TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN TOÁN ỨNG DỤNG VÀ TIN HỌC

———*———

BÁO CÁO BÀI TẬP LỚN

HỌC PHẦN: KỸ THUẬT LẬP TRÌNH

Bài 3

Giảng viên: TS. Nguyễn Thị Thanh Huyền

Mã Học phần: MI3310
Mã Lớp học: 116444

Sinh viên thực hiện: Nhóm 10

Nguyễn Trần Thức MSSV: 20185483 Mã Lớp: MI2
Đoàn Quốc Vĩnh MSSV: 20185427 Mã Lớp: MI1

HÀ NỘI, 6/2020

Mục lục

Danh sách hình vẽ

Chương 1

Các bước thực hiện phát triển chương trình

Bài 3 - Thiết kế và viết chương trình

Giải gần đúng hệ phương trình đại số tuyến tính $Ax = b$ bằng **phương pháp lặp đơn và lặp Seidel**:

1) Nhập A, b theo khuân dạng ma trận.

2) Kiểm tra tính chéo trội.

3) Tính chuẩn của ma trận, kiểm tra sự hội tụ.

4) Tính nghiệm gần đúng số lần lặp k cho trước, đánh giá sai số.

5) Tính nghiệm gần đúng với sai số e cho trước.

6) Tính nghiệm gần đúng $X^{(k)}$ thỏa mãn: $\left\|X^{(k)} - X^{(k-1)}\right\| \leq e$ cho trước.

- Mọi kết quả hiển thị với số chữ số thập phân nhập từ bàn phím.
- In cả kết quả trung gian ra màn hình và tệp văn bản.
- Chương trình có chức năng hiển thị kết quả từ tệp văn bản.[a]
- Điều khiển chương trình bằng menu.

[a] Theo ý hiểu của nhóm thì yêu cầu này có nghĩa: Ngoài ma trận nhập từ bàn phím, thì chương trình còn có tính năng nhập ma trận từ file dữ liệu.

Hình 1.1: Yêu cầu bài toán

1.1 Xác định bài toán

1.1.1 Dữ liệu đầu vào

Đầu vào của bài toán là hệ phương trình tuyến tính n ẩn, n phương trình. Khi biểu diễn trên ngôn ngữ lập trình ta đưa dữ liệu vào dưới dạng *ma trận hệ số* và *ma trận*

vế phải. Ngoài ra, Input còn có số lần lặp *k*, sai số *epsilon*, số *e* , số chữ số thập phân nhập từ bàn phím, và *file data* chứa các phần tử của ma trận để nhập ma trận từ file.

Về kiểu dữ liệu thì số ẩn, số k, số chữ số thập phân *kiểu nguyên*, sai số và số e kiểu *số thực*, còn ma trận có thể tạo kiểu dữ liệu ma trận hoặc đơn giản hơn là dùng *mảng 2 chiều* để biểu diễn.

Về menu điều khiển, bắt sự kiện từ bàn phím để điều khiển chương trình:

- Hai phím *ARROW UP* và *ARROW DOWN* để chuyển giữa các lựa chọn.
- *ENTER* để chọn lựa chọn đó.

1.1.2 Dữ liệu đầu ra

1. Tính chéo trội: chéo trội hoặc không chéo trội.
2. Chuẩn của ma trận: Kiểu số thực.
3. Sự hội tụ: Hội tụ tới X^* hoặc không hội tụ tới X^*.
4. Nghiệm gần đúng: Kiểu mảng 2 chiều, in ra màn hình và tệp văn bản(cả kết quả chung gian).
5. Sai số: Kiểu số thực.

1.2 Cơ sở học phần *Giải tích số*: giải gần đúng hệ phương trình tuyến tính

Trong học phần đại số tuyến tính mà nhóm đã học thì hệ Gramer được dùng phổ biến để giải hệ phương trình tuyến tính. Nhưng nhược điểm của phương pháp này khi cài đặt trên máy tính điện tử là số lượng phép tính sơ cấp khá lớn, do đó phương pháp này là không khả thi. Vì vậy có phương pháp khác hiệu quả hơn để tính gần đúng nghiệm của hệ phương trình tuyến tính là phương pháp lặp đơn và phiên bản tối ưu hơn của lặp đơn là lặp Seidel. Vì đây không phải là học phần giải tích số nên nhóm sẽ tóm gọn lại công thức liên quan đến bài toán.

1.2.1 Ma trận chéo trội, chuẩn của ma trận

A là ma trận chéo trội thì:

$$| a_{ii} | > \sum_{j=1, j\neq i}^{n} | a_{ij} | \qquad (i = \overline{1,n}) \tag{1.1}$$

Chuẩn của ma trận B theo hàng:

$$\left\|B\right\|_{(\infty)} = \max_i \sum_{j=1}^{n} \mid b_{ij} \mid \qquad (\left\|B\right\|_{(\infty)} \geq 0) \tag{1.2}$$

1.2.2 Điều kiện hội tụ tới nghiệm đúng

Để hệ phương trình $Ax = b \Leftrightarrow x = \alpha x + \beta$ hội tụ tới nghiệm X^* thì:

- A là ma trận chéo trội.
- $\left\|\alpha\right\|_{(\infty)} \leq q < 1$

1.2.3 Phương pháp lặp đơn

Ta xét hệ phương trình khi biến đổi về dạng: $x = \alpha x + \beta$ thỏa mãn (1.2.2)

Chọn xấp xỉ đầu $X^{(0)} = \beta$ được nghiệm gần đúng $X^{(k)}$ tính bởi công thức lặp:

$$X^{(k)} = \alpha X^{(k-1)} + \beta \tag{1.3}$$

$$x_i^{(k)} = \sum_{j=1}^{i-1} \alpha_{ij}.x_j^{(k)} + \sum_{j=i}^{n} \alpha_{ij}.x_j^{(k-1)} + \beta_i \tag{1.4}$$

$$(i = \overline{1,n}, \qquad k = 1,2,3...)$$

Công thức *lặp đơn* (1.3), và công thức *lặp Seidel* (1.4). Sở dĩ công thức *lặp Seidel* được viết ở mục này (1.2.3) là vì phương pháp lặp đơn là tổng quát, 2 phương pháp là *lặp Jacobi* (chi tiết hơn về cách biến đổi về dạng $x = \alpha x + \beta$) và *lặp Seidel* (cải tiến về công thức lặp).

1.2.4 Sai số phương pháp

$$\left\|X^{(k)} - X^*\right\|_{(\infty)} \leq \frac{q}{1-q}\left\|X^{(k)} - X^{(k-1)}\right\|_{(\infty)} \tag{1.5}$$

Hai phương pháp có điểm chung là cùng công thức sai số (1.5).

1.3 Ý tưởng giải quyết bài toán

1.3.1 Thiết kế bằng phương pháp tinh chỉnh dần từng bước

Phương pháp này sẽ đề cập chi tiết trong chương 2 của bài *báo cáo* này.

1.3.2 Cài đặt chương trình theo thuật toán

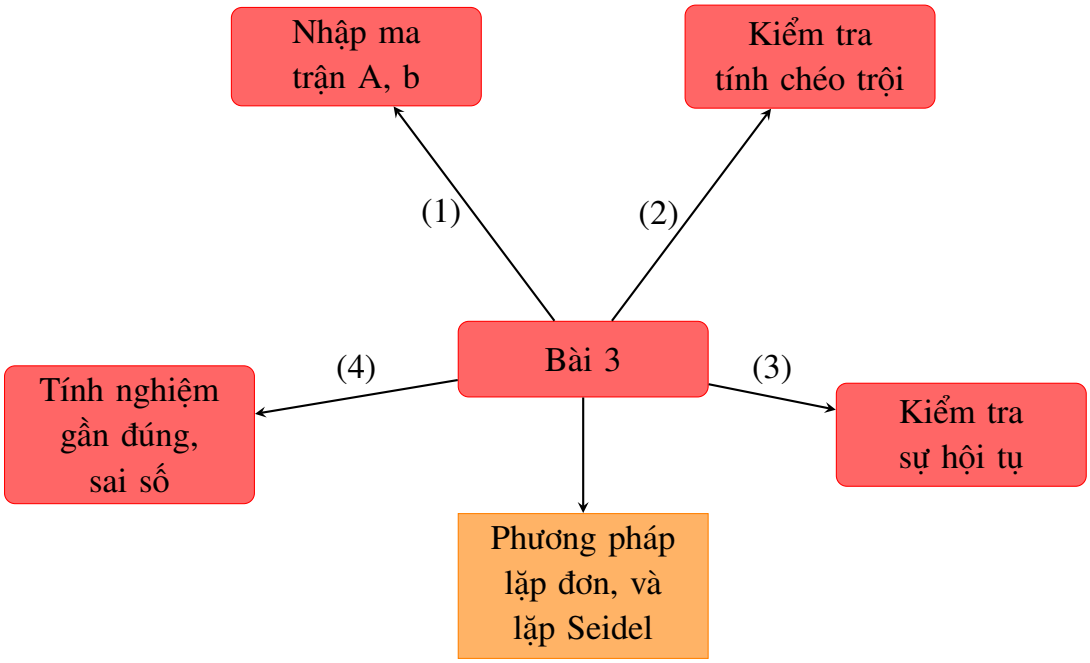
1. Viết các module. Kiểm nghiệm và sửa lỗi từng module thật kỹ trước rồi ghép và chương trình chính. Nếu có các module liên quan đến nhau, giao tiếp với nhau(một cách dễ hiểu là trong hàm này, có lệnh gọi đến hàm khác) thì viết chúng trong cùng 1 chương trình để kiểm nghiệm tính đúng đắn của các module đó. Nếu những module có thể tách thành các module khác nhỏ hơn thì viết code cho các module đó trước.

2. Viết chương trình chính, chương trình chính dùng các lệnh, và gọi lại các module theo 1 tính logic để hoàn thành bài toán. Lúc này việc sửa lỗi sẽ dựa trên chương trình chính và kết quả khi chạy chương trình.
   Ngược lại, nếu ta viết chương trình chính trước, thì sẽ khó có thể hình dung được kết quả trả về và tính logic của chương trình chính bởi vì lúc này chương trình không chạy được vì thiếu các module (Nhóm phản biện lại quan điểm viết chương trình chính trước, module sau trong Slide của giảng viên).

Chương 2

Phương pháp tinh chỉnh từng bước

2.1 Module hóa bài toán - thiết kế kiểu top-down



Hình 2.1: Các module chính của bài toán

Bài toán được chia thành 4 module và mặc định có 2 module xử lý chính của bài toán là phương pháp lặp đơn, lặp Seidel.

Module (1) được chia làm 2 module nhỏ hơn là nhập ma trận từ bàn phím và từ tệp văn bản.

Module (2) Giữ nguyên do không phân tách thành module nhỏ hơn.

Module (3) để giải được module này và các module phía sau nữa thì phải tạo ra thêm 2 module nữa là tính chuẩn của ma trận và biến đổi ma trận.

Module (4) không phân tách thành module nhỏ hơn nhưng do yêu cầu bài toán,

nhóm tách thành 3 module con là nghiệm gần đúng với số lần lặp k cho trước, đánh giá sai số, với sai số cho trước, thỏa mãn điều kiện $\left\|X^{(k)} - X^{(k-1)}\right\| \leq e$ cho trước.

Ngoài ra, ta cần viết thêm các module theo yêu cầu thêm của bài toán:

- In số chữ thập phân nhập từ bàn phím.
- Điều khiển chương tình bằng menu (điều khiển này được viết trong chương trình chính).
- In kết quả ra tệp văn bản (Nhóm thống nhất ghép chung vào module in kết quả vì nó thuận tiện và không đáng phải tách ra module riêng).

2.2 Chi tiết hóa dần Module

Việc module hóa bài toán ở mục (2.1) đã tạo ra khoảng trên 10 module nhỏ, việc chi tiết hóa từng module ở mục này sẽ đi mô tả, input, output, và giả code của các *module chính*.

Quy ước trong phạm vi chương trình:

- n là số ẩn của hệ phương trình.
- k là số lần lặp được nhập từ bàn phím(ý số 4), hoặc là kết quả từ việc xử lý ở ý 5,6 trong yêu cầu bài toán (1).
- Mảng 2 chiều *a* chứa ma trận A, b như hình:

a[0][0]	a[0][1]	a[0][2]	a[0][3]	...	a[0][n]	a[0][n+1]
a[1][0]	A[1][1]	A[1][2]	A[1][3]	...	A[1][n]	b[1][1]
a[2][0]	A[2][1]	A[2][2]	A[2][3]	...	A[2][n]	b[2][1]
a[3][0]	A[3][1]	A[3][2]	A[3][3]	...	A[3][n]	b[3][1]
⋮	⋮	⋮	⋮	⋱	⋮	⋮
a[n][0]	A[n][1]	A[n][2]	A[n][3]	...	A[n][n]	b[n][1]

Hình 2.2: Mảng 2 chiều a

- Khi biến đổi về dạng $x = Tx + c$ thì mảng 2 chiều a chứa ma trận A, b sẽ thay thế lần lượt bằng ma trận T, c bởi 1 module có chức năng đó.
- Mảng 2 chiều *result* chứa kết quả và kết quả trung gian như hình:

result[0][0]	result[0][1]	result[0][2]	result[0][3]	...	result[0][k]
$x_1^{(0)}$	$x_1^{(1)}$	$x_1^{(2)}$	$x_1^{(3)}$	...	$x_1^{(k)}$
$x_2^{(0)}$	$x_2^{(1)}$	$x_2^{(2)}$	$x_2^{(3)}$	...	$x_2^{(k)}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$x_n^{(0)}$	$x_n^{(1)}$	$x_n^{(2)}$	$x_n^{(3)}$	...	$x_n^{(k)}$

Hình 2.3: Mảng 2 chiều result

2.2.1 Module: Phương pháp lặp đơn

Input:

- Mảng 2 chiều a, result các phần tử là số thực.
- Số nguyên n, số thực k.

Output: Không trả về giá trị, kết quả xử lý truyền vào mảng 2 chiều result và được in ra bởi hàm in kết quả.

Mô tả:

- gán cột đầu tiên của ma trận kết quả bằng ma trận c (xấp xỉ đầu $X^{(0)} = c$).
- duyệt theo cột của ma trận kết quả tới số k.
- nhân ma trận T với ma trận kết quả này rồi cộng với ma trận c và trả về ma trận kết quả đứng sau nó (tính theo công thức lặp 1.3).
- Ma trận kết quả có số hàng là số nghiệm, số cột là số lần lặp k.

Giả code:

```
iterative_method (a[][], result[][], n, k)
    for i = 1 to n
        result[i][0] = a[i][n+1]
    for h = 1 to k
        for i = 1 to n
            result[i][h] = 0
            for j = 1 to n
                result[i][h] += a[i][j] * result[j][h-1]
            result[i][h] += a[i][n+1]
```

Hình 2.4: Phương pháp lặp đơn

2.2.2 Module: Phương pháp lặp Seidel

Input:

- Mảng 2 chiều a, result các phần tử là số thực.
- Số nguyên n, số thực k.

Output: Không trả về giá trị, kết quả xử lý truyền vào mảng 2 chiều result và được in ra bởi hàm in kết quả.

Mô tả:

- gán cột đầu tiên của ma trận kết quả bằng ma trận c (xấp xỉ đầu $X^{(0)} = c$).
- Các nghiệm x_1 (hàng 1 của ma trận kết quả) được tính như phương pháp lặp phía trên.
- Duyệt dòng thứ 2 tới n của ma trận kết quả có hai vòng for bên trong để tính 2 cái tổng $\sum$ của công thức lặp (1.4).
- Ma trận kết quả có số hàng là số nghiệm, số cột là số lần lặp k.

Giả code:

```
seidel_iterative_method (a[][], result[][], n, k)
    for i = 1 to n
        result[i][0] = a[i][n+1]
    for h = 1 to k
        result[1][h] = result[1][0]
        for j = 1 to n
            result[1][h] += a[1][j] * result[j][h-1]
        for i = 2 to n
            result[i][h] = result[i][0]
            for j = 1 to i-1
                result[i][h] += a[i][j] * result[j][h]
            for t = i to n
                result[i][h] += a[i][t] * result[t][h-1]
```

Hình 2.5: Phương pháp lặp Seidel

2.2.3 Module: Nhập ma trận

Input:

- Mảng 2 chiều a phần tử là số thực.
- Số nguyên n.

Output:

Không trả về giá trị.

Mô tả:

- Duyệt hàng 1 tới n, cột 1 tới n.
- Nhập hệ số của ma trận A.
- Duyệt hàng 1 tới n, cột n+1.
- Nhập hệ số của ma trận b.

Giả code:

```
get_matrix (a[][], n)
    for i = 1 to n
        for j = 1 to n
            scanf (a[i][j]) //nhap ma tran A
    for i = 1 to n
        scanf (a[i][n+1]) //nhap ma tran b
```

Hình 2.6: Nhập ma trận

2.2.4 Module: Xuất kết quả

Input:

- Mảng 2 chiều result phần tử là số thực.
- Số nguyên n.
- Số thực k

Output:

In ra kết quả nghiệm.

Mô tả:

- Duyệt mảng 2 chiều result từ hàng 1 tới n, và cột 0 tới k.
- In các phần tử.

  Giả code:

```
print_result (result[][], n, k)
    for i = 1 to n
        for j = 0 to k
            print (result[i][j])
```

Hình 2.7: Xuất kết quả

2.2.5 Module: Tìm chuẩn của ma trận

Input:

- Mảng 2 chiều a: chứa ma trận T, c sau khi biến đổi ($x = Tx + c$).
- Số nguyên n.

Output:

Trả về chuẩn của ma trận kiểu *double*.

Mô tả:

- Duyệt từng hàng của ma trận.
- Tính tổng các phần tử của hàng.
- So sánh tổng của các hàng với với nhau, tìm hàng có tổng max.
- Tổng max chính là chuẩn của ma trận đó.

* Dựa trên công thức (1.2).

Giả code:

```
norm_matrix (a[][], n)
    sum_row = 0
    sumMax = 0
    for i = 1 to n
        for j = 1 to n
            sum_row = sum_row + |a[i][j]|
        if sum_row >= sumMax
        then sumMax = sum_row
        sum_row = 0
    return sumMax
```

Hình 2.8: Tìm chuẩn của ma trận

2.2.6 Module: Kiểm tra tính chéo trội của ma trận

Input:

- Mảng 2 chiều a: chứa ma trận T, c sau khi biến đổi ($x = Tx + c$).
- Số nguyên n.

Output:

Trả về giá trị *True / False*.

Mô tả:

- Giả sử ma trận là chéo trội.
- Nếu phần tử trên đường chéo chính nhỏ hơn tổng các phần tử còn lại (cùng hàng) thì ma trận không phải là chéo trội.

* Dựa trên công thức (1.1).

Giả code:

```
is_diagonally_dominant_matrix (a[][], n)
    flag = true
    sum_row = 0
    for i = 1 to n
        for j = 1 to n
            if i != j
            then sum_row = sum_row + |a[i][j]|
        if fabs (a[i][i]) < sum_row or a[i][i] == 0
            flag = false
```

Hình 2.9: Kiểm tra tính chéo trội của ma trận

2.2.7 Module: Tìm nghiệm gần đúng với sai số cho trước

Input:

- Mảng 2 chiều a: chứa ma trận T, c sau khi biến đổi ($x = Tx + c$)
- Mảng 2 chiều result
- Số nguyên n.
- Số thực e.

Output:

Trả về số lần lặp k.

Mô tả:

* Từ công thức (1.5) $\Leftrightarrow \left\|X^{(k)} - X^{(k-1)}\right\| \leq \frac{\left\|X^{(k)} - X^{*}\right\|.(1-q)}{q}$

* Module này gọi tới Module tìm chuẩn của ma trận

- Tăng k ở vế trái lên
- Chừng nào mà vế trái còn lớn hơn vế phải thì còn tăng k
- Khi vế trái nhỏ hơn vế phải thì thoát vòng lặp
- Trả về giá trị của k

Giả code:

```
approximate_solution_method (a[][], result[][], n, e)
    q = norm_matrix (a, n)
    k = 0
    do
      k = k + 1
        norm = -999
        for i = 1 to n
            result[0][i] = result[i][k] - result[i][k-1]
            if |result[0][i]| > norm
                norm = result[0][i]
        c = norm
        r = e * (1 - q) / q
    while (c >= r)
    return k
```

Hình 2.10: Tìm nghiệm gần đúng

2.2.8 Module: Biến đổi ma trận về dạng: $x = Tx + c$

Input:

- Mảng 2 chiều a: chứa ma trận T, c sau khi biến đổi ($x = Tx + c$).
- Số nguyên n.

Output:

Ma trận a được biến đổi về dạng $x = Tx + c$ trong đó T là ma trận có đường chéo chính bằng 0.

Mô tả:

- Chuyển các phần tử trên đường chéo chính sang 1 vế.
- Chia cả hàng đó cho hệ số phần tử vừa chuyển vế.

Giả code:

```
matrix_transform (a[][], n)
    for i = 1 to n
        a[i][0] = -a[i][i]
        a[i][i] = 0
        a[i][n+1] = -a[i][n+1]
        x = a[i][0]
        for j = 1 to n+1
            if (i != j) then a[i][j]= a[i][j]/x
```

Hình 2.11: Biến đổi ma trận

2.2.9 Module: Tính sai số

Input:

- Mảng 2 chiều a: chứa ma trận T, c sau khi biến đổi ($x = Tx + c$).
- Mảng 2 chiều result.
- Số nguyên n.
- Số thực k.

Output:

In ra sai số 2 phương pháp.

Mô tả:

- Tính chuẩn của hiệu 2 ma trận của 2 lần lặp k cuối cùng.
- Nhân với chuẩn ma trận T chia 1 trừ chuẩn ma trận T.

* Module này có gọi đến module tìm chuẩn ma trận.

* Dựa trên công thức (1.5).

Giả code:

```
error_estimators (a[][], result[][], n, k)
   norm = -999
    for i = 1 to n
        result[i][k+1] = result[i][k] - result[i][k-1];
        if |result[i][k+1]| > norm
            norm = result[i][k+1]
    q = norm_matrix (a, n)
    saiso = q / (1-q) * norm
    print (saiso)
```

Hình 2.12: Kiểm tra tính chéo trội của ma trận

Chương 3

Đánh giá kết quả thu được

3.1 Tính đúng đắn của kết quả

Nhóm đã test với các hệ phương trình tuyến tính đã biết kết quả trước, và kết quả trả về của chương trình hoàn toàn trùng khớp với kết quả đã biết trước đó.

3.2 Đánh giá về phong cách lập trình

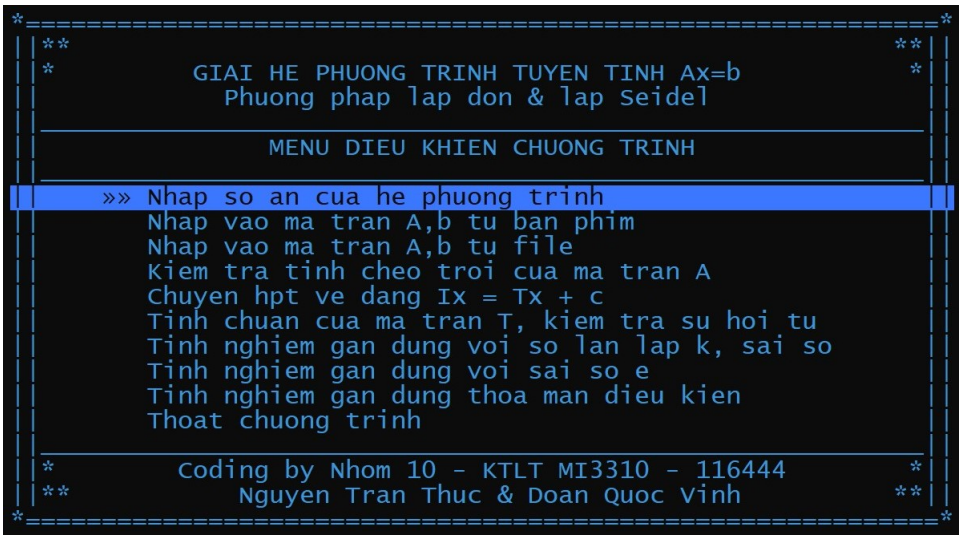
Nhóm trình bày code theo Linux kernel coding style có điểm khác 1 chút là thay vì lùi 8 khoảng trắng thì nhóm chỉ lùi 4 khoảng trắng trong bản code này để code không bị quá lệch về bên phải.

Về cơ bản thì code dễ nhìn, chú thích vừa đủ không quá nhiều.

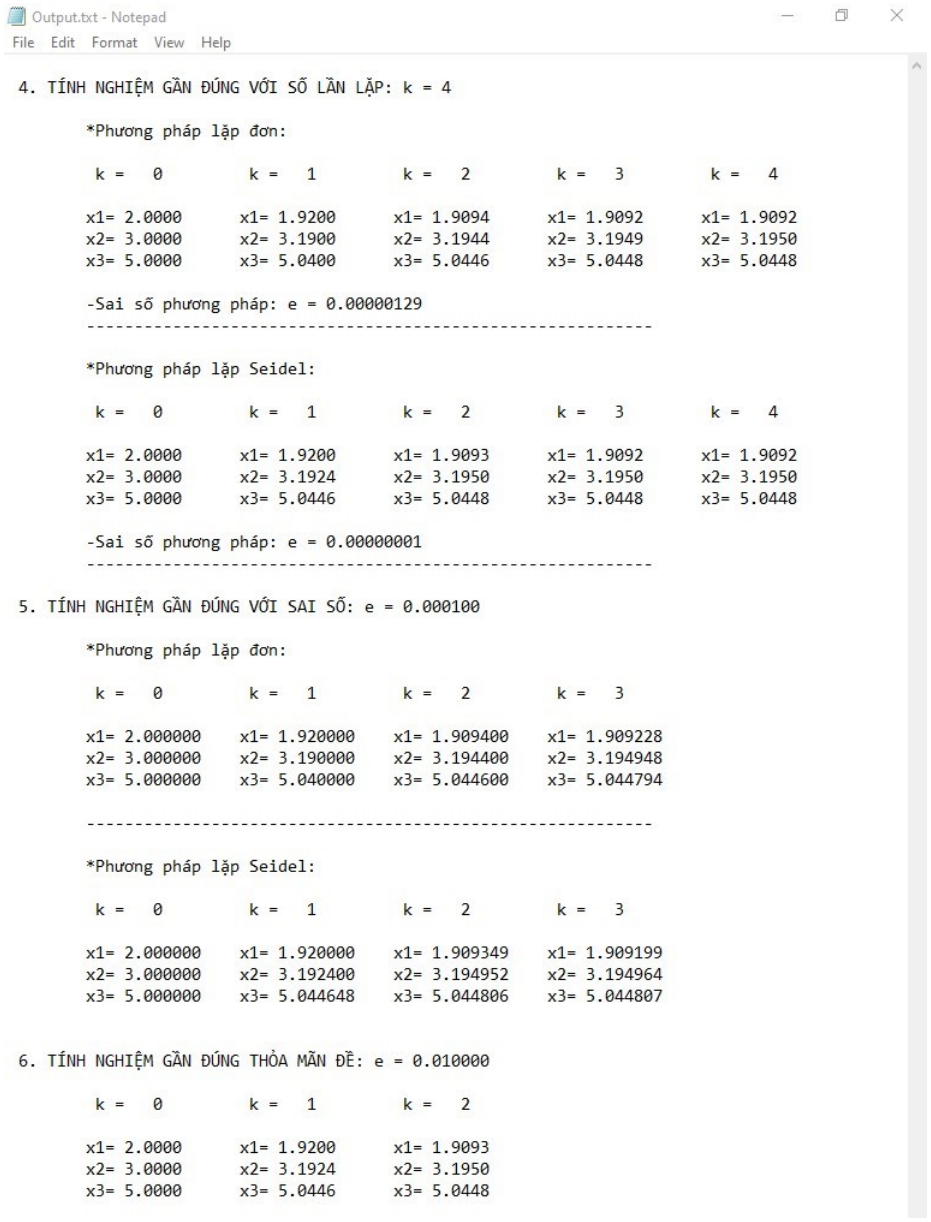
Phân ra làm 4 phần chính:

- Khai báo thư viện hàm.
- Khai báo biến toàn cục.
- Khai báo nguyên mẫu hàm.
- Hàm chính.
- Cài đặt các hàm con.

3.3 Hình ảnh kết quả thu được



Hình 3.1: Menu điều khiển chương trình



Output.txt - Notepad

File Edit Format View Help

4. TÍNH NGHIỆM GẦN ĐÚNG VỚI SỐ LẦN LẶP: k = 4

*Phương pháp lặp đơn:

k = 0	k = 1	k = 2	k = 3	k = 4
x1= 2.0000	x1= 1.9200	x1= 1.9094	x1= 1.9092	x1= 1.9092
x2= 3.0000	x2= 3.1900	x2= 3.1944	x2= 3.1949	x2= 3.1950
x3= 5.0000	x3= 5.0400	x3= 5.0446	x3= 5.0448	x3= 5.0448

-Sai số phương pháp: e = 0.00000129

---

*Phương pháp lặp Seidel:

k = 0	k = 1	k = 2	k = 3	k = 4
x1= 2.0000	x1= 1.9200	x1= 1.9093	x1= 1.9092	x1= 1.9092
x2= 3.0000	x2= 3.1924	x2= 3.1950	x2= 3.1950	x2= 3.1950
x3= 5.0000	x3= 5.0446	x3= 5.0448	x3= 5.0448	x3= 5.0448

-Sai số phương pháp: e = 0.00000001

---

5. TÍNH NGHIỆM GẦN ĐÚNG VỚI SAI SỐ: e = 0.000100

*Phương pháp lặp đơn:

k = 0	k = 1	k = 2	k = 3
x1= 2.000000	x1= 1.920000	x1= 1.909400	x1= 1.909228
x2= 3.000000	x2= 3.190000	x2= 3.194400	x2= 3.194948
x3= 5.000000	x3= 5.040000	x3= 5.044600	x3= 5.044794

---

*Phương pháp lặp Seidel:

k = 0	k = 1	k = 2	k = 3
x1= 2.000000	x1= 1.920000	x1= 1.909349	x1= 1.909199
x2= 3.000000	x2= 3.192400	x2= 3.194952	x2= 3.194964
x3= 5.000000	x3= 5.044648	x3= 5.044806	x3= 5.044807

6. TÍNH NGHIỆM GẦN ĐÚNG THỎA MÃN ĐỀ: e = 0.010000

k = 0	k = 1	k = 2
x1= 2.0000	x1= 1.9200	x1= 1.9093
x2= 3.0000	x2= 3.1924	x2= 3.1950
x3= 5.0000	x3= 5.0446	x3= 5.0448

Hình 3.2: Kết quả in ra tệp văn bản