**נספח**

 התוכנית הבאה מריצה את הרוטינה של אלימינציה של גאוס עם partial pivoting על מטריצה נתונה. על מנת להקל את השימוש בתוכנית מקדמי המטריצה והווקטור b נתונים בתוך קובץ ששמו מועבר כפרמטר לתוכנית ראשית (argv[1]). התוכנית מדפיסה את הפתרון אך מדפיסה פלט המאפשר למשתמש לבדוק ולהשתכנע שהפתרון שהוחזר אכן פותר את המערכת.

/\* gaus1a.c - Partial pivoting \*/

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

void print\_result(double \*A[], double x[], double b[], int n)

{

 int i, j;

 double sum;

 printf("Solution X:\n");

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

 printf(" X[%d] ", i);

 printf("\n");

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

 printf(" %8.3lf ", x[i]);

 printf("\n Verification:\n");

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

 {

 sum = A[i][0] \* x[0];

 printf("%8.3lf \* %8.3lf ", A[i][0], x[0] );

 for(j=1; j < n; j++)

 {

 sum = sum + A[i][j] \* x[j];

 printf(" + %8.3lf \* %10.3lf ", A[i][j], x[j] );

 } /\* for \*/

 printf(" = %8.3lf ?= %8.3lf\n", sum, b[i]);

 } /\* for \*/

} /\* print\_result \*/

void swap\_rows(double \*A[], int n, int m1, int m2)

{

 int i;

 double temp;

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

 {

 temp = A[m1][i];

 A[m1][i] = A[m2][i];

 A[m2][i] = temp;

 } /\* for \*/

} /\* swap\_rows \*/

void gaussian(double \*A[], double b[], int n, double x[])

{

 int i, j, k, p;

 double \*\*W;

 double \*\*M;

 double MaxValue;

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

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

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

 W[i] = (double \*)malloc((n+1)\*sizeof(double));

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

 M[i] = (double \*)malloc(n\*sizeof(double));

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

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

 W[i][j] = A[i][j];

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

 W[i][n] = b[i];

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

 {

 p = k;

 MaxValue = fabs(W[k][k]);

 for(i=k+1; i < n; i++)

 if (fabs(W[i][k]) > MaxValue)

 {

 p = i;

 MaxValue = fabs(W[i][k]);

 }

 swap\_rows(W, n, k, p);

 for(i=k+1; i < n; i++)

 M[i][k] = W[i][k]/W[k][k];

 for(i=k+1; i < n; i++)

 W[i][k] = 0;

 for(i=k+1; i < n; i++)

 for(j=k+1; j <= n; j++)

 W[i][j] = W[i][j] - M[i][k]\*W[k][j];

 } /\* for \*/

 x[n-1] = W[n-1][n]/W[n-1][n-1];

 for(i=n-2; i >= 0; i--)

 {

 double temp;

 temp = W[i][n];

 for(k=i+1; k < n; k++)

 temp = temp - W[i][k]\*x[k];

 x[i] = temp/W[i][i];

 } /\* for \*/

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

 {

 free(M[i]);

 free(W[i]);

 } /\* for \*/

 free(M);

 free(W);

} /\* gaussian \*/

void read\_file(FILE \*fp, double \*A[], double \*b, int n)

{

 int i, j;

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

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

 fscanf(fp, "%lf", &A[i][j]);

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

 fscanf(fp, "%lf", &b[i]);

} /\* read\_file \*/

void print\_original\_system(double \*A[], double b[], int n)

{

 int i, j;

 printf("Original System:\n");

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

 {

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

 printf("%10.3lf", A[i][j]);

 printf("%10.3lf\n", b[i]);

 } /\* for \*/

} /\* print\_original\_system \*/

int main(int argc, char \*argv[])

{

 FILE \*fp;

 int i, n;

 double \*\*A, \*b, \*x;

 if (argc < 2)

 {

 fprintf(stderr, "Usage: gaussian filename\n");

 return 0;

 } /\* if \*/

 fp = fopen(argv[1], "rt");

 fscanf(fp, "%d", &n);

 A = (double \*\*)malloc((n+1)\*sizeof(double \*));

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

 A[i] = (double \*)malloc(n\*sizeof(double));

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

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

 read\_file(fp, A, b, n);

 gaussian(A, b, n, x);

 printf(" A x = b\n");

 print\_original\_system(A, b, n);

 print\_result(A, x, b, n);

 return 0;

} /\* main \*/

אם, לדוגמא, תוכן הקובץ הוא

5 🡨------------------ n מימד המטריצה

2 3 -1 0 5 /||\

1 6 2 -3 -1 ||

2 0 1 4 -2 || מקדמי המטריצה

0 5 -2 1 3 ||

3 1 4 -2 7 \||/

17 /||\

10 ||

-24 || b ערכי הוקטור

-2 ||

48 \||/

פלט הריצה הינו

E:\> GAUS1A.EXE mat2.txt

A x = b

Original System:

 2.000 3.000 -1.000 0.000 5.000 17.000

 1.000 6.000 2.000 -3.000 -1.000 10.000

 2.000 0.000 1.000 4.000 -2.000 -24.000

 0.000 5.000 -2.000 1.000 3.000 -2.000

 3.000 1.000 4.000 -2.000 7.000 48.000

Solution X:

 X[0] X[1] X[2] X[3] X[4]

 1.000 -1.000 2.000 -5.000 4.000

 Verification:

 2.000 \* 1.000 + 3.000 \* -1.000 + -1.000 \* 2.000 + 0.000 \* -5.000 + 5.000 \* 4.000 = 17.000 ?= 17.000

 1.000 \* 1.000 + 6.000 \* -1.000 + 2.000 \* 2.000 + -3.000 \* -5.000 + -1.000 \* 4.000 = 10.000 ?= 10.000

 2.000 \* 1.000 + 0.000 \* -1.000 + 1.000 \* 2.000 + 4.000 \* -5.000 + -2.000 \* 4.000 = -24.000 ?= -24.000

 0.000 \* 1.000 + 5.000 \* -1.000 + -2.000 \* 2.000 + 1.000 \* -5.000 + 3.000 \* 4.000 = -2.000 ?= -2.000

 3.000 \* 1.000 + 1.000 \* -1.000 + 4.000 \* 2.000 + -2.000 \* -5.000 + 7.000 \* 4.000 = 48.000 ?= 48.000

התוכנית הבאה כמעט זהה לקודמת, אך כוללת קוד הדפסה המראה כיצד המטריצה משתנה בכל אחד מהשלבים:

/\* gaus1b.c - Partial pivoting with stage prining \*/

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

void print\_stage(int stage, double \*W[], int n)

{

 int i, j;

 printf("Stage %d:\n", stage);

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

 {

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

 printf("%10.3lf", W[i][j]);

 printf("\n");

 } /\* for \*/

} /\* print\_stage \*/

void print\_result(double \*A[], double x[], double b[], int n)

{

 int i, j;

 double sum;

 printf("Solution X:\n");

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

 printf(" X[%d] ", i);

 printf("\n");

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

 printf(" %8.3lf ", x[i]);

 printf("\n Verification:\n");

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

 {

 sum = A[i][0] \* x[0];

 printf("%8.3lf \* %8.3lf ", A[i][0], x[0] );

 for(j=1; j < n; j++)

 {

 sum = sum + A[i][j] \* x[j];

 printf(" + %8.3lf \* %10.3lf ", A[i][j], x[j] );

 } /\* for \*/

 printf(" = %8.3lf ?= %8.3lf\n", sum, b[i]);

 } /\* for \*/

} /\* print\_result \*/

void swap\_rows(double \*A[], int n, int m1, int m2)

{

 int i;

 double temp;

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

 {

 temp = A[m1][i];

 A[m1][i] = A[m2][i];

 A[m2][i] = temp;

 } /\* for \*/

} /\* swap\_rows \*/

void gaussian(double \*A[], double b[], int n, double x[])

{

 int i, j, k, p;

 double \*\*W;

 double \*\*M;

 double MaxValue;

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

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

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

 W[i] = (double \*)malloc((n+1)\*sizeof(double));

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

 M[i] = (double \*)malloc(n\*sizeof(double));

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

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

 W[i][j] = A[i][j];

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

 W[i][n] = b[i];

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

 {

 print\_stage(k, W, n);

 p = k;

 MaxValue = fabs(W[k][k]);

 for(i=k+1; i < n; i++)

 if (fabs(W[i][k]) > MaxValue)

 {

 p = i;

 MaxValue = fabs(W[i][k]);

 }

 swap\_rows(W, n, k, p);

 for(i=k+1; i < n; i++)

 M[i][k] = W[i][k]/W[k][k];

 for(i=k+1; i < n; i++)

 W[i][k] = 0;

 for(i=k+1; i < n; i++)

 for(j=k+1; j <= n; j++)

 W[i][j] = W[i][j] - M[i][k]\*W[k][j];

 } /\* for \*/

 x[n-1] = W[n-1][n]/W[n-1][n-1];

 for(i=n-2; i >= 0; i--)

 {

 double temp;

 temp = W[i][n];

 for(k=i+1; k < n; k++)

 temp = temp - W[i][k]\*x[k];

 x[i] = temp/W[i][i];

 } /\* for \*/

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

 {

 free(M[i]);

 free(W[i]);

 } /\* for \*/

 free(M);

 free(W);

} /\* gaussian \*/

void read\_file(FILE \*fp, double \*A[], double \*b, int n)

{

 int i, j;

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

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

 fscanf(fp, "%lf", &A[i][j]);

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

 fscanf(fp, "%lf", &b[i]);

} /\* read\_file \*/

void print\_original\_system(double \*A[], double b[], int n)

{

 int i, j;

 printf("Original System:\n");

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

 {

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

 printf("%10.3lf", A[i][j]);

 printf("%10.3lf\n", b[i]);

 } /\* for \*/

} /\* print\_original\_system \*/

int main(int argc, char \*argv[])

{

 FILE \*fp;

 int i, n;

 double \*\*A, \*b, \*x;

 if (argc < 2)

 {

 fprintf(stderr, "Usage: gaussian filename\n");

 return 0;

 } /\* if \*/

 fp = fopen(argv[1], "rt");

 fscanf(fp, "%d", &n);

 A = (double \*\*)malloc((n+1)\*sizeof(double \*));

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

 A[i] = (double \*)malloc(n\*sizeof(double));

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

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

 read\_file(fp, A, b, n);

 gaussian(A, b, n, x);

 printf(" A x = b\n");

 print\_original\_system(A, b, n);

 print\_result(A, x, b, n);

 return 0;

} /\* main \*/

פלט ריצה:

E:\>GAUS1b.EXE mat2.txt

Stage 0:

 2.000 3.000 -1.000 0.000 5.000 17.000

 1.000 6.000 2.000 -3.000 -1.000 10.000

 2.000 0.000 1.000 4.000 -2.000 -24.000

 0.000 5.000 -2.000 1.000 3.000 -2.000

 3.000 1.000 4.000 -2.000 7.000 48.000

Stage 1:

 3.000 1.000 4.000 -2.000 7.000 48.000

 0.000 5.667 0.667 -2.333 -3.333 -6.000

 0.000 -0.667 -1.667 5.333 -6.667 -56.000

 0.000 5.000 -2.000 1.000 3.000 -2.000

 0.000 2.333 -3.667 1.333 0.333 -15.000

Stage 2:

 3.000 1.000 4.000 -2.000 7.000 48.000

 0.000 5.667 0.667 -2.333 -3.333 -6.000

 0.000 0.000 -1.588 5.059 -7.059 -56.706

 0.000 0.000 -2.588 3.059 5.941 3.294

 0.000 0.000 -3.941 2.294 1.706 -12.529

Stage 3:

 3.000 1.000 4.000 -2.000 7.000 48.000

 0.000 5.667 0.667 -2.333 -3.333 -6.000

 0.000 0.000 -3.941 2.294 1.706 -12.529

 0.000 0.000 0.000 1.552 4.821 11.522

 0.000 0.000 0.000 4.134 -7.746 -51.657

Stage 4:

 3.000 1.000 4.000 -2.000 7.000 48.000

 0.000 5.667 0.667 -2.333 -3.333 -6.000

 0.000 0.000 -3.941 2.294 1.706 -12.529

 0.000 0.000 0.000 4.134 -7.746 -51.657

 0.000 0.000 0.000 0.000 7.729 30.917

 A x = b

Original System:

 2.000 3.000 -1.000 0.000 5.000 17.000

 1.000 6.000 2.000 -3.000 -1.000 10.000

 2.000 0.000 1.000 4.000 -2.000 -24.000

 0.000 5.000 -2.000 1.000 3.000 -2.000

 3.000 1.000 4.000 -2.000 7.000 48.000

Solution X:

 X[0] X[1] X[2] X[3] X[4]

 1.000 -1.000 2.000 -5.000 4.000

 Verification:

 2.000 \* 1.000 + 3.000 \* -1.000 + -1.000 \* 2.000 + 0

.000 \* -5.000 + 5.000 \* 4.000 = 17.000 ?= 17.000

 1.000 \* 1.000 + 6.000 \* -1.000 + 2.000 \* 2.000 + -3

.000 \* -5.000 + -1.000 \* 4.000 = 10.000 ?= 10.000

 2.000 \* 1.000 + 0.000 \* -1.000 + 1.000 \* 2.000 + 4

.000 \* -5.000 + -2.000 \* 4.000 = -24.000 ?= -24.000

 0.000 \* 1.000 + 5.000 \* -1.000 + -2.000 \* 2.000 + 1

.000 \* -5.000 + 3.000 \* 4.000 = -2.000 ?= -2.000

 3.000 \* 1.000 + 1.000 \* -1.000 + 4.000 \* 2.000 + -2

.000 \* -5.000 + 7.000 \* 4.000 = 48.000 ?= 48.000

E:\>

התוכנית הבאה מממשת את האלימינציה של גאוס עם scaling בנוסף ל-partial pivoting:

/\* gaus2a.c - Partial pivoting with scaling \*/

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

void print\_result(double \*A[], double x[], double b[], int n)

{

 int i, j;

 double sum;

 printf("Solution X:\n");

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

 printf(" X[%d] ", i);

 printf("\n");

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

 printf(" %8.3lf ", x[i]);

 printf("\n Verification:\n");

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

 {

 sum = A[i][0] \* x[0];

 printf("%8.3lf \* %8.3lf ", A[i][0], x[0] );

 for(j=1; j < n; j++)

 {

 sum = sum + A[i][j] \* x[j];

 printf(" + %8.3lf \* %10.3lf ", A[i][j], x[j] );

 } /\* for \*/

 printf(" = %8.3lf ?= %8.3lf\n", sum, b[i]);

 } /\* for \*/

} /\* print\_result \*/

void swap\_rows(double \*A[], int n, int m1, int m2)

{

 int i;

 double temp;

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

 {

 temp = A[m1][i];

 A[m1][i] = A[m2][i];

 A[m2][i] = temp;

 } /\* for \*/

} /\* swap\_rows \*/

void gaussian(double \*A[], double b[], int n, double x[])

{

 int i, j, k, p;

 double \*\*W;

 double \*\*M;

 double MaxValue, ScaleValue, temp;

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

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

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

 W[i] = (double \*)malloc((n+1)\*sizeof(double));

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

 M[i] = (double \*)malloc(n\*sizeof(double));

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

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

 W[i][j] = A[i][j];

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

 W[i][n] = b[i];

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

 {

 ScaleValue = fabs(W[i][0]);

 for(j=1; j < n; j++)

 {

 temp = fabs(W[i][j]);

 if (temp > ScaleValue)

 ScaleValue = temp;

 }

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

 W[i][j] = W[i][j]/ScaleValue;

 } /\* for \*/

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

 {

 p = k;

 MaxValue = fabs(W[k][k]);

 for(i=k+1; i < n; i++)

 if (fabs(W[i][k]) > MaxValue)

 {

 p = i;

 MaxValue = fabs(W[i][k]);

 }

 swap\_rows(W, n, k, p);

 for(i=k+1; i < n; i++)

 M[i][k] = W[i][k]/W[k][k];

 for(i=k+1; i < n; i++)

 W[i][k] = 0;

 for(i=k+1; i < n; i++)

 for(j=k+1; j <= n; j++)

 W[i][j] = W[i][j] - M[i][k]\*W[k][j];

 } /\* for \*/

 x[n-1] = W[n-1][n]/W[n-1][n-1];

 for(i=n-2; i >= 0; i--)

 {

 double temp;

 temp = W[i][n];

 for(k=i+1; k < n; k++)

 temp = temp - W[i][k]\*x[k];

 x[i] = temp/W[i][i];

 } /\* for \*/

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

 {

 free(M[i]);

 free(W[i]);

 } /\* for \*/

 free(M);

 free(W);

} /\* gaussian \*/

void read\_file(FILE \*fp, double \*A[], double \*b, int n)

{

 int i, j;

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

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

 fscanf(fp, "%lf", &A[i][j]);

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

 fscanf(fp, "%lf", &b[i]);

} /\* read\_file \*/

void print\_original\_system(double \*A[], double b[], int n)

{

 int i, j;

 printf("Original System:\n");

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

 {

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

 printf("%10.3lf", A[i][j]);

 printf("%10.3lf\n", b[i]);

 } /\* for \*/

} /\* print\_original\_system \*/

int main(int argc, char \*argv[])

{

 FILE \*fp;

 int i, n;

 double \*\*A, \*b, \*x;

 if (argc < 2)

 {

 fprintf(stderr, "Usage: gaussian filename\n");

 return 0;

 } /\* if \*/

 fp = fopen(argv[1], "rt");

 fscanf(fp, "%d", &n);

 A = (double \*\*)malloc((n+1)\*sizeof(double \*));

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

 A[i] = (double \*)malloc(n\*sizeof(double));

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

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

 read\_file(fp, A, b, n);

 gaussian(A, b, n, x);

 printf(" A x = b\n");

 print\_original\_system(A, b, n);

 print\_result(A, x, b, n);

 return 0;

} /\* main \*/

פלט ריצה:

E:\>GAUS2a.EXE mat2.txt

A x = b

Original System:

 2.000 3.000 -1.000 0.000 5.000 17.000

 1.000 6.000 2.000 -3.000 -1.000 10.000

 2.000 0.000 1.000 4.000 -2.000 -24.000

 0.000 5.000 -2.000 1.000 3.000 -2.000

 3.000 1.000 4.000 -2.000 7.000 48.000

Solution X:

 X[0] X[1] X[2] X[3] X[4]

 1.000 -1.000 2.000 -5.000 4.000

 Verification:

 2.000 \* 1.000 + 3.000 \* -1.000 + -1.000 \* 2.000 + 0.000 \* -5.000 + 5.000 \* 4.000 = 17.000 ?= 17.000

 1.000 \* 1.000 + 6.000 \* -1.000 + 2.000 \* 2.000 + -3.000 \* -5.000 + -1.000 \* 4.000 = 10.000 ?= 10.000

 2.000 \* 1.000 + 0.000 \* -1.000 + 1.000 \* 2.000 + 4.000 \* -5.000 + -2.000 \* 4.000 = -24.000 ?= -24.000

 0.000 \* 1.000 + 5.000 \* -1.000 + -2.000 \* 2.000 + 1.000 \* -5.000 + 3.000 \* 4.000 = -2.000 ?= -2.000

 3.000 \* 1.000 + 1.000 \* -1.000 + 4.000 \* 2.000 + -2.000 \* -5.000 + 7.000 \* 4.000 = 48.000 ?= 48.000

התוכנית הבאה מממשת מוסיפה לתוכנית האחרונה גם הדפסות ביניים:

/\* gaus2b.c - Partial pivoting with scaling and stage printing \*/

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

void print\_stage(int stage, double \*W[], int n)

{

 int i, j;

 printf("Stage %d:\n", stage);

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

 {

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

 printf("%10.3lf", W[i][j]);

 printf("\n");

 } /\* for \*/

} /\* print\_stage \*/

void print\_result(double \*A[], double x[], double b[], int n)

{

 int i, j;

 double sum;

 printf("Solution X:\n");

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

 printf(" X[%d] ", i);

 printf("\n");

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

 printf(" %8.3lf ", x[i]);

 printf("\n Verification:\n");

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

 {

 sum = A[i][0] \* x[0];

 printf("%8.3lf \* %8.3lf ", A[i][0], x[0] );

 for(j=1; j < n; j++)

 {

 sum = sum + A[i][j] \* x[j];

 printf(" + %8.3lf \* %10.3lf ", A[i][j], x[j] );

 } /\* for \*/

 printf(" = %8.3lf ?= %8.3lf\n", sum, b[i]);

 } /\* for \*/

} /\* print\_result \*/

void swap\_rows(double \*A[], int n, int m1, int m2)

{

 int i;

 double temp;

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

 {

 temp = A[m1][i];

 A[m1][i] = A[m2][i];

 A[m2][i] = temp;

 } /\* for \*/

} /\* swap\_rows \*/

void gaussian(double \*A[], double b[], int n, double x[])

{

 int i, j, k, p;

 double \*\*W;

 double \*\*M;

 double MaxValue, ScaleValue, temp;

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

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

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

 W[i] = (double \*)malloc((n+1)\*sizeof(double));

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

 M[i] = (double \*)malloc(n\*sizeof(double));

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

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

 W[i][j] = A[i][j];

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

 W[i][n] = b[i];

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

 {

 ScaleValue = fabs(W[i][0]);

 for(j=1; j < n; j++)

 {

 temp = fabs(W[i][j]);

 if (temp > ScaleValue)

 ScaleValue = temp;

 }

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

 W[i][j] = W[i][j]/ScaleValue;

 } /\* for \*/

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

 {

 print\_stage(k, W, n);

 p = k;

 MaxValue = fabs(W[k][k]);

 for(i=k+1; i < n; i++)

 if (fabs(W[i][k]) > MaxValue)

 {

 p = i;

 MaxValue = fabs(W[i][k]);

 }

 swap\_rows(W, n, k, p);

 for(i=k+1; i < n; i++)

 M[i][k] = W[i][k]/W[k][k];

 for(i=k+1; i < n; i++)

 W[i][k] = 0;

 for(i=k+1; i < n; i++)

 for(j=k+1; j <= n; j++)

 W[i][j] = W[i][j] - M[i][k]\*W[k][j];

 } /\* for \*/

 x[n-1] = W[n-1][n]/W[n-1][n-1];

 for(i=n-2; i >= 0; i--)

 {

 double temp;

 temp = W[i][n];

 for(k=i+1; k < n; k++)

 temp = temp - W[i][k]\*x[k];

 x[i] = temp/W[i][i];

 } /\* for \*/

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

 {

 free(M[i]);

 free(W[i]);

 } /\* for \*/

 free(M);

 free(W);

} /\* gaussian \*/

void read\_file(FILE \*fp, double \*A[], double \*b, int n)

{

 int i, j;

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

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

 fscanf(fp, "%lf", &A[i][j]);

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

 fscanf(fp, "%lf", &b[i]);

} /\* read\_file \*/

void print\_original\_system(double \*A[], double b[], int n)

{

 int i, j;

 printf("Original System:\n");

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

 {

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

 printf("%10.3lf", A[i][j]);

 printf("%10.3lf\n", b[i]);

 } /\* for \*/

} /\* print\_original\_system \*/

int main(int argc, char \*argv[])

{

 FILE \*fp;

 int i, n;

 double \*\*A, \*b, \*x;

 if (argc < 2)

 {

 fprintf(stderr, "Usage: gaussian filename\n");

 return 0;

 } /\* if \*/

 fp = fopen(argv[1], "rt");

 fscanf(fp, "%d", &n);

 A = (double \*\*)malloc((n+1)\*sizeof(double \*));

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

 A[i] = (double \*)malloc(n\*sizeof(double));

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

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

 read\_file(fp, A, b, n);

 gaussian(A, b, n, x);

 printf(" A x = b\n");

 print\_original\_system(A, b, n);

 print\_result(A, x, b, n);

 return 0;

} /\* main \*/

פלט ריצה:

E:\>GAUS2b.EXE mat2.txt

Stage 0:

 0.400 0.600 -0.200 0.000 1.000 3.400

 0.167 1.000 0.333 -0.500 -0.167 1.667

 0.500 0.000 0.250 1.000 -0.500 -6.000

 0.000 1.000 -0.400 0.200 0.600 -0.400

 0.429 0.143 0.571 -0.286 1.000 6.857

Stage 1:

 0.500 0.000 0.250 1.000 -0.500 -6.000

 0.000 1.000 0.250 -0.833 0.000 3.667

 0.000 0.600 -0.400 -0.800 1.400 8.200

 0.000 1.000 -0.400 0.200 0.600 -0.400

 0.000 0.143 0.357 -1.143 1.429 12.000

Stage 2:

 0.500 0.000 0.250 1.000 -0.500 -6.000

 0.000 1.000 0.250 -0.833 0.000 3.667

 0.000 0.000 -0.550 -0.300 1.400 6.000

 0.000 0.000 -0.650 1.033 0.600 -4.067

 0.000 0.000 0.321 -1.024 1.429 11.476

Stage 3:

 0.500 0.000 0.250 1.000 -0.500 -6.000

 0.000 1.000 0.250 -0.833 0.000 3.667

 0.000 0.000 -0.650 1.033 0.600 -4.067

 0.000 0.000 0.000 -1.174 0.892 9.441

 0.000 0.000 0.000 -0.513 1.725 9.465

Stage 4:

 0.500 0.000 0.250 1.000 -0.500 -6.000

 0.000 1.000 0.250 -0.833 0.000 3.667

 0.000 0.000 -0.650 1.033 0.600 -4.067

 0.000 0.000 0.000 -1.174 0.892 9.441

 0.000 0.000 0.000 0.000 1.336 5.342

 A x = b

Original System:

 2.000 3.000 -1.000 0.000 5.000 17.000

 1.000 6.000 2.000 -3.000 -1.000 10.000

 2.000 0.000 1.000 4.000 -2.000 -24.000

 0.000 5.000 -2.000 1.000 3.000 -2.000

 3.000 1.000 4.000 -2.000 7.000 48.000

Solution X:

 X[0] X[1] X[2] X[3] X[4]

 1.000 -1.000 2.000 -5.000 4.000

 Verification:

 2.000 \* 1.000 + 3.000 \* -1.000 + -1.000 \* 2.000 + 0

.000 \* -5.000 + 5.000 \* 4.000 = 17.000 ?= 17.000

 1.000 \* 1.000 + 6.000 \* -1.000 + 2.000 \* 2.000 + -3

.000 \* -5.000 + -1.000 \* 4.000 = 10.000 ?= 10.000

 2.000 \* 1.000 + 0.000 \* -1.000 + 1.000 \* 2.000 + 4

.000 \* -5.000 + -2.000 \* 4.000 = -24.000 ?= -24.000

 0.000 \* 1.000 + 5.000 \* -1.000 + -2.000 \* 2.000 + 1

.000 \* -5.000 + 3.000 \* 4.000 = -2.000 ?= -2.000

 3.000 \* 1.000 + 1.000 \* -1.000 + 4.000 \* 2.000 + -2

.000 \* -5.000 + 7.000 \* 4.000 = 48.000 ?= 48.000

E:\>

התוכנית מממשת את האלימינציה של גאוס עם scaling ו-full pivoting:

/\* gaus3a.c - scaling and full pivoting \*/

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

void print\_result(double \*A[], double x[], double b[], int n)

{

 int i, j;

 double sum;

 printf("Solution X:\n");

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

 printf(" X[%d] ", i);

 printf("\n");

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

 printf(" %8.3lf ", x[i]);

 printf("\n Verification:\n");

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

 {

 sum = A[i][0] \* x[0];

 printf("%8.3lf \* %8.3lf ", A[i][0], x[0] );

 for(j=1; j < n; j++)

 {

 sum = sum + A[i][j] \* x[j];

 printf(" + %8.3lf \* %10.3lf ", A[i][j], x[j] );

 } /\* for \*/

 printf(" = %8.3lf ?= %8.3lf\n", sum, b[i]);

 } /\* for \*/

} /\* print\_result \*/

void swap\_rows(double \*A[], int n, int m1, int m2)

{

 int i;

 double temp;

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

 {

 temp = A[m1][i];

 A[m1][i] = A[m2][i];

 A[m2][i] = temp;

 } /\* for \*/

} /\* swap\_rows \*/

void swap\_cols(double \*A[], int n, int m1, int m2, int xindex[])

{

 int i, itemp;

 double dtemp;

 itemp = xindex[m1];

 xindex[m1] = xindex[m2];

 xindex[m2] = itemp;

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

 {

 dtemp = A[i][m1];

 A[i][m1] = A[i][m2];

 A[i][m2] = dtemp;

 } /\* for \*/

} /\* swap\_cols \*/

void gaussian(double \*A[], double b[], int n, double x[])

{

 int i, j, k, p, q;

 double \*\*W;

 double \*\*M;

 double \*y;

 int \*xindex;

 double MaxValue, ScaleValue, temp;

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

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

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

 W[i] = (double \*)malloc((n+1)\*sizeof(double));

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

 M[i] = (double \*)malloc(n\*sizeof(double));

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

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

 W[i][j] = A[i][j];

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

 W[i][n] = b[i];

 xindex = (int \*)malloc(n\*sizeof(int));

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

 xindex[i] = i;

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

 {

 ScaleValue = fabs(W[i][0]);

 for(j=1; j < n; j++)

 {

 temp = fabs(W[i][j]);

 if (temp > ScaleValue)

 ScaleValue = temp;

 }

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

 W[i][j] = W[i][j]/ScaleValue;

 } /\* for \*/

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

 {

 p = k;

 q = k;

 MaxValue = fabs(W[k][k]);

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

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

 if (fabs(W[i][j]) > MaxValue)

 {

 p = i;

 q = j;

 MaxValue = fabs(W[i][j]);

 } /\* if \*/

 swap\_cols(W, n, k, q, xindex);

 swap\_rows(W, n, k, p);

 for(i=k+1; i < n; i++)

 M[i][k] = W[i][k]/W[k][k];

 for(i=k+1; i < n; i++)

 W[i][k] = 0;

 for(i=k+1; i < n; i++)

 for(j=k+1; j <= n; j++)

 W[i][j] = W[i][j] - M[i][k]\*W[k][j];

 } /\* for \*/

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

 y[n-1] = W[n-1][n]/W[n-1][n-1];

 for(i=n-2; i >= 0; i--)

 {

 double temp;

 temp = W[i][n];

 for(k=i+1; k < n; k++)

 temp = temp - W[i][k]\*y[k];

 y[i] = temp/W[i][i];

 } /\* for \*/

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

 x[xindex[i]] = y[i];

 free(xindex);

 free(y);

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

 {

 free(M[i]);

 free(W[i]);

 } /\* for \*/

 free(M);

 free(W);

} /\* gaussian \*/

void read\_file(FILE \*fp, double \*A[], double \*b, int n)

{

 int i, j;

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

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

 fscanf(fp, "%lf", &A[i][j]);

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

 fscanf(fp, "%lf", &b[i]);

} /\* read\_file \*/

void print\_original\_system(double \*A[], double b[], int n)

{

 int i, j;

 printf("Original System:\n");

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

 {

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

 printf("%10.3lf", A[i][j]);

 printf("%10.3lf\n", b[i]);

 } /\* for \*/

} /\* print\_original\_system \*/

int main(int argc, char \*argv[])

{

 FILE \*fp;

 int i, n;

 double \*\*A, \*b, \*x;

 if (argc < 2)

 {

 fprintf(stderr, "Usage: gaussian filename\n");

 return 0;

 } /\* if \*/

 fp = fopen(argv[1], "rt");

 fscanf(fp, "%d", &n);

 A = (double \*\*)malloc((n+1)\*sizeof(double \*));

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

 A[i] = (double \*)malloc(n\*sizeof(double));

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

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

 read\_file(fp, A, b, n);

 gaussian(A, b, n, x);

 printf(" A x = b\n");

 print\_original\_system(A, b, n);

 print\_result(A, x, b, n);

 return 0;

} /\* main \*/

פלט ריצה:

E:\users\eytan\NA>GAUS3a.EXE mat2.txt

A x = b

Original System:

 2.000 3.000 -1.000 0.000 5.000 17.000

 1.000 6.000 2.000 -3.000 -1.000 10.000

 2.000 0.000 1.000 4.000 -2.000 -24.000

 0.000 5.000 -2.000 1.000 3.000 -2.000

 3.000 1.000 4.000 -2.000 7.000 48.000

Solution X:

 X[0] X[1] X[2] X[3] X[4]

 1.000 -1.000 2.000 -5.000 4.000

 Verification:

 2.000 \* 1.000 + 3.000 \* -1.000 + -1.000 \* 2.000 + 0.000 \* -5.000 + 5.000 \* 4.000 = 17.000 ?= 17.000

 1.000 \* 1.000 + 6.000 \* -1.000 + 2.000 \* 2.000 + -3.000 \* -5.000 + -1.000 \* 4.000 = 10.000 ?= 10.000

 2.000 \* 1.000 + 0.000 \* -1.000 + 1.000 \* 2.000 + 4.000 \* -5.000 + -2.000 \* 4.000 = -24.000 ?= -24.000

 0.000 \* 1.000 + 5.000 \* -1.000 + -2.000 \* 2.000 + 1.000 \* -5.000 + 3.000 \* 4.000 = -2.000 ?= -2.000

 3.000 \* 1.000 + 1.000 \* -1.000 + 4.000 \* 2.000 + -2.000 \* -5.000 + 7.000 \* 4.000 = 48.000 ?= 48.000

התוכנית הבאה מממשת מוסיפה לתוכנית האחרונה גם הדפסות ביניים:

/\* gaus3b.c - scaling and full pivoting with stage printing \*/

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

void print\_stage(int stage, double \*W[], int n)

{

 int i, j;

 printf("Stage %d:\n", stage);

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

 {

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

 printf("%10.3lf", W[i][j]);

 printf("\n");

 } /\* for \*/

} /\* print\_stage \*/

void print\_result(double \*A[], double x[], double b[], int n)

{

 int i, j;

 double sum;

 printf("Solution X:\n");

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

 printf(" X[%d] ", i);

 printf("\n");

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

 printf(" %8.3lf ", x[i]);

 printf("\n Verification:\n");

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

 {

 sum = A[i][0] \* x[0];

 printf("%8.3lf \* %8.3lf ", A[i][0], x[0] );

 for(j=1; j < n; j++)

 {

 sum = sum + A[i][j] \* x[j];

 printf(" + %8.3lf \* %10.3lf ", A[i][j], x[j] );

 } /\* for \*/

 printf(" = %8.3lf ?= %8.3lf\n", sum, b[i]);

 } /\* for \*/

} /\* print\_result \*/

void swap\_rows(double \*A[], int n, int m1, int m2)

{

 int i;

 double temp;

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

 {

 temp = A[m1][i];

 A[m1][i] = A[m2][i];

 A[m2][i] = temp;

 } /\* for \*/

} /\* swap\_rows \*/

void swap\_cols(double \*A[], int n, int m1, int m2, int xindex[])

{

 int i, itemp;

 double dtemp;

 itemp = xindex[m1];

 xindex[m1] = xindex[m2];

 xindex[m2] = itemp;

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

 {

 dtemp = A[i][m1];

 A[i][m1] = A[i][m2];

 A[i][m2] = dtemp;

 } /\* for \*/

} /\* swap\_cols \*/

void gaussian(double \*A[], double b[], int n, double x[])

{

 int i, j, k, p, q;

 double \*\*W;

 double \*\*M;

 double \*y;

 int \*xindex;

 double MaxValue, ScaleValue, temp;

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

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

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

 W[i] = (double \*)malloc((n+1)\*sizeof(double));

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

 M[i] = (double \*)malloc(n\*sizeof(double));

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

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

 W[i][j] = A[i][j];

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

 W[i][n] = b[i];

 xindex = (int \*)malloc(n\*sizeof(int));

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

 xindex[i] = i;

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

 {

 ScaleValue = fabs(W[i][0]);

 for(j=1; j < n; j++)

 {

 temp = fabs(W[i][j]);

 if (temp > ScaleValue)

 ScaleValue = temp;

 }

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

 W[i][j] = W[i][j]/ScaleValue;

 } /\* for \*/

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

 {

 print\_stage(k, W, n);

 p = k;

 q = k;

 MaxValue = fabs(W[k][k]);

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

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

 if (fabs(W[i][j]) > MaxValue)

 {

 p = i;

 q = j;

 MaxValue = fabs(W[i][j]);

 } /\* if \*/

 swap\_cols(W, n, k, q, xindex);

 swap\_rows(W, n, k, p);

 for(i=k+1; i < n; i++)

 M[i][k] = W[i][k]/W[k][k];

 for(i=k+1; i < n; i++)

 W[i][k] = 0;

 for(i=k+1; i < n; i++)

 for(j=k+1; j <= n; j++)

 W[i][j] = W[i][j] - M[i][k]\*W[k][j];

 } /\* for \*/

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

 y[n-1] = W[n-1][n]/W[n-1][n-1];

 for(i=n-2; i >= 0; i--)

 {

 double temp;

 temp = W[i][n];

 for(k=i+1; k < n; k++)

 temp = temp - W[i][k]\*y[k];

 y[i] = temp/W[i][i];

 } /\* for \*/

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

 x[xindex[i]] = y[i];

 free(xindex);

 free(y);

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

 {

 free(M[i]);

 free(W[i]);

 } /\* for \*/

 free(M);

 free(W);

} /\* gaussian \*/

void read\_file(FILE \*fp, double \*A[], double \*b, int n)

{

 int i, j;

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

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

 fscanf(fp, "%lf", &A[i][j]);

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

 fscanf(fp, "%lf", &b[i]);

} /\* read\_file \*/

void print\_original\_system(double \*A[], double b[], int n)

{

 int i, j;

 printf("Original System:\n");

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

 {

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

 printf("%10.3lf", A[i][j]);

 printf("%10.3lf\n", b[i]);

 } /\* for \*/

} /\* print\_original\_system \*/

int main(int argc, char \*argv[])

{

 FILE \*fp;

 int i, n;

 double \*\*A, \*b, \*x;

 if (argc < 2)

 {

 fprintf(stderr, "Usage: gaussian filename\n");

 return 0;

 } /\* if \*/

 fp = fopen(argv[1], "rt");

 fscanf(fp, "%d", &n);

 A = (double \*\*)malloc((n+1)\*sizeof(double \*));

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

 A[i] = (double \*)malloc(n\*sizeof(double));

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

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

 read\_file(fp, A, b, n);

 gaussian(A, b, n, x);

 printf(" A x = b\n");

 print\_original\_system(A, b, n);

 print\_result(A, x, b, n);

 return 0;

} /\* main \*/

פלט ריצה:

E:\>GAUS3b.EXE mat2.txt

Stage 0:

 0.400 0.600 -0.200 0.000 1.000 3.400

 0.167 1.000 0.333 -0.500 -0.167 1.667

 0.500 0.000 0.250 1.000 -0.500 -6.000

 0.000 1.000 -0.400 0.200 0.600 -0.400

 0.429 0.143 0.571 -0.286 1.000 6.857

Stage 1:

 1.000 0.600 -0.200 0.000 0.400 3.400

 0.000 1.100 0.300 -0.500 0.233 2.233

 0.000 0.300 0.150 1.000 0.700 -4.300

 0.000 0.640 -0.280 0.200 -0.240 -2.440

 0.000 -0.457 0.771 -0.286 0.029 3.457

Stage 2:

 1.000 0.600 -0.200 0.000 0.400 3.400

 0.000 1.100 0.300 -0.500 0.233 2.233

 0.000 0.000 0.068 1.136 0.636 -4.909

 0.000 0.000 -0.455 0.491 -0.376 -3.739

 0.000 0.000 0.896 -0.494 0.126 4.385

Stage 3:

 1.000 0.600 0.000 -0.200 0.400 3.400

 0.000 1.100 -0.500 0.300 0.233 2.233

 0.000 0.000 1.136 0.068 0.636 -4.909

 0.000 0.000 0.000 -0.484 -0.651 -1.619

 0.000 0.000 0.000 0.926 0.402 2.253

Stage 4:

 1.000 0.600 0.000 -0.200 0.400 3.400

 0.000 1.100 -0.500 0.300 0.233 2.233

 0.000 0.000 1.136 0.068 0.636 -4.909

 0.000 0.000 0.000 0.926 0.402 2.253

 0.000 0.000 0.000 0.000 -0.441 -0.441

 A x = b

Original System:

 2.000 3.000 -1.000 0.000 5.000 17.000

 1.000 6.000 2.000 -3.000 -1.000 10.000

 2.000 0.000 1.000 4.000 -2.000 -24.000

 0.000 5.000 -2.000 1.000 3.000 -2.000

 3.000 1.000 4.000 -2.000 7.000 48.000

Solution X:

 X[0] X[1] X[2] X[3] X[4]

 1.000 -1.000 2.000 -5.000 4.000

 Verification:

 2.000 \* 1.000 + 3.000 \* -1.000 + -1.000 \* 2.000 + 0.000 \* -5.000 + 5.000 \* 4.000 = 17.000 ?= 17.000

 1.000 \* 1.000 + 6.000 \* -1.000 + 2.000 \* 2.000 + -3.000 \* -5.000 + -1.000 \* 4.000 = 10.000 ?= 10.000

 2.000 \* 1.000 + 0.000 \* -1.000 + 1.000 \* 2.000 + 4.000 \* -5.000 + -2.000 \* 4.000 = -24.000 ?= -24.000

 0.000 \* 1.000 + 5.000 \* -1.000 + -2.000 \* 2.000 + 1.000 \* -5.000 + 3.000 \* 4.000 = -2.000 ?= -2.000

 3.000 \* 1.000 + 1.000 \* -1.000 + 4.000 \* 2.000 + -2.000 \* -5.000 + 7.000 \* 4.000 = 48.000 ?= 48.000

התוכנית הבאה היא תוכנית המבצעת אלימינציה של גאוס תוך הבאה בחשבון שהמטריצה עשויה להיות סינגולרית:

/\* gaus4a.c - Partial pivoting with scaling,

 detect singular matrices \*/

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

void print\_result(double \*A[], double x[], double b[], int n)

{

 int i, j;

 double sum;

 printf("Solution X:\n");

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

 printf(" X[%d] ", i);

 printf("\n");

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

 printf(" %8.3lf ", x[i]);

 printf("\n Verification:\n");

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

 {

 sum = A[i][0] \* x[0];

 printf("%8.3lf \* %8.3lf ", A[i][0], x[0] );

 for(j=1; j < n; j++)

 {

 sum = sum + A[i][j] \* x[j];

 printf(" + %8.3lf \* %10.3lf ", A[i][j], x[j] );

 } /\* for \*/

 printf(" = %8.3lf ?= %8.3lf\n", sum, b[i]);

 } /\* for \*/

} /\* print\_result \*/

void swap\_rows(double \*A[], int n, int m1, int m2)

{

 int i;

 double temp;

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

 {

 temp = A[m1][i];

 A[m1][i] = A[m2][i];

 A[m2][i] = temp;

 } /\* for \*/

} /\* swap\_rows \*/

int gaussian(double \*A[], double b[], int n, double x[])

{

 int i, j, k, p;

 double \*\*W;

 double \*\*M;

 double MaxValue, ScaleValue, temp;

 double epsilon = 0.0000001;

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

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

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

 W[i] = (double \*)malloc((n+1)\*sizeof(double));

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

 M[i] = (double \*)malloc(n\*sizeof(double));

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

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

 W[i][j] = A[i][j];

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

 W[i][n] = b[i];

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

 {

 ScaleValue = fabs(W[i][0]);

 for(j=1; j < n; j++)

 {

 temp = fabs(W[i][j]);

 if (temp > ScaleValue)

 ScaleValue = temp;

 }

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

 W[i][j] = W[i][j]/ScaleValue;

 } /\* for \*/

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

 {

 /\* Check if matrix is singular by

 testng if the current row is zero \*/

 MaxValue = 0;

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

 {

 temp = fabs(W[k][j]);

 if (MaxValue < temp)

 MaxValue = temp;

 } /\* for \*/

 if (MaxValue < epsilon) /\* Row of zeros? \*/

 return 0;

 /\* End of singular check \*/

 p = k;

 MaxValue = fabs(W[k][k]);

 for(i=k+1; i < n; i++)

 if (fabs(W[i][k]) > MaxValue)

 {

 p = i;

 MaxValue = fabs(W[i][k]);

 }

 swap\_rows(W, n, k, p);

 for(i=k+1; i < n; i++)

 M[i][k] = W[i][k]/W[k][k];

 for(i=k+1; i < n; i++)

 W[i][k] = 0;

 for(i=k+1; i < n; i++)

 for(j=k+1; j <= n; j++)

 W[i][j] = W[i][j] - M[i][k]\*W[k][j];

 } /\* for \*/

 x[n-1] = W[n-1][n]/W[n-1][n-1];

 for(i=n-2; i >= 0; i--)

 {

 double temp;

 temp = W[i][n];

 for(k=i+1; k < n; k++)

 temp = temp - W[i][k]\*x[k];

 x[i] = temp/W[i][i];

 } /\* for \*/

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

 {

 free(M[i]);

 free(W[i]);

 } /\* for \*/

 free(M);

 free(W);

 return 1;

} /\* gaussian \*/

void read\_file(FILE \*fp, double \*A[], double \*b, int n)

{

 int i, j;

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

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

 fscanf(fp, "%lf", &A[i][j]);

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

 fscanf(fp, "%lf", &b[i]);

} /\* read\_file \*/

void print\_original\_system(double \*A[], double b[], int n)

{

 int i, j;

 printf("Original System:\n");

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

 {

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

 printf("%10.3lf", A[i][j]);

 printf("%10.3lf\n", b[i]);

 } /\* for \*/

} /\* print\_original\_system \*/

int main(int argc, char \*argv[])

{

 FILE \*fp;

 int i, n;

 double \*\*A, \*b, \*x;

 if (argc < 2)

 {

 fprintf(stderr, "Usage: gaussian filename\n");

 return 0;

 } /\* if \*/

 fp = fopen(argv[1], "rt");

 fscanf(fp, "%d", &n);

 A = (double \*\*)malloc((n+1)\*sizeof(double \*));

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

 A[i] = (double \*)malloc(n\*sizeof(double));

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

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

 read\_file(fp, A, b, n);

 if (gaussian(A, b, n, x) == 0)

 {

 fprintf(stderr, "\n\nMATRIX IS SINGULAR\n\n");

 return 0;

 } /\* if \*/

 printf(" A x = b\n");

 print\_original\_system(A, b, n);

 print\_result(A, x, b, n);

 return 0;

} /\* main \*/

אם נריץ אותה על הקובץ הבא:

4

1 -1 1 -1

2 0 3 -1

3 1 -1 4

4 2 1 4

1

-3

2

2

נקבל פלט:

E:\>GAUS4A.EXE mat6.txt

MATRIX IS SINGULAR

התוכנית הבאה עושה כאחרונה אבל עם הדפסות ביניים:

/\* gaus4b.c - Partial pivoting with scaling and stage printing,

 detect singular matrices \*/

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

void print\_stage(int stage, double \*W[], int n)

{

 int i, j;

 printf("Stage %d:\n", stage);

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

 {

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

 printf("%10.3lf", W[i][j]);

 printf("\n");

 } /\* for \*/

} /\* print\_stage \*/

void print\_result(double \*A[], double x[], double b[], int n)

{

 int i, j;

 double sum;

 printf("Solution X:\n");

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

 printf(" X[%d] ", i);

 printf("\n");

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

 printf(" %8.3lf ", x[i]);

 printf("\n Verification:\n");

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

 {

 sum = A[i][0] \* x[0];

 printf("%8.3lf \* %8.3lf ", A[i][0], x[0] );

 for(j=1; j < n; j++)

 {

 sum = sum + A[i][j] \* x[j];

 printf(" + %8.3lf \* %10.3lf ", A[i][j], x[j] );

 } /\* for \*/

 printf(" = %8.3lf ?= %8.3lf\n", sum, b[i]);

 } /\* for \*/

} /\* print\_result \*/

void swap\_rows(double \*A[], int n, int m1, int m2)

{

 int i;

 double temp;

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

 {

 temp = A[m1][i];

 A[m1][i] = A[m2][i];

 A[m2][i] = temp;

 } /\* for \*/

} /\* swap\_rows \*/

int gaussian(double \*A[], double b[], int n, double x[])

{

 int i, j, k, p;

 double \*\*W;

 double \*\*M;

 double MaxValue, ScaleValue, temp;

 double epsilon = 0.0000001;

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

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

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

 W[i] = (double \*)malloc((n+1)\*sizeof(double));

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

 M[i] = (double \*)malloc(n\*sizeof(double));

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

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

 W[i][j] = A[i][j];

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

 W[i][n] = b[i];

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

 {

 ScaleValue = fabs(W[i][0]);

 for(j=1; j < n; j++)

 {

 temp = fabs(W[i][j]);

 if (temp > ScaleValue)

 ScaleValue = temp;

 }

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

 W[i][j] = W[i][j]/ScaleValue;

 } /\* for \*/

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

 {

 print\_stage(k, W, n);

 /\* Check if matrix is singular by

 testng if the current row is zero \*/

 MaxValue = 0;

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

 {

 temp = fabs(W[k][j]);

 if (MaxValue < temp)

 MaxValue = temp;

 } /\* for \*/

 if (MaxValue < epsilon) /\* Row of zeros? \*/

 return 0;

 /\* End of singular check \*/

 p = k;

 MaxValue = fabs(W[k][k]);

 for(i=k+1; i < n; i++)

 if (fabs(W[i][k]) > MaxValue)

 {

 p = i;

 MaxValue = fabs(W[i][k]);

 }

 swap\_rows(W, n, k, p);

 for(i=k+1; i < n; i++)

 M[i][k] = W[i][k]/W[k][k];

 for(i=k+1; i < n; i++)

 W[i][k] = 0;

 for(i=k+1; i < n; i++)

 for(j=k+1; j <= n; j++)

 W[i][j] = W[i][j] - M[i][k]\*W[k][j];

 } /\* for \*/

 x[n-1] = W[n-1][n]/W[n-1][n-1];

 for(i=n-2; i >= 0; i--)

 {

 double temp;

 temp = W[i][n];

 for(k=i+1; k < n; k++)

 temp = temp - W[i][k]\*x[k];

 x[i] = temp/W[i][i];

 } /\* for \*/

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

 {

 free(M[i]);

 free(W[i]);

 } /\* for \*/

 free(M);

 free(W);

 return 1;

} /\* gaussian \*/

void read\_file(FILE \*fp, double \*A[], double \*b, int n)

{

 int i, j;

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

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

 fscanf(fp, "%lf", &A[i][j]);

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

 fscanf(fp, "%lf", &b[i]);

} /\* read\_file \*/

void print\_original\_system(double \*A[], double b[], int n)

{

 int i, j;

 printf("Original System:\n");

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

 {

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

 printf("%10.3lf", A[i][j]);

 printf("%10.3lf\n", b[i]);

 } /\* for \*/

} /\* print\_original\_system \*/

int main(int argc, char \*argv[])

{

 FILE \*fp;

 int i, n;

 double \*\*A, \*b, \*x;

 if (argc < 2)

 {

 fprintf(stderr, "Usage: gaussian filename\n");

 return 0;

 } /\* if \*/

 fp = fopen(argv[1], "rt");

 fscanf(fp, "%d", &n);

 A = (double \*\*)malloc((n+1)\*sizeof(double \*));

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

 A[i] = (double \*)malloc(n\*sizeof(double));

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

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

 read\_file(fp, A, b, n);

 if (gaussian(A, b, n, x) == 0)

 {

 fprintf(stderr, "\n\nMATRIX IS SINGULAR\n\n");

 return 0;

 } /\* if \*/

 printf(" A x = b\n");

 print\_original\_system(A, b, n);

 print\_result(A, x, b, n);

 return 0;

} /\* main \*/

פלט ריצה על אותו קובץ:

E:\>GAUS4B.EXE mat6.txt

Stage 0:

 1.000 -1.000 1.000 -1.000 1.000

 0.667 0.000 1.000 -0.333 -1.000

 0.750 0.250 -0.250 1.000 0.500

 1.000 0.500 0.250 1.000 0.500

Stage 1:

 1.000 -1.000 1.000 -1.000 1.000

 0.000 0.667 0.333 0.333 -1.667

 0.000 1.000 -1.000 1.750 -0.250

 0.000 1.500 -0.750 2.000 -0.500

Stage 2:

 1.000 -1.000 1.000 -1.000 1.000

 0.000 1.500 -0.750 2.000 -0.500

 0.000 0.000 -0.500 0.417 0.083

 0.000 0.000 0.667 -0.556 -1.444

Stage 3:

 1.000 -1.000 1.000 -1.000 1.000

 0.000 1.500 -0.750 2.000 -0.500

 0.000 0.000 0.667 -0.556 -1.444

 0.000 0.000 0.000 0.000 -1.000

MATRIX IS SINGULAR

E:\>

התוכנית הבאה היא גרסת השדרוג של scaling ו-full pivoting:

/\* gaus5a.c - scaling and full pivoting with stage printing,

 detect singular matrices \*/

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

void print\_result(double \*A[], double x[], double b[], int n)

{

 int i, j;

 double sum;

 printf("Solution X:\n");

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

 printf(" X[%d] ", i);

 printf("\n");

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

 printf(" %8.3lf ", x[i]);

 printf("\n Verification:\n");

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

 {

 sum = A[i][0] \* x[0];

 printf("%8.3lf \* %8.3lf ", A[i][0], x[0] );

 for(j=1; j < n; j++)

 {

 sum = sum + A[i][j] \* x[j];

 printf(" + %8.3lf \* %10.3lf ", A[i][j], x[j] );

 } /\* for \*/

 printf(" = %8.3lf ?= %8.3lf\n", sum, b[i]);

 } /\* for \*/

} /\* print\_result \*/

void swap\_rows(double \*A[], int n, int m1, int m2)

{

 int i;

 double temp;

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

 {

 temp = A[m1][i];

 A[m1][i] = A[m2][i];

 A[m2][i] = temp;

 } /\* for \*/

} /\* swap\_rows \*/

void swap\_cols(double \*A[], int n, int m1, int m2, int xindex[])

{

 int i, itemp;

 double dtemp;

 itemp = xindex[m1];

 xindex[m1] = xindex[m2];

 xindex[m2] = itemp;

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

 {

 dtemp = A[i][m1];

 A[i][m1] = A[i][m2];

 A[i][m2] = dtemp;

 } /\* for \*/

} /\* swap\_cols \*/

int gaussian(double \*A[], double b[], int n, double x[])

{

 int i, j, k, p, q;

 double \*\*W;

 double \*\*M;

 double \*y;

 int \*xindex;

 double MaxValue, ScaleValue, temp;

 double epsilon = 0.0000001;

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

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

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

 W[i] = (double \*)malloc((n+1)\*sizeof(double));

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

 M[i] = (double \*)malloc(n\*sizeof(double));

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

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

 W[i][j] = A[i][j];

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

 W[i][n] = b[i];

 xindex = (int \*)malloc(n\*sizeof(int));

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

 xindex[i] = i;

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

 {

 ScaleValue = fabs(W[i][0]);

 for(j=1; j < n; j++)

 {

 temp = fabs(W[i][j]);

 if (temp > ScaleValue)

 ScaleValue = temp;

 }

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

 W[i][j] = W[i][j]/ScaleValue;

 } /\* for \*/

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

 {

 /\* Check if matrix is singular by

 testng if the current row is zero \*/

 MaxValue = 0;

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

 {

 temp = fabs(W[k][j]);

 if (MaxValue < temp)

 MaxValue = temp;

 } /\* for \*/

 if (MaxValue < epsilon) /\* Row of zeros? \*/

 return 0;

 /\* End of singular check \*/

 p = k;

 q = k;

 MaxValue = fabs(W[k][k]);

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

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

 if (fabs(W[i][j]) > MaxValue)

 {

 p = i;

 q = j;

 MaxValue = fabs(W[i][j]);

 } /\* if \*/

 swap\_cols(W, n, k, q, xindex);

 swap\_rows(W, n, k, p);

 for(i=k+1; i < n; i++)

 M[i][k] = W[i][k]/W[k][k];

 for(i=k+1; i < n; i++)

 W[i][k] = 0;

 for(i=k+1; i < n; i++)

 for(j=k+1; j <= n; j++)

 W[i][j] = W[i][j] - M[i][k]\*W[k][j];

 } /\* for \*/

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

 y[n-1] = W[n-1][n]/W[n-1][n-1];

 for(i=n-2; i >= 0; i--)

 {

 double temp;

 temp = W[i][n];

 for(k=i+1; k < n; k++)

 temp = temp - W[i][k]\*y[k];

 y[i] = temp/W[i][i];

 } /\* for \*/

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

 x[xindex[i]] = y[i];

 free(xindex);

 free(y);

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

 {

 free(M[i]);

 free(W[i]);

 } /\* for \*/

 free(M);

 free(W);

 return 1;

} /\* gaussian \*/

void read\_file(FILE \*fp, double \*A[], double \*b, int n)

{

 int i, j;

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

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

 fscanf(fp, "%lf", &A[i][j]);

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

 fscanf(fp, "%lf", &b[i]);

} /\* read\_file \*/

void print\_original\_system(double \*A[], double b[], int n)

{

 int i, j;

 printf("Original System:\n");

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

 {

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

 printf("%10.3lf", A[i][j]);

 printf("%10.3lf\n", b[i]);

 } /\* for \*/

} /\* print\_original\_system \*/

int main(int argc, char \*argv[])

{

 FILE \*fp;

 int i, n;

 double \*\*A, \*b, \*x;

 if (argc < 2)

 {

 fprintf(stderr, "Usage: gaussian filename\n");

 return 0;

 } /\* if \*/

 fp = fopen(argv[1], "rt");

 fscanf(fp, "%d", &n);

 A = (double \*\*)malloc((n+1)\*sizeof(double \*));

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

 A[i] = (double \*)malloc(n\*sizeof(double));

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

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

 read\_file(fp, A, b, n);

 if (gaussian(A, b, n, x) == 0)

 {

 fprintf(stderr, "\n\nMATRIX IS SINGULAR\n\n");

 return 0;

 } /\* if \*/

 printf(" A x = b\n");

 print\_original\_system(A, b, n);

 print\_result(A, x, b, n);

 return 0;

} /\* main \*/

פלט ריצה:

E:\>GAUS5A.EXE mat6.txt

MATRIX IS SINGULAR

גרסה כאחרונה עם הדפסות ביניים:

/\* gaus5b.c - scaling and full pivoting with stage printing,

 detect singular matrices \*/

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

void print\_stage(int stage, double \*W[], int n)

{

 int i, j;

 printf("Stage %d:\n", stage);

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

 {

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

 printf("%10.3lf", W[i][j]);

 printf("\n");

 } /\* for \*/

} /\* print\_stage \*/

void print\_result(double \*A[], double x[], double b[], int n)

{

 int i, j;

 double sum;

 printf("Solution X:\n");

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

 printf(" X[%d] ", i);

 printf("\n");

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

 printf(" %8.3lf ", x[i]);

 printf("\n Verification:\n");

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

 {

 sum = A[i][0] \* x[0];

 printf("%8.3lf \* %8.3lf ", A[i][0], x[0] );

 for(j=1; j < n; j++)

 {

 sum = sum + A[i][j] \* x[j];

 printf(" + %8.3lf \* %10.3lf ", A[i][j], x[j] );

 } /\* for \*/

 printf(" = %8.3lf ?= %8.3lf\n", sum, b[i]);

 } /\* for \*/

} /\* print\_result \*/

void swap\_rows(double \*A[], int n, int m1, int m2)

{

 int i;

 double temp;

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

 {

 temp = A[m1][i];

 A[m1][i] = A[m2][i];

 A[m2][i] = temp;

 } /\* for \*/

} /\* swap\_rows \*/

void swap\_cols(double \*A[], int n, int m1, int m2, int xindex[])

{

 int i, itemp;

 double dtemp;

 itemp = xindex[m1];

 xindex[m1] = xindex[m2];

 xindex[m2] = itemp;

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

 {

 dtemp = A[i][m1];

 A[i][m1] = A[i][m2];

 A[i][m2] = dtemp;

 } /\* for \*/

} /\* swap\_cols \*/

int gaussian(double \*A[], double b[], int n, double x[])

{

 int i, j, k, p, q;

 double \*\*W;

 double \*\*M;

 double \*y;

 int \*xindex;

 double MaxValue, ScaleValue, temp;

 double epsilon = 0.0000001;

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

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

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

 W[i] = (double \*)malloc((n+1)\*sizeof(double));

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

 M[i] = (double \*)malloc(n\*sizeof(double));

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

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

 W[i][j] = A[i][j];

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

 W[i][n] = b[i];

 xindex = (int \*)malloc(n\*sizeof(int));

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

 xindex[i] = i;

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

 {

 ScaleValue = fabs(W[i][0]);

 for(j=1; j < n; j++)

 {

 temp = fabs(W[i][j]);

 if (temp > ScaleValue)

 ScaleValue = temp;

 }

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

 W[i][j] = W[i][j]/ScaleValue;

 } /\* for \*/

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

 {

 print\_stage(k, W, n);

 /\* Check if matrix is singular by

 testng if the current row is zero \*/

 MaxValue = 0;

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

 {

 temp = fabs(W[k][j]);

 if (MaxValue < temp)

 MaxValue = temp;

 } /\* for \*/

 if (MaxValue < epsilon) /\* Row of zeros? \*/

 return 0;

 /\* End of singular check \*/

 p = k;

 q = k;

 MaxValue = fabs(W[k][k]);

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

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

 if (fabs(W[i][j]) > MaxValue)

 {

 p = i;

 q = j;

 MaxValue = fabs(W[i][j]);

 } /\* if \*/

 swap\_cols(W, n, k, q, xindex);

 swap\_rows(W, n, k, p);

 for(i=k+1; i < n; i++)

 M[i][k] = W[i][k]/W[k][k];

 for(i=k+1; i < n; i++)

 W[i][k] = 0;

 for(i=k+1; i < n; i++)

 for(j=k+1; j <= n; j++)

 W[i][j] = W[i][j] - M[i][k]\*W[k][j];

 } /\* for \*/

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

 y[n-1] = W[n-1][n]/W[n-1][n-1];

 for(i=n-2; i >= 0; i--)

 {

 double temp;

 temp = W[i][n];

 for(k=i+1; k < n; k++)

 temp = temp - W[i][k]\*y[k];

 y[i] = temp/W[i][i];

 } /\* for \*/

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

 x[xindex[i]] = y[i];

 free(xindex);

 free(y);

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

 {

 free(M[i]);

 free(W[i]);

 } /\* for \*/

 free(M);

 free(W);

 return 1;

} /\* gaussian \*/

void read\_file(FILE \*fp, double \*A[], double \*b, int n)

{

 int i, j;

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

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

 fscanf(fp, "%lf", &A[i][j]);

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

 fscanf(fp, "%lf", &b[i]);

} /\* read\_file \*/

void print\_original\_system(double \*A[], double b[], int n)

{

 int i, j;

 printf("Original System:\n");

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

 {

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

 printf("%10.3lf", A[i][j]);

 printf("%10.3lf\n", b[i]);

 } /\* for \*/

} /\* print\_original\_system \*/

int main(int argc, char \*argv[])

{

 FILE \*fp;

 int i, n;

 double \*\*A, \*b, \*x;

 if (argc < 2)

 {

 fprintf(stderr, "Usage: gaussian filename\n");

 return 0;

 } /\* if \*/

 fp = fopen(argv[1], "rt");

 fscanf(fp, "%d", &n);

 A = (double \*\*)malloc((n+1)\*sizeof(double \*));

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

 A[i] = (double \*)malloc(n\*sizeof(double));

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

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

 read\_file(fp, A, b, n);

 if (gaussian(A, b, n, x) == 0)

 {

 fprintf(stderr, "\n\nMATRIX IS SINGULAR\n\n");

 return 0;

 } /\* if \*/

 printf(" A x = b\n");

 print\_original\_system(A, b, n);

 print\_result(A, x, b, n);

 return 0;

} /\* main \*/

פלט ריצה:

Stage 0:

 1.000 -1.000 1.000 -1.000 1.000

 0.667 0.000 1.000 -0.333 -1.000

 0.750 0.250 -0.250 1.000 0.500

 1.000 0.500 0.250 1.000 0.500

Stage 1:

 1.000 -1.000 1.000 -1.000 1.000

 0.000 0.667 0.333 0.333 -1.667

 0.000 1.000 -1.000 1.750 -0.250

 0.000 1.500 -0.750 2.000 -0.500

Stage 2:

 1.000 -1.000 1.000 -1.000 1.000

 0.000 2.000 -0.750 1.500 -0.500

 0.000 0.000 -0.344 -0.312 0.188

 0.000 0.000 0.458 0.417 -1.583

Stage 3:

 1.000 -1.000 1.000 -1.000 1.000

 0.000 2.000 -0.750 1.500 -0.500

 0.000 0.000 0.458 0.417 -1.583

 0.000 0.000 0.000 -0.000 -1.000

MATRIX IS SINGULAR