lý phần thập phân

for (i = 0 to n -1 ) :

current\_digit = (int) ((num \* pow (10, i + 1)) % 10 )

next\_digit = (int) ((num \* pow (10, i + 2)) % 10 )

if (i == n – 1 && next\_digit > 4 ) // bước làm tròn

print ( abs (current\_digit + 1) )

else: print ( abs (current\_digit + 1) )

Bước 29: Hàm xóa nội dung trong file

Ý tưởng: sử dụng chế độ “ w ” để xóa dữ liệu trong file

Funtion clearFile

FILE \* f = fopen (“newton.txt” , “w”)

fclpse(f)

Bước 30: Các hàm in kết quả ra file

Ý tưởng chung: lấy các kết quả đã được thực hiện từ các hàm chức năng bên trên để truyền vào hàm.

Coppy các thao tác in ra màn hình và sửa thành .

Các thao tác được sử dụng trong các hàm in kết quả ra file là:

* FILE \* fout = fopen (“newton.txt”, “a”) : khai báo biến fout (sử dụng con trỏ file) để mở tệp Newton.txt
* chế độ: “a” : mở 1 file văn bản và ghi tiếp vào cuối văn bản, nếu nó chưa tồn tại thì file mới được tạo
* fclose( fout )