Report of Assignment 4

Using Jacobian and Gauss-Seidel Methods to Solve Ax = b

Meng Wei

Results:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ITER | Error from Jacobi’s Method | Convergence  Rate | Error from Gauss-Seidel Method | Convergence  Rate |
| 20 | 3.29386819 | 0.32593799 | 2.63680716 | 0.63861700 |
| 40 | 2.62778062 | 0.64822606 | 1.69369613 | 1.29230216 |
| 80 | 1.67669326 | 1.29635047 | 0.69153439 | 2.59240748 |
| 160 | 0.68267381 | 2.59270012 | 0.11466249 | 5.18539874 |
| 320 | 0.11317037 |  | 0.00315109 |  |

PS: Codes as follows:

Jacobi-and-Gauss-Seidel-Methods:

// Jacobi-and-Gauss-Seidel-Methods.cpp : Defines the entry point for the console application.

// Using Jacobian and Gauss-Seidel Methods to solve Ax = b.

#include "stdafx.h"

#include <stdio.h>

#include <malloc.h>

#include <math.h>

#pragma warning(disable:4996)//Disable fopen function warning.

#define ITER 160 // Define the iteration steps number: 20,40,80,160.

// Define Jacobi function

void Jacobi(int n, double \* A, double \* b, double \* x)

{

double s;

int i, j, k = 0;

double e=0; // e represents Error.

x = (double\*)malloc(n\*sizeof(double));

for (i = 0; i < n; i++)

x[i] = 0;

double \*xnew = (double\*)malloc(n\*sizeof(double));

for (i = 0; i < n; i++)

xnew[i] = 0;

while (k<ITER)

{

for (i = 0; i < n; i++)

{

s = 0;

for (j = 0; j < n; j++)

{

if (j != i)

s = s + A[i\*n + j] \* x[j];

}

xnew[i] = (b[i] - s) / A[i\*n + i];

}

for (i = 0; i < n; i++)

{

x[i] = xnew[i];

}

k++;

}

// Print out the results of Jacobi Method

printf("Results of Jacobi Method are :\n");

for (i = 0; i < n; i++)

printf("%.8f\n", x[i]);

//Write Jacobi(n, A, b, x) to text file

FILE \* Jacobi = fopen("Jacobi160.txt", "w");

for (i = 0; i < n; i++)

fprintf(Jacobi, "%.8f\n", x[i]);

fclose(Jacobi);

//Compute error from Jacobi Method by the vector Euclidian distance from your solution to the exact solution sol= {1, 1, …, 1}.

for (i = 0; i < n; i++)

{

e = e + pow(x[i] - 1, 2);

}

e = sqrt(e);

printf("Error from Jacobi is: %.8f\n", e);

}

// Define Gauss-Seidel function

void GaussSeidel (int n, double \* A, double \* b, double \* x)

{

double s;

double e;

int i, j, k = 0;

x = (double\*)malloc(n\*sizeof(double));

for (i = 0; i < n; i++)

x[i] = 0;

while (k< ITER)

{

for (i = 0; i < n; i++)

{

s = 0;

for (j = 0; j < n; j++)

if (j != i)

{

s = s + A[i\*n + j] \* x[j];

}

x[i] = (b[i] - s) / A[i\*n + i];

}

k++;

}

// Print out the results of Gauss-Seidel Method

printf("Results of Gauss-Seidel Method are :\n");

for (i = 0; i < n; i++)

printf("%.8f\n", x[i]);

//Write GaussSeidel(n, A, b, x) to text file

FILE \* GaussSeidel = fopen("GaussSeidel160.txt", "w");

for (i = 0; i < n; i++)

fprintf(GaussSeidel,"%.8f\n", x[i]);

fclose(GaussSeidel);

//Compute error from Gauss-Seidel Method by the vector Euclidian distance from your solution to the exact solution sol= {1, 1, …, 1}.

e = 0;

for (i = 0; i < n; i++)

{

e = e + pow(x[i] - 1, 2);

}

e = sqrt(e);

printf("Error from Gauss-Seidel Method is: %.8f\n", e);

}

//main function

int main()

{

int n;

n=20;

int i, j;

double \*x = (double\*)malloc(n\*sizeof(double));

//Define matrix A satisfies: A[i][i]= 2.0; A[i][i + 1] = -1.0; A[i + 1][i] = -1.0; all other elements of A are zeros.

double \*A = (double\*)malloc((n\*n)\*sizeof(double));

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

if (i == j - 1)

{

A[i\*n + j] = -1.0;

}

else if (i == j)

{

A[i\*n + j] = 2.0;

}

else if (i == j + 1)

{

A[i\*n + j] = -1.0;

}

else { A[i\*n + j] = 0.0; }

}

}

//Define vector b satifies: all elements of b are zeros, except b[0] = b[19] = 1.0.

double \*b = (double\*)malloc(n\*sizeof(double));

for (i = 0; i < n; i++)

{

if (i == 0 || i == n - 1)

b[i] = 1;

else b[i] = 0;

}

// Call Jacobi function.

Jacobi(n, A, b, x);

// Call GaussSeidel function.

GaussSeidel(n, A, b, x);

free(A);

free(b);

}