Rozwiązania zestawu 1
Katarzyna Sowa

6N

Podana macierz jest trójdiagonalna trójkątna. W bibliotece GSl znaleziono funckję *gsl_linalg_solve_tridiag* przeznaczoną dla rozwiązywania równań z taką macierzą.
Kod programu:

```
#include <iostream>
#include <gsl/gsl_linalg.h>
using namespace std;

int main()
{
    gsl_vector diag; //diagonala
    gsl_vector e; // dolna krotka diagonala
    gsl_vector f; // gorna krotka diagonala
    gsl_vector b; // wyrazy wolne
    gsl_vector x;

    diag.size=b.size=x.size=7;
    e.size=f.size=6;
    diag.stride=e.stride=f.stride=b.stride=x.stride=1;

    // ustalamy wartosci

    double tabd[7] = {4,4,4,4,4,4,4};
    double tabef[6]={1,1,1,1,1,1};
    double tabb[7]={1,2,3,4,5,6,7};
    double tabx[7]={0};

    diag.data=tabd;
    f.data=e.data=tabef;
    b.data=tabb;
    x.data=tabx;

    gsl_linalg_solve_tridiag(&diag, &e, &f, &b, &x);

    for (int i=0; i<7; i++)
    {cout << "x" << i+1 << " = " << tabx[i] << endl; }

    return 0;}
```

Wynik:

```
x1 = 0.166789
x2 = 0.332842
x3 = 0.501841
x4 = 0.659794
x5 = 0.858984
x6 = 0.904271
x7 = 1.52393
```

9N

Macierz układu jest trójdiagoalna symetryczna. Z biblioteki GSL wybrano funkcję *gsl_linalg_solve_symm_cyc_tridiag*. Obydwie funkcje z zadań 6N i 9N działają na podstawie faktoryzacji Cholesky'ego.
Kod:

```
 #include <iostream>
#include <gsl/gsl_linalg.h>
using namespace std;

int main()
{
    gsl_vector diag; //diagonala
    gsl_vector e; // elementy poza diagonala
    gsl_vector b; // wyrazy wolne
    gsl_vector x;

    diag.size=e.size=b.size=x.size=7;
    diag.stride=e.stride=b.stride=x.stride=1;

    // ustalamy wartosci

    double tabd[7] = {4,4,4,4,4,4,4};
    double tabef[7]={1,1,1,1,1,1,1};
    double tabb[7]={1,2,3,4,5,6,7};
    double tabx[7]={0};

    diag.data=tabd;
    e.data=tabef;
    b.data=tabb;
    x.data=tabx;

    gsl_linalg_solve_symm_cyc_tridiag(&diag, &e, &b, &x);

    for (int i=0; i<7; i++)
    {cout << "x" << i+1 << " = " << tabx[i] << endl;}
    return 0;}
```

Wynik:

```
x1 = -0.260163
x2 = 0.447154
x3 = 0.471545
x4 = 0.666667
x5 = 0.861789
x6 = 0.886179
x7 = 1.5935
```

10N

Skorzystano z własności:

$$z_i = A^{-1} b_i$$
$$A z_i = b_i$$

ponieważ obliczaie macierzy odwrotnej byłoby zbyt kosztowne. Kod:

```
#include <iostream>
#include <gsl/gsl_linalg.h>
#include <cmath>
using namespace std;

int main()
{
    double normb12=0, normb34=0, normz12=0, normz34=0, norm1=0, norm2=0;

    double b11[5] = {-0.33388066, 1.08033290, -0.98559856, 1.31947922, -0.09473435};
    double b22[5] = {-0.33388066, 1.08033290, -0.98559855, 1.32655028, -0.10180541};
    double b33[5] = {0.72677951, 0.72677951, -0.27849178, 0.96592583, 0.96592583};
    double b44[5] = {0.73031505, 0.73031505, -0.27142071, 0.96946136, 0.96946136};

    double AA[5][5] = {{-116.66654, 583.33346, -333.33308, 100.00012, 100.00012},
{583.33346, -116.66654, -333.33308, 100.00012, 100.00012},
{-333.33308, -333.33308, 133.33383, 200.00025, 200.00025},
{100.00012, 100.00012, 200.00025, 50.000125, -649.99988},
{100.00012, 100.00012, 200.00025, -649.99988, 50.000125}};


  double temp[5][5];
  for (int i=0; i<5; i++)
    for (int j=0; j<5; j++) temp[i][j]=AA[i][j];
        gsl_vector b1, b2, b3, b4, z1, z2, z3, z4;
gsl_matrix A;

A.size1=A.size2=b1.size=b2.size=b3.size=b4.size=z1.size=z2.size=z3.size=z4.size=5;
A.tda=5;
b1.stride=b2.stride=b3.stride=b4.stride=z1.stride=z2.stride=z3.stride=z4.stride=1;

A.data=*AA;
 // z1
double z11[5]={0};
b1.data=b11;
z1.data=z11;
gsl_linalg_HH_solve(&A,&b1,&z1);
    for (int i=0; i<5; i++)
      for (int j=0; j<5; j++) AA[i][j]=temp[i][j];
cout << "z1 = (";
    for(int i=1; i<=5; i++) cout << fixed << z11[i-1] << " ";
cout << "]" << endl;
```

```
//z2
double z22[5]={0};
b2.data=b22;
z2.data=z22;
gsl_linalg_HH_solve(&A,&b2,&z2);
    for (int i=0; i<5; i++)
      for (int j=0; j<5; j++) AA[i][j]=temp[i][j];
cout << "z2 = (";
    for(int i=1; i<=5; i++) cout << fixed << z22[i-1] << " ";
cout << "]" << endl;
 //z3
double z33[5]={0};
b3.data=b33;
z3.data=z33;
gsl_linalg_HH_solve(&A,&b3,&z3);
    for (int i=0; i<5; i++)
      for (int j=0; j<5; j++) AA[i][j]=temp[i][j];
cout << "z3 = (";
    for(int i=1; i<=5; i++) cout << fixed << z33[i-1] << " ";
cout << "]" << endl;
 //z4
double z44[5]={0};
b4.data=b44;
z4.data=z44;
gsl_linalg_HH_solve(&A,&b4,&z4);
    for (int i=0; i<5; i++)
      for (int j=0; j<5; j++) AA[i][j]=temp[i][j];
cout << "z4 = (";
    for(int i=1; i<=5; i++) cout << fixed << z44[i-1] << " ";
cout << "]" << endl;

 // ||b1-b2||, ||b3-b4||

 for (int i=0; i<5; i++)
 {   normb12 += pow(b11[i-1]-b22[i-2],2);
     normb34 += pow(b33[i-1]-b44[i-1],2); }
  cout << "||b1-b2|| = " << sqrt(normb12) << endl;
  cout << "||b3-b4|| = " << sqrt(normb34) << endl;

  // ||z1-z2||, ||z3-z4||
  for (int i=0; i<5; i++)
  {    normz12 += pow(z11[i-1] - z22[i-1],2);
       normz34 += pow(z33[i-1] - z44[i-1],2); }
   cout << "||z1-z2|| = " << sqrt(normz12) << endl;
   cout << "||z3-z4|| = " << sqrt(normz34) << endl;

// ||z1-z2||/||b1-b2||, ||z3-z4||/||b3-b4||
   cout << "||z1-z2||/||b1-b2|| = " << sqrt(normz12)/sqrt(normb12) << endl;
   cout << "||z3-z4||/||b3-b4|| = " << sqrt(normz34)/sqrt(normb34) << endl;
      return 0;}
```

Wynik:

```
z1 = (0.001983 -0.000037 -0.000220 0.000241 -0.001780 ]
z2 = (0.001985 -0.000035 -0.000215 0.000253 -0.001787 ]
z3 = (354.885181 354.885181 709.768198 354.883432 354.883432 ]
z4 = (358.434025 358.434025 716.865884 358.432276 358.432276 ]
||b1-b2|| = 3.553301
||b3-b4|| = 0.009354
||z1-z2|| = 0.000014
||z3-z4|| = 9.389356
||z1-z2||/||b1-b2|| = 0.000004
||z3-z4||/||b3-b4|| = 1003.763915
```

Normy $\|\|b1 - b2\|\|$ i $\|\|b3 - b4\|\|$ różnią się aż o 3 rzędy wielkości. Podobnie jest z normami $\|\|z1 - z2\|\|$ i $\|\|z3 - z4\|\|$, które różnią się o 5 rzędów wielkości. Dlatego też przy dzieleniu norm przez siebie powstają liczby: mała i duża. Różnią się one o 9 rzędów wielkości.