Due Friday, February 10 in class

0. Look over Sections 1.5–2.5 of the class notes and Section 1.11 of the textbook.

1. (15 points) Write a recursive program in Matlab, Python or C that inverts a nonsingular lower triangular matrix. The inverse of an \( n \times n \) matrix should be computed in terms of the inverse of a \( n-1 \) by \( n-1 \) matrix unless \( n = 1 \), in which case it should be computed directly. The program should accept a two-dimensional array and return the inverse. You may assume valid input. Try your program on the 8 by 8 matrix

\[
\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
h^2 -1 & h^2 -2 & 1 & & & & & \\
1 & h^2 -2 & 1 & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
\end{bmatrix}
\]

First try it with \( h = \frac{1}{2} \). Then try it on the inverse that you just computed to see if you get the original matrix back again. Finally try it with \( h = 0 \). (The above matrix arises in the discretization of the dynamics of a spring by an eighteenth century integrator known variously as the Störmer method, the leapfrog method, and the Verlet method.) Hand in a printed copy of your program and its output.

\[
\begin{bmatrix}
L & 0 \\
\ell^T & \lambda
\end{bmatrix}^{-1} =
\begin{bmatrix}
L^{-1} & 0 \\
-\lambda^{-1}\ell^T L^{-1} & \lambda^{-1}
\end{bmatrix}
\]

Here is the matlab code:

matrix.m:
\[
\text{function res = matrix(h, n)}
\text{result = zeros(n, n);}
\text{for i=1:n}
  \text{if (i == 1)}
    \text{result(1, 1) = 1;}
  \text{else if (i == 2)}
    \text{result (2, 1) = h*h/2 -1 ;}
    \text{result(2, 2) = 1;}
  \text{else}
    \text{result(i, i-2) = 1;}
    \text{result(i, i-1) = h*h-2;}
    \text{result(i, i) = 1;}
  \text{end}
\text{end}
\text{res = result;}
\text{return;}
\]
inverse.m:

function res = inverse(A)
[m,n]=size(A);
if (m ~= n) return;
else if (m == 1 & n == 1)
  B = 1/A(1, 1);
else
  C = A(1:m-1, 1:n-1);
  D = A(m, 1:n-1);
  E = A(m, n);
  F = inverse(C);
  B = [F, zeros(m-1, 1); -(1/E)*D*F, 1/E];
end
end
res = B;
return;
end

main.m:

function main()
n = 8;
h = [0 0.5];

for i=1:2
  fprintf(1, 'h = %f
', h(i));
  A = matrix(h(i), 8);
  fprintf(1, 'The matrix A is:
');
  A
  B = inverse(A);
  fprintf(1, 'The inverse matrix of A:
');
  B
  C = A * B;
  fprintf(1, 'The product of A and its inverse matrix is:
');
  C
end
end

Here is the C code:

#include <stdio.h>

const int SIZE = 8;
void print(double matr[SIZE][SIZE])
{
    int i, j;
    for (i = 0; i < SIZE; i++ ) {
        for (j = 0; j < SIZE; j++ )
            printf("%f", matr[i][j]);
        printf("\n");
    }
    printf("\n\n");
}

void zero(double matr[SIZE][SIZE])
{
    int i, j;
    for (i = 0; i < SIZE; i++)
        for (j = 0; j < SIZE; j++)
            matr[i][j] = 0;
}

void init(double matr[SIZE][SIZE], double h)
{
    int i;
    matr[0][0] = 1.0;
    matr[1][0] = h*h/2.0 - 1.0;
    matr[1][1] = 1.0;
    for (i = 2; i < SIZE; i++) {
        matr[i][i-2] = 1.0;
        matr[i][i-1] = h*h - 2.0;
        matr[i][i] = 1.0;
    }
}

void inverse(double a[SIZE][SIZE], double b[SIZE][SIZE], int size)
{
    int i, j;
    if (size == 1) {
        b[0][0] = 1.0/a[0][0];
        return;
    }
    inverse(a, b, size-1);
    b[size-1][size-1] = 1/a[size-1][size-1];
    for (i = 0; i < size-1; i++) {
        b[size-1][i] = 0;
        for (j = 0; j < size-1; j++)
            b[size-1][i] += a[size-1][j]*b[j][i];
        b[size-1][i] = -b[size-1][i]/a[size-1][size-1];
    }
    for (i = 0; i < size-1; i++)
        b[i][size-1] = 0;
}
void comp(double a[SIZE][SIZE], double b[SIZE][SIZE], double c[SIZE][SIZE])
{
    int i, j, k;
    for (i = 0; i < SIZE; i++)
        for (j = 0; j < SIZE; j++)
            for (k = 0; k < SIZE; k++)
                c[i][j] += a[i][k]*b[k][j];
}

int main () {
    double a[SIZE][SIZE], b[SIZE][SIZE], c[SIZE][SIZE];
    int i;
    double h[2];
    zero(a);
    zero(b);
    h[0] = 0;
    h[1] = 1.0/2.0;
    for (i = 0; i < 2; i++) {
        printf("h = %e\n", h[i]);
        init(a, h[i]);
        printf("The Matrix A is \n");
        print(a);
        inverse(a,b, 8);
        printf("Its inverse Matrix is \n");
        print(b);
        comp(a, b, c);
        printf("Their product is \n");
        print(c);
    }
    return 1;
}

Here is the Python code:

import numpy as Numeric
from numpy import *

def inv_matrix(matrix_in):
    #global result
    dim=matrix_in.shape
    if dim[0]!=dim[1]:  #the function must operate on square matrix"
        return "Error! Must input a square matrix"
    if dim[0]>1:
tmp = matrix_in[0:dim[0]-1, 0:dim[1]-1]  
result = inv_matrix(tmp)  
result = concatenate((result, (-matrix_in[-1, 0:-1]*result)/(matrix_in[-1, -1])), 0)  
k = concatenate((zeros(((dim[0]-1), 1)), [[1/matrix_in[-1, -1]]]), 0)  
result = concatenate((result, k), 1)  
return result  
elif dim[0]==1:  
result = 1/matrix_in  
return result  
else:  
return "unknown error"

def homework_test():  
a = identity(8)  
a = matrix(a, Float)  
i = 0  
h = 0.5  
while i<6:  
a[i+2, i] = 1  
a[i+2, i+1] = h**2 - 2  
i = i+1  
a[1, 0] = h**2/2 - 1  
print "the original matrix:\n", a  
b = inv_matrix(a)  
print "the reverse matrix:\n", b  
c = inv_matrix(b)  
print "the reverse back:\n", c  

h = 0  
i = 0  
while i<6:  
a[i+2, i+1] = h**2 - 2  
i = i+1  
a[1, 0] = h**2/2 - 1  
print "the matrix of h=0:\n", a  
print "the reverse matrix of the one above:\n", inv_matrix(a)
4. (6 points) Suppose that a matrix $A$ has a factorization $A = M_1^{-1}P_2M_2^{-1}U$ where

$$
\begin{bmatrix}
A & a \\
a^T & \alpha
\end{bmatrix} = 
\begin{bmatrix}
\hat{G} & 0 \\
g^T & \gamma
\end{bmatrix}^T 
\begin{bmatrix}
\hat{G} & 0 \\
g^T & \gamma
\end{bmatrix} = 
\begin{bmatrix}
\hat{G}^T & g \\
g^T & \gamma
\end{bmatrix} = 
\begin{bmatrix}
\hat{G}\hat{G}^T & \hat{G}g \\
g^T\hat{G}^T & g^Tg + \gamma^2
\end{bmatrix}
$$

1. $\hat{A} \rightarrow \hat{G}\hat{G}^T$ (recursive call)
2. $g = \hat{G}^{-1}a$ (forward substitution)
3. $\gamma = \sqrt{\alpha - g^Tg}$

3. (5 points) Show in the case of base ten floating-point arithmetic that if $\frac{1}{2} \leq x/y \leq 2$, the difference $x - y$ is computed exactly. (Your argument should be convincing to a person, but not a mechanical proof checker.) Show that this is untrue if (say) $0.4 \leq x/y \leq 2.5$.

For the first part there are two cases. In one case $x$ and $y$ both have the same exponent. If they have $t$-digit mantissas, then clearly difference is representable with $t$ digits. In the other case $x$ and $y$ have exponents that differ by one in which case $x - y$ has the form $\pm d_0d_1d_2...d_t \times 10^e$ where $e$ is the exponent of the smaller of $x$ and $y$. However, because $x - y$ is no larger than the smaller of $x$ and $y$, the leading digit $d_0$ must be zero.

For the second part choose precision $t = 3$, $x = 2.00$ and $y = 0.999$. The difference $x - y = 1.001$ requires 4 digits to represent it.

4. (6 points) Suppose that a matrix $A$ has a factorization $A = M_1^{-1}P_2M_2^{-1}U$ where

$$
M_1 = \begin{bmatrix}
1 & 0 & 0 \\
-1/2 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix},
P_2 = \begin{bmatrix}
0 & 1 & 0 \\
1 & 0 & 0 \\
0 & 0 & 1
\end{bmatrix},
M_2 = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 1/2 & 1
\end{bmatrix},
U = \begin{bmatrix}
2 & -2 & 0 \\
0 & 4 & -2 \\
0 & 0 & 1
\end{bmatrix}.
$$

Without re-forming the original matrix $A$, use this information to solve $A^Tx = b$ efficiently where $b = [5, 1, -4]$. $x = A^{-T}b = (M_1^{-1}P_2M_2^{-1}U)^{-T}b = M_1^TP_2M_2^TU^{-T}b$.

Let $y = U^{-T}b$, then $\begin{bmatrix}
2 & 0 & 0 \\
-2 & 4 & 0 \\
0 & -2 & 1
\end{bmatrix} y = \begin{bmatrix}
5 \\
1 \\
-4
\end{bmatrix} \Rightarrow y = \begin{bmatrix}
5/2 \\
3/2 \\
-1
\end{bmatrix}$

$M_2^Ty = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 1/2 \\
0 & 0 & 1
\end{bmatrix} y = \begin{bmatrix}
5/2 \\
1 \\
-1
\end{bmatrix}$

$P_2M_2^Ty = \begin{bmatrix}
1 \\
5/2 \\
-1
\end{bmatrix}$

$x = \begin{bmatrix}
1 & -1/2 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix} = \begin{bmatrix}
-1/4 \\
5/2 \\
-1
\end{bmatrix}$