Daniel Perry
cs6210 / Fall 2005
Homework 4

1. Exact solution is the following. (work by hand is attached)
   $x_2 = x_3 = (1 – (1+\epsilon^2)) / (2 - (1+\epsilon^2) - (1+\epsilon^2)^2)$
   $x_1 = 1 – x_2 - (1+\epsilon^2)*x_3$

(a)
The normal equation was solved using the Cholesky algorithm, with different values for ϵ, and compared with the exact solution computed with the above equations. The results are displayed below, first with float precision, then with double precision.

Results:

precision	epsilon (ϵ)	error from exact solution
float	1.00E+000	0, 2.9802322e-08, 2.9802322e-08,
	1.00E-001	2.5331974e-06, 1.2516975e-06, 1.2814999e-06
	1.00E-002	0.00024840236, 0.00012421608, 0.00012421608
	1.00E-003	1.4901161e-07, 1.4901161e-07, 2.9802322e-08
	1.00E-004	0.33333331, 0.33333334, nan
double	1.00E+000	8.3266727e-17, 1.110223e-16, 5.5511151e-17
	1.00E-001	5.9952043e-15, 2.9976022e-15, 2.9976022e-15
	1.00E-002	6.4054317e-13, 3.2018832e-13, 3.2035485e-13
	1.00E-003	3.4387881e-11, 1.7193913e-11, 1.7194024e-11
	1.00E-004	1.6660573e-09, 8.3302865e-10, 8.3302865e-10
	1.00E-005	5.5511151e-17, 0, 0

double	1.00E-006	5.5511151e-17, 5.5511151e-17, 5.5511151e-17
	1.00E-007	0.0050125313, 0.0025062657, 0.0025062657
	1.00E-008	0.33333333, 0.33333333, nan

Analysis:

The smallest value for epsilon I could use and still get a valid result, was 10e-4 for float precision, and 10e-8 for double precision. Once I stepped onto epsilon >= 10e-5 (or 10e-9 for double), I was unable to obtain valid results. When I implemented Cholesky factorization in homework 2 (I used the same code, modified slightly for this assignment) I check for 0's along the diagonal. For some reason, when I pass those numbers for epsilon, it will warn me of zeros on the diagonal. I think this is due in part because the matrix I am using is entirely 1's and ($1+\epsilon^2$)'s and when ϵ^2 get's small enough, then when we do the cholesky factorization, that extra ϵ^2 is lost, and for example instead of 1-($1+\epsilon^2$), it is more like 1-1.

Although there seemed to be some pattern in the correctness of the result with the change in epsilon, I couldn't find a correlation between the machine epsilon and this correctness or the failure. I was using machine epsilon = 2^{-23} for float, and machine epsilon = 2^{-52} for double (machine epsilon = 2^{-t}, where t is the size of the fraction portion of IEEE representation, and 23 and 52 were what I thought were the sizes of that for float and double respectively). However, as you can see from my results, I didn't really approach either of those numbers.

Float correctness seemed to worsen until epsilon = 10e-4, where it became better. Likewise, Double correctness seemed to worsen until epsilon = 10e-8, where it also became suddenly much better. I'm not exactly sure why that occurred.

(b)

Similar to the Cholesky version, I also used householder QR decomposition to solve the linear least squares problem, with different values for epsilon. The

Daniel Perry
cs6210 / Fall 2005
Homework 4

Results:

precision	epsilon (∊)	error from exact solution
float	1.00E+000	.9604645e-08, 4.4703484e-08, 0
	1.00E-001	5.3942204e-06, 2.682209e-06, 2.6524067e-06
	1.00E-002	0.00016257167, 8.1241131e-05, 8.1270933e-05
	1.00E-003	0.020007849, 0.010003865, 0.010003924
	1.00E-004	(error)
double	1.00E+000	5.5511151e-17, 2.7755576e-17, 0
	1.00E-001	1.2434498e-14, 6.1062266e-15, 6.1617378e-15
	1.00E-002	1.0556556e-12, 5.2763349e-13, 5.2791105e-13
	1.00E-003	6.0312866e-12, 3.0155878e-12, 3.0158098e-12
	1.00E-004	3.7007724e-09, 1.8503862e-09, 1.8503861e-09
	1.00E-005	2.8856932e-07, 1.4428466e-07, 1.4428466e-07
	1.00E-006	5.1316615e-05, 2.5658308e-05, 2.5658308e-05

double	1.00E-007	0.0090326802, 0.0045163401, 0.0045163401
	1.00E-008	(error)

Analysis:

The householder method had very similar results to the Cholesky method. It also failed for floats when epsilon >= 10e-5, and for doubles when epsilon >= 10e-9. However, while its error continually worsened like Cholesky, it did not have a sudden jump to better correctness like in the Cholesky instance.

2. Power Method

(i) – This implementation was fairly straightforward. The results are below:
iterations required to converge: 5
(where iteration was stopped when (л$_{n+1}$ - л$_{n}$) < 10^{-7})
lambda1=10.995 (eigenval)
x1=[0.37149,0.74298,0.556755,] (eigenvec)

a simple check to see if A*x1 = lambda1*x1:
A*x1=[4.08543,8.17086,6.1291,]
lambda1*x1=[4.08455,8.16909,6.12155,]

(ii)– householder deflation. This was a little more tricky to implement. The results are below:

```
H1*A*H1 matrix(deflated):
10.6301, 5.07989, 4.42139,
0, -3.39731, -1.08418,
0, 4.08081, -0.525253,

B matrix:
-3.39731,-1.08418,
4.08081,-0.525253,
```

Daniel Perry
cs6210 / Fall 2005
Homework 4

RESULTS:
iterations required to converge = 396893
lambda1=-3.21647 (second eigenvector of A and B)
x1=[-0.412744,0.910847,] (eigenvector of B)
bT=[5.07989,4.42139,]

x1=[0.140802,0.427966,-0.944439] (second eigenvector of A)

A simple check was done to check if A*x1 = lambda1*x1:
A*x1=[-0.323375,-1.08584,2.04576,]
lambda1*x1=[-0.452886,-1.37654,3.03776,]

(iii)– Use a general library to compute all eigenvalues and eigenvectors.
I tried 3 different libraries.
First, I used a TI-89 calculator, it gave the following results:
eigenvalues=(-2,11,-3)
eigenvectors:
x0=[-.182574, -.365148, .91287]
x1=[-.371391, -.742781, -.557086]
x2=[0, .5547, -.83205]

I next tried using LAPACK. It gave similar results for eigenvalues -2, and -3, but would return a NaN for the third. I am not very familiar with the system, so it may have been due to user error. The corresponding eigenvectors were also similar.

Finally I used matlab. The resulting printout is attached. It gave the same results (some different signs... I'm not sure why, but they probably don't matter a whole lot).

Analysis:

As compared with the general library routines, my power method implementation did pretty well. It obtained the same results of the eigenvalue 11 and it's corresponding eigenvector. However, the deflation and subsequent power method to obtain the next eigenvalue and eigenvector had a lot of error. I assume that is due to the number of iterations the submatrix of HAH^{-1} required to converge (396893!)

Daniel Perry
cs6210 / Fall 2005
Homework 4

The code I wrote for this assignment will follow. You can also find it at http://www.cs.utah.edu/~dperry/classes/cs6210/ along with this writeup.

Code:
1. Cholesky Solution to Normal Equations
2. Householder Solution
3. Power Method
4. Deflation and power method

Following those:
5. Written solution to 1.
6. Matlab printout

Daniel Perry
cs6210 / Fall 2005
Homework 4

```
/* Daniel Perry
 * cs6210 - homework 4 problem 1 - implement cholesky for normal
equatins:
 *
 */

typedef double myreal;

#include <iostream>
using namespace std;

#include <math.h>
#include <stdlib.h>
//#include "matrix.h"

// decomposition
void cholesky( myreal ** m , int n );
void householder( myreal ** m , myreal ** v, int n );

// helper functions
void hilbert( myreal ** m , int n );
void exactanswer( myreal ** m , myreal * b , int n );
void makeSPDmatrix( myreal ** m , int n );
void makematrix( myreal ** m , int n );


// evaluation functions
void FSA( myreal ** m , myreal * x , myreal * b , int n );
void BSA( myreal ** m , myreal * x , myreal * b , int n );

//matrix functions
void transpose( myreal ** m , int n );
void mult( myreal ** m , myreal ** m2 , int n );
void mult2( myreal ** m , myreal * v , int n );

myreal exact_num( int n ){
  switch( n ){
  case 1:
    return 10e-1;
  case 2:
    return 10e-2;
  case 3:
    return 10e-3;
  case 4:
    return 10e-4;
  case 5:
    return 10e-5;
  case 6:
    return 10e-6;
  case 7:
    return 10e-7;
  case 8:
    return 10e-8;
  case 9:
    return 10e-9;
  }
}

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

  // cout.setf(ios_base::scientific);
  cout.precision( 8 );

  int exponent = 4;
  if( argc == 2 ){
    exponent = atoi( argv[1] );
  }

  int n = 3;
  myreal epsilon = exact_num( exponent );
  myreal epsilon2 = epsilon*epsilon;

  // special vals for checking against:
  double mach_eps_float = 2e-23;
  double sqrt_mach_eps_float = sqrt( mach_eps_float );
  double mach_eps_double = 2e-52;
  double sqrt_mach_eps_double = sqrt( mach_eps_double );

  if( epsilon < mach_eps_float ){
    cout<<endl<<"epsilon is < mach_eps_float!!!!!!!!!!"<<endl<<endl;
  }
  if( epsilon < sqrt_mach_eps_float ){
    cout<<endl<<"epsilon is <
sqrt_mach_eps_float!!!!!!!!!!"<<endl<<endl;
  }
  if( epsilon < mach_eps_double ){
    cout<<endl<<"epsilon is < mach_eps_double!!!!!!!!!!"<<endl<<endl;
  }
  if( epsilon < sqrt_mach_eps_double ){
    cout<<endl<<"epsilon is <
sqrt_mach_eps_double!!!!!!!!!!"<<endl<<endl;
  }


  // declaration, allocation:
  myreal ** m = new myreal * [n], // matrix
    ** m_orig = new myreal * [n], // backup of matrix.
    * b = new myreal[n],  // right hand side vector
    * y = new myreal[n],  // temporary solution vector.
    * x = new myreal[n],  // solution vector
    * b_orig = new myreal[n];  // backup of b
  for( int i=0 ; i<n ; i++ ){
    m[i] = new myreal[n];
    m_orig[i] = new myreal[n];
  }


  /*
  // note: m[rows][columns]
  // good cholesky test matrix... (answer given in text)
  m[0][0] = 3; m[0][1] = -1; m[0][2] = -1;
  m[1][0] = -1; m[1][1] = 3; m[1][2] = -1;
  m[2][0] = -1; m[2][1] = -1; m[2][2] = 3;
  */

  // or make m a hilbert matrix of size n:
  //hilbert( m , n );
  // or make an SPD matrix:
  //makematrix( m , n );

  m[0][0] = (1+epsilon2); m[0][1] = 1; m[0][2] = 1;
  m[1][0] = 1; m[1][1] = (1+epsilon2); m[1][2] = 1;
  m[2][0] = 1; m[2][1] = 1; m[2][2] = (1+epsilon2);

  cout<<"A matrix:"<<endl;
  for( int i = 0 ; i < n ; i++ ){
    for( int j = 0 ; j < n ; j++ ){
      cout<<m[i][j]<<", ";
    }
    cout<<endl;
  }
  cout<<endl;

  // make a backup of m:
  for( int i=0 ; i<n ; i++ ){
    for( int j=0 ; j<n ; j++ ){
      m_orig[i][j] = m[i][j];
    }
  }

  // for testing, figure out what b should be to get x = [1,1,1,...,1]
  //exactanswer( m , b , n );

  // set b=[1,1,1]
  for( int i=0 ; i<n ; i++ ){
    b[i] = 1;
  }
  cout<<"b = ";
  for( int i=0 ; i<n ; i++ ){
    cout<<b[i]<<", ";
  }
  cout<<endl;

  //copy b:
  for( int i=0 ; i<n ; i++ ){
    b_orig[i] = b[i];
  }


  // decompose into L:
  cholesky( m , n );

  //solve using FSA (for lower triangle systems)
  FSA( m , y , b , n );

  cout<<"y = ";
  for( int i=0 ; i<n ; i++ ){
    cout<<y[i]<<", ";
  }
  cout<<endl;


  //transpose m ( to becom L^(-1) )
  transpose( m , n );

  //solve using BSA (for upper triangle systems)
  BSA( m , x , y , n );

  cout<<endl<<"x = ";
  for( int i=0 ; i<n ; i++ ){
    cout<<x[i]<<", ";
  }
  cout<<endl;

  myreal correct_x[3];

  correct_x[1] = correct_x[2] = (1-(1+epsilon2))/(2-(1+epsilon2)-
(1+epsilon2)*(1+epsilon2));
  correct_x[0] = 1 - correct_x[1] - (1+epsilon2)*correct_x[2];

  cout<<"correct_x = ";
  for( int i=0 ; i<3 ; i++ ){
    cout<<correct_x[i]<<", ";
  }
  cout<<endl;
  cout<<"error_x = ";
  for( int i=0 ; i<n ; i++ ){
    cout<<fabs(x[i]-correct_x[i])<<", ";
  }
  cout<<endl;

  return 0;
}


//////////////////////////////////////////////
// Implementation of Cholesky's Algorithm:
//////////////////////////////////////////////
void cholesky( myreal ** m , int n ){
  /*
  cout<<"before cholesky's algorithm:"<<endl;
  for( int i = 0 ; i < n ; i++ ){
    for( int j = 0 ; j < n ; j++ ){
      cout<<m[i][j]<<", ";
    }
    cout<<endl;
  }
  */

  // Cholesky's algorithm, as described on p. 86 of "Scientific
Computing" by Heath.

  for( int k=0 ; k < n ; k++ ){  // loop over columns
    /*
    cout<<"A matrix (k="<<k<<"):"<<endl;
    for( int i = 0 ; i < n ; i++ ){
      for( int j = 0 ; j < n ; j++ ){
      cout<<m[i][j]<<", ";
      }
      cout<<endl;
    }
    cout<<endl;
    */

    m[k][k] = sqrt( m[k][k] );

    for( int i=k+1 ; i < n ; i++ ){  // scale current column
      m[i][k] = m[i][k] / m[k][k];
    }

    for( int j=k+1 ; j < n ; j++ ){  // from each remaining column,
subtract multiple of current column.
      for( int i=k+1 ; i < n ; i++ ){
```

```
      m[i][j] = m[i][j] - ( m[i][k] * m[j][k] ) ;
      }
    }
  }

  //fill in zeros as appropriate (for sanity check):
  for( int c=0 ; c<n ; c++ ){
    for( int r=c-1 ; r>=0 ; r-- ){
      m[r][c] = 0;
    }
  }

  /*
  cout<<"after cholesky's algorithm:"<<endl;
  for( int i = 0 ; i < n ; i++ ){
    for( int j = 0 ; j < n ; j++ ){
      cout<<m[i][j]<<", ";
    }
    cout<<endl;
  }
  cout<<endl;
  */
}

// forward substitution to solve LOWER triangle systems. (from p. 65 in
Heath)
void FSA( myreal ** m , myreal * x , myreal * b , int n){

  for( int j=0 ; j<n ; j++ ){  // loop over columns
    if( m[j][j] == 0 ){       // stop if matrix is singular
      cerr<<"**********"<<endl<<"FSA: matrix is singular,
stopping"<<endl<<"( element m["<<j<<"]["<<j<<"]=0 )
"<<endl<<"*************"<<endl;
      return;
    }
    x[j] = b[j] / m[j][j];  // compute solution component
    for( int i=j+1 ; i < n ; i++ ){  //update right-hand side.
      b[i] = b[i] - m[i][j]*x[j];
    }
  }

}


// backward substitution to solve UPPER triangle systems (from p.66 in
Heath)
void BSA( myreal ** m , myreal * x , myreal * b , int n){

  for( int j=(n-1) ; j >= 0 ; j-- ){  //loop backwards over columns
```

```
    if( m[j][j] == 0 ){  // stop if matrix is singular
      cerr<<"**********"<<endl<<"BSA: matrix is singular,
stopping"<<endl<<"( element m["<<j<<"]["<<j<<"]=0 )
"<<endl<<"*************"<<endl;
      return;
    }
    x[j] = b[j] / m[j][j];  // compute solution component.
    for( int i=0 ; i<= j-1 ; i++ ){  // update right hand side.
      b[i] = b[i] - m[i][j]*x[j];
    }
  }

}

// transposes a matrix in itself:
void transpose( myreal ** m , int n ){

  myreal ** temp = new myreal*[n];
  for( int i=0 ; i<n ; i++ ){
    temp[i] = new myreal[n];
    for( int j=0 ; j<n ; j++ ){
      temp[i][j] = m[i][j];
    }
  }


  /*
  cout<<"after temp matrix copied over:"<<endl;
  for( int i = 0 ; i < n ; i++ ){
    for( int j = 0 ; j < n ; j++ ){
      cout<<temp[i][j]<<", ";
    }
    cout<<endl;
  }
  */

  for( int i=0 ; i<n ; i++ ){
    for( int j=0 ; j<n ; j++ ){
      m[i][j] = temp[j][i];
    }
  }

  /*
  cout<<"after transpose matrix:"<<endl;
  for( int i = 0 ; i < n ; i++ ){
    for( int j = 0 ; j < n ; j++ ){
      cout<<m[i][j]<<", ";
    }
    cout<<endl;
  }
  */

  for( int r=0 ; r < n ; r++ ){
    delete[] temp[r];
  }
  delete[] temp;
}

// matrix * matrix multiplication
void mult( myreal ** m1 , myreal ** m2 , int n ){
  myreal ** temp = new myreal*[n];
  for( int i=0 ; i<n ; i++ ){
    temp[i] = new myreal[n];
    for( int j=0; j<n ; j++ ){
      temp[i][j] = 0;
      for( int k=0; k<n ; k++ ){
      temp[i][j] += m1[i][k]*m2[k][j];
      }
    }
  }

  for( int i=0 ; i<n ; i++ ){
    for( int j=0 ; j<n ; j++ ){
      m2[i][j] = temp[i][j];
    }
  }

  for( int r=0 ; r < n ; r++ ){
    delete[] temp[r];
  }
  delete[] temp;

}



// matrix * vector multiplication
void mult2( myreal ** m , myreal * v , int n ){

  myreal * temp = new myreal[n];

  for( int i=0 ; i<n ; i++ ){
    temp[i] = 0;
    for( int j=0 ; j<n ; j++ ){
      temp[i] += m[i][j]*v[j];
    }
  }
  for( int i=0; i<n ; i++ ){
    v[i] = temp[i];
  }

  delete[] temp;

}
```

```
/* Daniel Perry
 * cs6210 - homework 4 problem 1b - implement householder:
 *
 */

typedef double myreal;

#include <iostream>
using namespace std;

#include <math.h>
//#include "matrix.h"

// decomposition
void cholesky( myreal ** m , int n );
void householder( myreal ** m , myreal ** v, int n );

// helper functions
void hilbert( myreal ** m , int n );
void exactanswer( myreal ** m , myreal * b , int n );
void makeSPDmatrix( myreal ** m , int n );
void makematrix( myreal ** m , int n );

// evaluation functions
void FSA( myreal ** m , myreal * x , myreal * b , int n );
void BSA( myreal ** m , myreal * x , myreal * b , int n );
void householder_solver( myreal ** v , myreal * y , myreal * b , int
n ); // detailed description above definition.

//matrix functions
void transpose( myreal ** m , int n );
void mult( myreal ** m , myreal ** m2 , int n );
void mult2( myreal ** m , myreal * v , int n );

myreal exact_num( int n ){
  switch( n ){
  case 1:
    return 10e-1;
  case 2:
    return 10e-2;
  case 3:
    return 10e-3;
  case 4:
    return 10e-4;
  case 5:
    return 10e-5;
  case 6:
    return 10e-6;
  case 7:
    return 10e-7;
  case 8:
    return 10e-8;
  case 9:
    return 10e-9;
  }
}

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

  cout.precision( 8 );

  int exponent = 4;
  if( argc == 2 ){
    exponent = atoi( argv[1] );
  }

  int n = 3;
  myreal epsilon = exact_num( exponent );
  myreal epsilon2 = epsilon*epsilon;

  // declaration, allocation:
  myreal ** m = new myreal * [n], // matrix
    ** m_orig = new myreal * [n], // backup of matrix.
    ** v = new myreal * [n], // backup of matrix.
    * b = new myreal[n],  // right hand side vector
    * y = new myreal[n],  // temporary solution vector.
    * x = new myreal[n],  // solution vector
    * b_orig = new myreal[n];  // backup of b
  for( int i=0 ; i<n ; i++ ){
    m[i] = new myreal[n];
    m_orig[i] = new myreal[n];
    v[i] = new myreal[n];
  }

   // special vals for checking against:
  double mach_eps_float = 2e-23;
  double sqrt_mach_eps_float = sqrt( mach_eps_float );
  double mach_eps_double = 2e-52;
  double sqrt_mach_eps_double = sqrt( mach_eps_double );

  cout<<"eps:"<<epsilon<<endl;
  cout<<"eps2:"<<epsilon<<endl;
  cout<<"mach_eps:"<<mach_eps_float<<endl;
  cout<<"sqrt_mach_eps:"<<sqrt_mach_eps_float<<endl;
  cout<<"mach_eps_dbl:"<<mach_eps_double<<endl;
  cout<<"sqrt_mach_eps:_dbl"<<sqrt_mach_eps_double<<endl;

  if( epsilon < mach_eps_float ){
    cout<<endl<<"epsilon is < mach_eps_float!!!!!!!!!!"<<endl<<endl;
  }
  if( epsilon < sqrt_mach_eps_float ){
    cout<<endl<<"epsilon is <
sqrt_mach_eps_float!!!!!!!!!!"<<endl<<endl;
  }
  if( epsilon < mach_eps_double ){
    cout<<endl<<"epsilon is < mach_eps_double!!!!!!!!!!"<<endl<<endl;
  }
  if( epsilon < sqrt_mach_eps_double ){
    cout<<endl<<"epsilon is <
sqrt_mach_eps_double!!!!!!!!!!"<<endl<<endl;
  }



  m[0][0] = (1+epsilon2); m[0][1] = 1; m[0][2] = 1;
  m[1][0] = 1; m[1][1] = (1+epsilon2); m[1][2] = 1;
  m[2][0] = 1; m[2][1] = 1; m[2][2] = (1+epsilon2);

  cout<<"used this matrix:"<<endl;
  for( int i = 0 ; i < n ; i++ ){
    for( int j = 0 ; j < n ; j++ ){
      cout<<m[i][j]<<", ";
    }
    cout<<endl;
  }
  cout<<endl;

  // make a backup of m:
  for( int i=0 ; i<n ; i++ ){
    for( int j=0 ; j<n ; j++ ){
      m_orig[i][j] = m[i][j];
    }
  }


  /*
  // b-vector to go with the 6x6 above from text:
  b[0]=1237;
  b[1]=1941;
  b[2]=2417;
  b[3]=711;
  b[4]=1177;
  b[5]=475;
  */

  // make b=[1,1,1]
  for( int i=0 ; i<n ; i++ ){
    b[i] = 1;
  }

  cout<<"b = ";
  for( int i=0 ; i<n ; i++ ){
    cout<<b[i]<<", ";
  }
  cout<<endl;


  //copy b:
  for( int i=0 ; i<n ; i++ ){
    b_orig[i] = b[i];
  }

  // decompose into Q (vectors v) and R (m):
  householder( m , v , n );


  // solve Qy=b, by using the individual matrices H, created from the v
vectors:
  householder_solver( v , y , b , n );

  cout<<"y = ";
  for( int i=0 ; i<n ; i++ ){
    cout<<y[i]<<", ";
  }
  cout<<endl;


  //transpose m ( to becom L^(-1) )
  //transpose( m , n );

  // Now solve Rx=y, using the BSA (because R is upper) - R = m.
  //solve using BSA (for upper triangle systems)
  BSA( m , x , y , n );

  cout<<endl<<"x = ";
  for( int i=0 ; i<n ; i++ ){
    cout<<x[i]<<", ";
  }
  cout<<endl;

   myreal correct_x[3];

  correct_x[1] = correct_x[2] = (1-(1+epsilon2))/(2-(1+epsilon2)-
(1+epsilon2)*(1+epsilon2));
  correct_x[0] = 1 - correct_x[1] - (1+epsilon2)*correct_x[2];
  cout<<"correct_x = ";
  for( int i=0 ; i<3 ; i++ ){
    cout<<correct_x[i]<<", ";
  }
  cout<<endl;

  cout<<"error_x = ";
  for( int i=0 ; i<n ; i++ ){
    cout<<fabs(x[i]-correct_x[i])<<", ";
  }
  cout<<endl;

  return 0;
}


///////////////////////////////////////////////
// Implementation of Householder's Algorithm
///////////////////////////////////////////////
void householder( myreal ** m, myreal ** v , int n ){
  /*
  cout<<"before householder's algorithm:"<<endl;
  for( int i = 0 ; i < n ; i++ ){
    for( int j = 0 ; j < n ; j++ ){
      cout<<m[i][j]<<", ";
    }
    cout<<endl;
  }
  */

  // note: m[rows][columns]

  myreal * alpha = new myreal[n],
    * beta = new myreal[n],
    * gamma = new myreal[n];
  myreal sign = 1, sum;

  for( int k=0 ; k < n ; k++ ){ //loop over columns
    sign = ( m[k][k] < 0 ) ? 1 : -1 ;
    sum = 0;
    for( int q=k ; q < n ; q++ ){
      sum += m[q][k]*m[q][k];
    }
    alpha[k] = sign * sqrt( sum );

    // note that v[columns-of-m][rows-of-m]

    for( int i=0 ; i<k ; i++ ){
      v[k][i] = 0;
    }
    for( int i=k ; i < n ; i++ ){
      v[k][i] = m[i][k];
    }
    v[k][k] -= alpha[k];  // ie: v[k] - e[k]*alpha[k]

    beta[k] = 0;
    for( int q=0 ; q < n ; q++ ){
      beta[k] += v[k][q]*v[k][q];
    }

    if( beta[k] == 0 ){ //|| fabs( beta[k] ) < FLT_MIN ){
      continue;
    }

    for( int j=k ; j < n ; j++ ){
      gamma[j] = 0;
      for( int q=0 ; q < n ; q++ ){
      gamma[j] += v[k][q]*m[q][j];
      }
      for( int q=0 ; q < n ; q++ ){
      m[q][j] -= (2*gamma[j]/beta[k])*v[k][q];
      }
    }

    /*
    cout<<endl<<"*******"<<endl<<"iteration(row)
"<<k<<endl<<"********"<<endl;

    cout<<endl<<"v["<<k<<"] = ";
    for( int p=0 ; p<n ; p++ ){
      cout<<v[k][p]<<", ";
    }
    cout<<endl;
    cout<<"alpha = "<<alpha[k]<<endl;

    cout<<endl<<"m:"<<endl;
    for( int p=0 ; p<n ; p++ ){
      for(int q=0 ; q<n ; q++ ){
      cout<<m[p][q]<<", ";
      }
      cout<<endl;
    }
    */

  }

  /*
  cout<<"after householder's algorithm:"<<endl;
  for( int i = 0 ; i < n ; i++ ){
    for( int j = 0 ; j < n ; j++ ){
      cout<<m[i][j]<<", ";
    }
    cout<<endl;
  }
  cout<<"v:"<<endl;
  for( int i = 0 ; i < n ; i++ ){
    for( int j = 0 ; j < n ; j++ ){
      cout<<v[i][j]<<", ";
    }
    cout<<endl;
  }
  cout<<"alpha:"<<endl;
    for( int j = 0 ; j < n ; j++ ){
      cout<<alpha[j]<<", ";
    }
    cout<<endl;
  cout<<"beta:"<<endl;
    for( int j = 0 ; j < n ; j++ ){
      cout<<beta[j]<<", ";
    }
    cout<<endl;
  */
}

////////////////////////////////
// HouseHolder vectors solver (instead of using FSA with a matrix, this
takes the vectors and solves it)
//
//  explanation:
//     after householder(), should have A=QR, where R is the modified-
in-place 'm', and Q is the collection of vectors 'v'
//    to solve the equation Ax=b, now we have QRx=b, which can be
solved:
//        Qy=b, and Rx=y (as with cholesky).
//     but instead of using FSA as in cholesky, we do this: Qy=b, y=
(Q^t)b=H[n-1]*...*H[2]*H[1]*b
//         which can be calculated using the vectors v (which store all
info necessary for reconstructin H[n] and thus Q.
//
//   This function performs the task of solving Qy=b, using the vectors
v.
//
////////////////////////////////
void householder_solver( myreal ** v , myreal * y , myreal * b , int
n ){

  myreal beta, gamma;
  // going to do arithmetic in place, so we will copy b to y, and use y:
  for( int i=0 ; i<n ; i++ ){
    y[i] = b[i];
  }

  //cout<<"householder solver"<<endl;

  // loop over every vector v, create H from v, and multiply H[k]*(H[k-
1]*...*H[1]*b).
  // here H*b = b - (2*(v*b)/(v*v))*v
  for( int k=0 ; k<n ; k++ ){

    /*
    cout<<"y = ";
    for(int i=0; i<n ; i++ ){
      cout<<y[i]<<", ";
    }
    cout<<endl;
    */

    beta = 0; // calculate v*v
    for( int i=0 ; i<n ; i++ ){
      beta += v[k][i]*v[k][i];
    }
    if( beta == 0 ){ continue; }

    gamma = 0;  //calculate v*b
    for( int i=0 ; i<n ; i++ ){
      gamma += v[k][i]*y[i];
    }

    // now calculate b = b - (2*gamma/beta)*v
    for( int i=k ; i<n ; i++ ){
      y[i] -= (2*gamma/beta)*v[k][i];
    }
    /*
    cout<<"y = ";
    for(int p=0 ; p<n ;p++ ){
      cout<<y[p]<<", ";
    }
    cout<<endl;
    */

  }


}
```

```
// forward substitution to solve LOWER triangle systems. (from p. 65 in
Heath)
void FSA( myreal ** m , myreal * x , myreal * b , int n){

  for( int j=0 ; j<n ; j++ ){  // loop over columns
    if( m[j][j] == 0 ){       // stop if matrix is singular
      cerr<<"**********"<<endl<<"FSA: matrix is singular,
stopping"<<endl<<"( element m["<<j<<"]["<<j<<"]=0 )
"<<endl<<"*************"<<endl;
      return;
    }
    x[j] = b[j] / m[j][j];  // compute solution component
    for( int i=j+1 ; i < n ; i++ ){  //update right-hand side.
      b[i] = b[i] - m[i][j]*x[j];
    }
  }

}


// backward substitution to solve UPPER triangle systems (from p.66 in
Heath)
void BSA( myreal ** m , myreal * x , myreal * b , int n){

  for( int j=(n-1) ; j >= 0 ; j-- ){  //loop backwards over columns
    if( m[j][j] == 0 ){  // stop if matrix is singular
      cerr<<"**********"<<endl<<"BSA: matrix is singular,
stopping"<<endl<<"( element m["<<j<<"]["<<j<<"]=0 )
"<<endl<<"*************"<<endl;
      return;
    }
    x[j] = b[j] / m[j][j];  // compute solution component.
    for( int i=0 ; i<= j-1 ; i++ ){  // update right hand side.
      b[i] = b[i] - m[i][j]*x[j];
    }
  }

}

// transposes a matrix in itself:
void transpose( myreal ** m , int n ){

  myreal ** temp = new myreal*[n];
  for( int i=0 ; i<n ; i++ ){
    temp[i] = new myreal[n];
    for( int j=0 ; j<n ; j++ ){
      temp[i][j] = m[i][j];
    }
  }


  /*
  cout<<"after temp matrix copied over:"<<endl;
  for( int i = 0 ; i < n ; i++ ){
    for( int j = 0 ; j < n ; j++ ){
      cout<<temp[i][j]<<", ";
    }
    cout<<endl;
  }
  */

  for( int i=0 ; i<n ; i++ ){
    for( int j=0 ; j<n ; j++ ){
      m[i][j] = temp[j][i];
    }
  }

  /*
  cout<<"after transpose matrix:"<<endl;
  for( int i = 0 ; i < n ; i++ ){
    for( int j = 0 ; j < n ; j++ ){
      cout<<m[i][j]<<", ";
    }
    cout<<endl;
  }
  */

  for( int r=0 ; r < n ; r++ ){
    delete[] temp[r];
  }
  delete[] temp;
}

// matrix * matrix multiplication
void mult( myreal ** m1 , myreal ** m2 , int n ){
  myreal ** temp = new myreal*[n];
  for( int i=0 ; i<n ; i++ ){
    temp[i] = new myreal[n];
    for( int j=0; j<n ; j++ ){
      temp[i][j] = 0;
      for( int k=0; k<n ; k++ ){
      temp[i][j] += m1[i][k]*m2[k][j];
      }
    }
  }

  for( int i=0 ; i<n ; i++ ){
    for( int j=0 ; j<n ; j++ ){
      m2[i][j] = temp[i][j];
    }
  }

  for( int r=0 ; r < n ; r++ ){
    delete[] temp[r];
  }
  delete[] temp;

}



// matrix * vector multiplication
void mult2( myreal ** m , myreal * v , int n ){

  myreal * temp = new myreal[n];

  for( int i=0 ; i<n ; i++ ){
    temp[i] = 0;
    for( int j=0 ; j<n ; j++ ){
      temp[i] += m[i][j]*v[j];
    }
  }

  for( int i=0; i<n ; i++ ){
    v[i] = temp[i];
  }

  delete[] temp;

}
```

Daniel Perry
cs6210 / Fall 2005
Homework 4

* Daniel Perry
```
 * cs6210 - homework 4 problem 2 - implement power method:
 *
 */


typedef float myreal;

#include <iostream>
using namespace std;

#include <math.h>

//matrix functions
void transpose( myreal ** m , int n );
void mult( myreal ** m , myreal ** m2 , int n );
void mult2( myreal ** m , myreal * v , int n );
myreal norm2( myreal * v  , int n );
myreal mult3( myreal * v1 , myreal * v2 , int n );

int main(){

  int n = 3;

  // declaration, allocation:
  myreal ** m = new myreal * [n], // matrix
    lambda0 = 0, // eigenval
    lambda1 = 0, // eigenval
    * y1 = new myreal[n],  // temporary solution vector.
    * x0 = new myreal[n],  // temporary solution vector
    * x1 = new myreal[n];  // temporary solution vector
  for( int i=0 ; i<n ; i++ ){
    m[i] = new myreal[n];
  }

  // initialize the matrix:
  m[0][0] = 2; m[0][1] = 3; m[0][2] = 2;
  m[1][0] = 10; m[1][1] = 3; m[1][2] = 4;
  m[2][0] = 3; m[2][1] = 6; m[2][2] = 1;

  cout<<"used this matrix:"<<endl;
  for( int i = 0 ; i < n ; i++ ){
    for( int j = 0 ; j < n ; j++ ){
      cout<<m[i][j]<<", ";
    }
    cout<<endl;
  }
  cout<<endl;
  //initialze x0:
  //for( int i=0 ; i<n ; i++ ){
  //  x0[i] = 1;
  //}
  x0[0]=0;x0[1]=0;x0[2]=1;

  cout<<"x0=[";
  for( int i=0 ; i<n ; i++ ){
    cout<<x0[i]<<",";
  }
  cout<<"]"<<endl;

  int k;
  for( k=1 ; k<100 ; k++ ){

    // y1 = A*x0
    for( int i=0 ; i<n ; i++ ){
      y1[i] = x0[i];
    }
    mult2( m , y1 , n );

    //x1 = y1 / ||y1||
    myreal norm = norm2( y1 , n );
    for( int i=0 ; i<n ; i++ ){
      x1[i] = y1[i]/norm;
    }

    //lambda1 = x0*y1
    lambda1 = mult3( x0 , y1 , n );

    /*
    cout<<endl<<"lambda1="<<lambda1<<endl;
    cout<<"lambda0="<<lambda0<<endl;
    cout<<"x0=[";
    for( int i=0 ; i<n ; i++ )
      cout<<x0[i]<<",";
    cout<<"]"<<endl;
    cout<<"x1=[";
    for( int i=0 ; i<n ; i++ )
      cout<<x1[i]<<",";
    cout<<"]"<<endl;
    cout<<"y1=[";
    for( int i=0 ; i<n ; i++ )
      cout<<y1[i]<<",";
    cout<<"]"<<endl<<endl;
    */

    if( lambda1 - lambda0 < 10e-8 && k>1 ){
      break;
    }else{
      lambda0 = lambda1;
      for( int i=0 ; i<n ; i++ ){
      x0[i] = x1[i];
      }
    }

  }

  cout<<endl<<"RESULTS:"<<endl;
  cout<<"iters="<<k<<endl;
  cout<<"lambda1="<<lambda1<<endl;
  cout<<"lambda0="<<lambda0<<endl;
  cout<<"x0=[";
  for( int i=0 ; i<n ; i++ )
    cout<<x0[i]<<",";
  cout<<"]"<<endl;
  cout<<"x1=[";
  for( int i=0 ; i<n ; i++ )
    cout<<x1[i]<<",";
  cout<<"]"<<endl;
  cout<<"y1=[";
  for( int i=0 ; i<n ; i++ )
    cout<<y1[i]<<",";
  cout<<"]"<<endl<<endl;

  myreal * checkvec = new myreal[n];
  for( int i=0; i<n ; i++ ){
    checkvec[i] = x1[i];
    x1[i] *= lambda1;
  }

  cout<<endl<<"check:"<<endl;
  mult2( m , checkvec , n );
  cout<<"A*x1=[";
  for( int i=0; i<n ; i++ ){
    cout<<checkvec[i]<<",";
  }
  cout<<"]"<<endl;

  cout<<"lambda1*x1=[";
  for( int i=0 ; i<n ; i++ ){
    cout<<x1[i]<<",";
  }
  cout<<"]"<<endl<<endl;

  return 0;
}
```

```
// transposes a matrix in itself:
void transpose( myreal ** m , int n ){

  myreal ** temp = new myreal*[n];
  for( int i=0 ; i<n ; i++ ){
    temp[i] = new myreal[n];
    for( int j=0 ; j<n ; j++ ){
      temp[i][j] = m[i][j];
    }
  }


  /*
  cout<<"after temp matrix copied over:"<<endl;
  for( int i = 0 ; i < n ; i++ ){
    for( int j = 0 ; j < n ; j++ ){
      cout<<temp[i][j]<<", ";
    }
    cout<<endl;
  }
  */

  for( int i=0 ; i<n ; i++ ){
    for( int j=0 ; j<n ; j++ ){
      m[i][j] = temp[j][i];
    }
  }

  /*
  cout<<"after transpose matrix:"<<endl;
  for( int i = 0 ; i < n ; i++ ){
    for( int j = 0 ; j < n ; j++ ){
      cout<<m[i][j]<<", ";
    }
    cout<<endl;
  }
  */

  for( int r=0 ; r < n ; r++ ){
    delete[] temp[r];
  }
  delete[] temp;
}

// matrix * matrix multiplication
void mult( myreal ** m1 , myreal ** m2 , int n ){
  myreal ** temp = new myreal*[n];
  for( int i=0 ; i<n ; i++ ){
    temp[i] = new myreal[n];
    for( int j=0; j<n ; j++ ){
      temp[i][j] = 0;
      for( int k=0; k<n ; k++ ){
      temp[i][j] += m1[i][k]*m2[k][j];
      }
    }
  }

  for( int i=0 ; i<n ; i++ ){
    for( int j=0 ; j<n ; j++ ){
      m2[i][j] = temp[i][j];
    }
  }

  for( int r=0 ; r < n ; r++ ){
    delete[] temp[r];
  }
  delete[] temp;

}



// matrix * vector multiplication
void mult2( myreal ** m , myreal * v , int n ){

  myreal * temp = new myreal[n];

  for( int i=0 ; i<n ; i++ ){
    temp[i] = 0;
    for( int j=0 ; j<n ; j++ ){
      temp[i] += m[i][j]*v[j];
    }
  }

  for( int i=0; i<n ; i++ ){
    v[i] = temp[i];
  }

  delete[] temp;

}

// vector * vector multiplication
myreal mult3( myreal * v1 , myreal * v2 , int n ){

  myreal temp = 0;
  for( int i=0; i<n ; i++ ){
    temp += v1[i]*v2[i];
  }

  return temp;

}

myreal norm2( myreal * v , int n ){

  myreal result = 0;

  for( int i=0 ; i<n ; i++ ){
    result += (v[i]*v[i]);
  }

  return sqrt( result );
}
```

Daniel Perry
cs6210 / Fall 2005
Homework 4

```
/* Daniel Perry
 * cs6210 - homework 4 problem 2(ii) - implement householder deflation:
 *
 */


typedef double myreal;

#include <iostream>
using namespace std;

#include <math.h>

void householder( myreal ** m, myreal ** v , int n );

//matrix functions
void transpose( myreal ** m , int n );
void mult( myreal ** m , myreal ** m2 , int n );
void mult2( myreal ** m , myreal * v , int n );
myreal norm2( myreal * v  , int n );
myreal mult3( myreal * v1 , myreal * v2 , int n );
void dyadic( myreal * v1 , myreal * v2 , myreal ** m , int n );

int main(){
//....
//  HERE PUT ALL OF CODE IN main() from power code...
//....

  ///////////////////////////
  // NOW DEFLATE A, using one iteration of Householder algorithm, twice
  ///////////////////////////

  myreal ** v = new myreal*[n];
  myreal ** m_bak = new myreal*[n];
  myreal ** H = new myreal*[n];
  for(int i=0; i<n ; i++ ){
    v[i] = new myreal[n];
    m_bak[i] = new myreal[n];
    H[i] = new myreal[n];
    for( int j=0 ; j<n ; j++ ){
      m_bak[i][j] = m[i][j];
    }
  }

  cout<<"A matrix:"<<endl;
  for( int i = 0 ; i < n ; i++ ){
    for( int j = 0 ; j < n ; j++ ){
      cout<<m[i][j]<<", ";
    }
    cout<<endl;
  }
  cout<<endl;


  myreal * w = new myreal[n]; // to store w from householder...

  householder( m , v , n ); // results in m = H1*A
  for( int i=0 ; i<n ; i++ ){
    w[i] = v[0][i];
  }
  householder( m , v , n ); // results in m = H1*A*H1

  cout<<"H1*A*H1 matrix(deflated):"<<endl;
  for( int i = 0 ; i < n ; i++ ){
    for( int j = 0 ; j < n ; j++ ){
      cout<<m[i][j]<<", ";
    }
    cout<<endl;
  }
  cout<<endl;


  myreal ** m_sub = new myreal*[n-1];
  for( int i=0 ; i<(n-1) ; i++ ){
    m_sub[i] = new myreal[n-1];
    for( int j=0 ; j<(n-1) ; j++ ){
      m_sub[i][j] = m[i+1][j+1];
    }
  }

  //////////////////////////////////
  // NOW user power method on m_sub:
  //////////////////////////////////

  // reset vectors (only going to be using *[0] and *[1].
  x0[0]=1;x0[1]=1;x0[2]=0;
  x1[0]=0;x1[1]=0;x1[2]=0;
  y1[0]=0;y1[1]=0;y1[2]=0;

  cout<<"B matrix:"<<endl;
  for( int i = 0 ; i < (n-1) ; i++ ){
    for( int j = 0 ; j < (n-1) ; j++ ){
      cout<<m_sub[i][j]<<",";
    }
    cout<<endl;
  }
  cout<<endl;
  for( k=1 ; k<1000000 ; k++ ){

    // y1 = A*x0
    for( int i=0 ; i<(n-1) ; i++ ){
      y1[i] = x0[i];
    }
    mult2( m_sub , y1 , (n-1) );

    //x1 = y1 / ||y1||
    myreal norm = norm2( y1 , (n-1) );
    for( int i=0 ; i<(n-1) ; i++ ){
      x1[i] = y1[i]/norm;
    }

    //lambda1 = x0*y1
    lambda1 = mult3( x0 , y1 , (n-1) );

    /*
    cout<<endl<<"lambda1="<<lambda1<<endl;
    cout<<"lambda0="<<lambda0<<endl;
    cout<<"x0=[";
    for( int i=0 ; i<(n-1) ; i++ )
      cout<<x0[i]<<",";
    cout<<"]"<<endl;
    cout<<"x1=[";
    for( int i=0 ; i<(n-1) ; i++ )
      cout<<x1[i]<<",";
    cout<<"]"<<endl;
    cout<<"y1=[";
    for( int i=0 ; i<(n-1) ; i++ )
      cout<<y1[i]<<",";
    cout<<"]"<<endl<<endl;
    */

    if( fabs(lambda1 - lambda0) < 10e-7 && k>1 ){
      break;
    }else{
      lambda0 = lambda1;
      for( int i=0 ; i<(n-1) ; i++ ){
      x0[i] = x1[i];
      }
    }

  }

  cout<<endl<<"RESULTS:"<<endl;
  cout<<"iters="<<k<<endl;
  cout<<"lambda1="<<lambda1<<endl;
  cout<<"lambda0="<<lambda0<<endl;
  cout<<"x0=[";
  for( int i=0 ; i<(n-1) ; i++ )
    cout<<x0[i]<<",";
  cout<<"]"<<endl;
  cout<<"x1=[";
  for( int i=0 ; i<(n-1) ; i++ )
    cout<<x1[i]<<",";
  cout<<"]"<<endl;
  cout<<"y1=[";
  for( int i=0 ; i<(n-1) ; i++ )
    cout<<y1[i]<<",";
  cout<<"]"<<endl<<endl;

  //////////////////////
  // construct eigenvector of A from eigenvector of B:

  myreal * bT = new myreal[n-1], gamma;

  cout<<"bT=[";
  for( int i=0 ; i<(n-1) ; i++ ){
    bT[i] = m[0][i+1];
    cout<<bT[i]<<",";
  }
  cout<<"]"<<endl;

  gamma = mult3( bT , x1 , n-1 ); // bT*eigenvec(B)
  gamma /= lambda1-lambda1_bak;

  x1[2]=x1[1]; x1[1]=x1[0]; x1[0]= gamma;

  //construct H from w (saved from above)... H=(I-2*w*w^T) = I-2*dyadic
(w,w^T)
  dyadic( x1 , x1 , m , n );
  for( int i=0 ; i<n ; i++ ){
    for( int j=0 ; j<n ; j++ ){
      if( i==j ){
      m[i][j] = 1-2*m[i][j];
      }else{
      m[i][j] = -2*m[i][j];
      }
    }
  }
  // now m=H

  mult2( m , x1 , n ); // H*x1

  cout<<"second eigenvec:"<<endl;
  cout<<"x1=";
  for(int i=0 ; i<n ; i++ ){
    cout<<x1[i]<<",";
  }
  cout<<endl<<endl;

  /// Check it

  for( int i=0; i<n ; i++ ){
    checkvec[i] = x1[i];
    x1[i] *= lambda1;
  }

  cout<<endl<<"check:"<<endl;
  mult2( m_bak , checkvec , n );
  cout<<"A*x1=[";
  for( int i=0; i<n ; i++ ){
    cout<<checkvec[i]<<",";
  }
  cout<<"]"<<endl;

  cout<<"lambda1*x1=[";
  for( int i=0 ; i<n ; i++ ){
    cout<<x1[i]<<",";
  }
  cout<<"]"<<endl<<endl;

  return 0;
}

// most of the same helper functions as power method, only kept those
that are different:

/////////////////////////////////////////////////
// Implementation of Householder's Algorithm
// ----> modified, so that it only performs one iteration (not the
complete decomposition).
/////////////////////////////////////////////////
void householder( myreal ** m, myreal ** v , int n ){
  /*
  cout<<"before householder's algorithm:"<<endl;
  for( int i = 0 ; i < n ; i++ ){
    for( int j = 0 ; j < n ; j++ ){
      cout<<m[i][j]<<", ";
    }
    cout<<endl;
  }
  */

  // note: m[rows][columns]
  myreal * alpha = new myreal[n],
    * beta = new myreal[n],
    * gamma = new myreal[n];
  myreal sign = 1, sum;

  for( int k=0 ; k < 1 ; k++ ){ //loop over columns
    sign = ( m[k][k] < 0 ) ? 1 : -1 ;
    sum = 0;
    for( int q=k ; q < n ; q++ ){
      sum += m[q][k]*m[q][k];
    }
    alpha[k] = sign * sqrt( sum );

    // note that v[columns-of-m][rows-of-m]

    for( int i=0 ; i<k ; i++ ){
      v[k][i] = 0;
    }
    for( int i=k ; i < n ; i++ ){
      v[k][i] = m[i][k];
    }
    v[k][k] -= alpha[k];  // ie: v[k] - e[k]*alpha[k]

    beta[k] = 0;
    for( int q=0 ; q < n ; q++ ){
      beta[k] += v[k][q]*v[k][q];
    }

    if( beta[k] == 0 ){ //|| fabs( beta[k] ) < FLT_MIN ){
      continue;
    }

    for( int j=k ; j < n ; j++ ){
      gamma[j] = 0;
      for( int q=0 ; q < n ; q++ ){
      gamma[j] += v[k][q]*m[q][j];
      }
      for( int q=0 ; q < n ; q++ ){
      m[q][j] -= (2*gamma[j]/beta[k])*v[k][q];
      }
    }

    /*
    cout<<endl<<"*******"<<endl<<"iteration(row)
"<<k<<endl<<"*******"<<endl;

    cout<<endl<<"v["<<k<<"] = ";
    for( int p=0 ; p<n ; p++ ){
      cout<<v[k][p]<<", ";
    }
    cout<<endl;
    cout<<"alpha = "<<alpha[k]<<endl;

    cout<<endl<<"m:"<<endl;
    for( int p=0 ; p<n ; p++ ){
      for(int q=0 ; q<n ; q++ ){
      cout<<m[p][q]<<", ";
      }
      cout<<endl;
    }
    */

  }

  /*
  cout<<"after householder's algorithm:"<<endl;
  for( int i = 0 ; i < n ; i++ ){
    for( int j = 0 ; j < n ; j++ ){
      cout<<m[i][j]<<", ";
    }
    cout<<endl;
  }
  cout<<"v:"<<endl;
  for( int i = 0 ; i < n ; i++ ){
    for( int j = 0 ; j < n ; j++ ){
      cout<<v[i][j]<<", ";
    }
    cout<<endl;
  }
  cout<<"alpha:"<<endl;
    for( int j = 0 ; j < n ; j++ ){
      cout<<alpha[j]<<", ";
    }
    cout<<endl;
  cout<<"beta:"<<endl;
    for( int j = 0 ; j < n ; j++ ){
      cout<<beta[j]<<", ";
    }
    cout<<endl;
  */
}

// vector * vector^(T) = matrix
void dyadic( myreal * v1 , myreal * v2 , myreal ** m , int n ){

  myreal temp = 0;

  for( int i=0; i<n ; i++ ){
    for( int j=0 ; j<n ; j++ ){
      m[i][j] = v1[i]*v2[j];
    }
  }
}

// second norm of a vector.. (length)
myreal norm2( myreal * v , int n ){

  myreal result = 0;

  for( int i=0 ; i<n ; i++ ){
    result += (v[i]*v[i]);
  }

  return sqrt( result );
}
```