MATH 225 Linear Algebra and Differential Equations

Fall 2007 MATLAB Homework 2 Solutions

1. (a) Since the Hilbert matrices are nonsingular, it should reduce to the identity matrix.

```
>> H8=hilb(8);
>> R8=rref(H8);
>> R8
R8 =
     1     0     0     0     0     0     0     0
     0     1     0     0     0     0     0     0
     0     0     1     0     0     0     0     0
     0     0     0     1     0     0     0     0
     0     0     0     0     1     0     0     0
     0     0     0     0     0     1     0     0
     0     0     0     0     0     0     1     0
     0     0     0     0     0     0     0     1
```

H8 is OK!

```
>> H12=hilb(12);
>> R12=rref(H12);
```

We see that last $\boldsymbol{R12} \neq \boldsymbol{I}$; its last column is not equal to $\boldsymbol{e}_{12}$ and its last row is zero.
For n=15, the 13th and 15th columns are not equal to $\boldsymbol{e}_{12}$ and $\boldsymbol{e}_{12}$, and the last two rows are zero! Hence the situation is even worse!
We were expecting identity matrices for all instances but we did not obtain the identity matrices for n=12 and n=15. In fact, Matlab says that H12 and H15 is singular! This is because of the finite precision of the computers; for n=12 and n=15 the entries of Hilbert matrices becomes very small and they are not represented correctly on the computer.

(b)

***n*=8:**

```
>> x=ones(8,1);
>> b=H8*x;
>> xComputed=H8\b

xComputed =

    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
```

xComputed is OK for n=8!

***n*=12:**

```
>> xComputed=H12\b
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 2.409320e-017.

xComputed =

    1.0000
    1.0000
    0.9999
```

```
    1.0013
    0.9907
    1.0408
    0.8867
    1.2046
    0.7605
    1.1752
    0.9272
    1.0131
```

***n*=15:**

```
>> x=ones(15,1);
>> b=H15*x;
>> xComputed=H15\b
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 1.543404e-018.

xComputed =

    1.0000
    1.0000
    1.0002
    0.9962
    1.0453
    0.6907
    2.3152
   -2.6164
    7.4503
   -6.1175
    4.9552
    1.4203
   -1.1554
    2.2795
    0.7366
```

We see that as *n* gets larger, the solution gets worse. For the last two instances, the reduced row echelon forms are not correct, hence we get incorrect results when we solve the system.

2. `invByCol` function:

```
function Ainv=invByCol(A)
n=length(A);
I=eye(n);
Ainv=[];
for j=1:n
    x=A\I(:,j);
    Ainv=[Ainv x];
end
```

The inverse of the test matrix is $\boldsymbol{A}^{-1}=\begin{bmatrix} 3 & -4 & 3 \\ 1 & -2 & 2 \\ -7 & 11 & -9 \end{bmatrix}$. You can find this by hand or by `inv` routine of Matlab. The `invByCol(A)` also returns this matrix.

The code to compare `inv` and `invByCol` is

```
function compareInvMethods(n)
A=rand(n);
tic;
inv(A);
t=toc;
disp('Built-in inv');
t
tic;
invByCol(A);
t=toc;
disp('invByCol');
t
```

When we run this code with *n=5,10,100*, we get

```
>> compareInvMethods(5)
Built-in inv
t =
  1.1845e-004
invByCol
t =
    0.0309

>> compareInvMethods(10)
Built-in inv
t =
  1.4303e-004
invByCol
t =
  5.8192e-004

>> compareInvMethods(100)
Built-in inv
t =
    0.0609
invByCol
t =
    0.2053
```

We see that inv routine is much faster. This is because Matlab uses more efficient LAPACK routines for this calculation.

3. The parabola function:

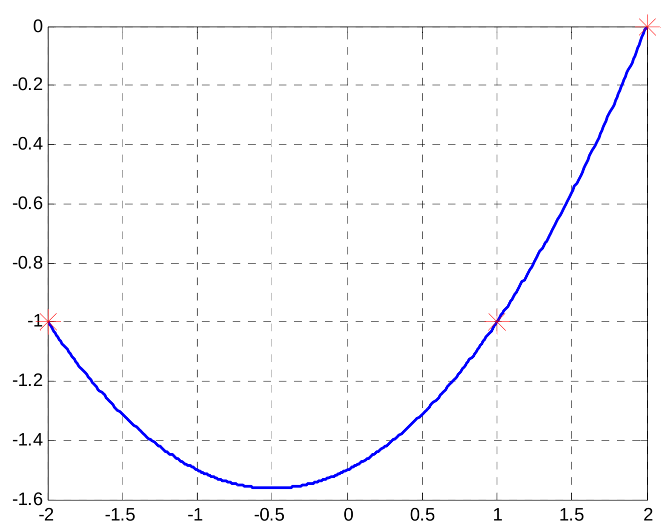
```
function c=parabola(x,y)
rhs=y;
A(:,1)=x.^2;
A(:,2)=x;
A(:,3)=1;
c=A\rhs;
```

You can insert the following lines to this function for checking:

```
xPoints=min(x):0.01:max(x);
yComputed=c(1)*xPoints.^2+c(2)*xPoints+c(3);
plot(xPoints,yComputed,'LineWidth',2);
hold on;
for i=1:3
    plot(x(i),y(i),'r*','MarkerSize',14);
end
grid;
hold off;
```

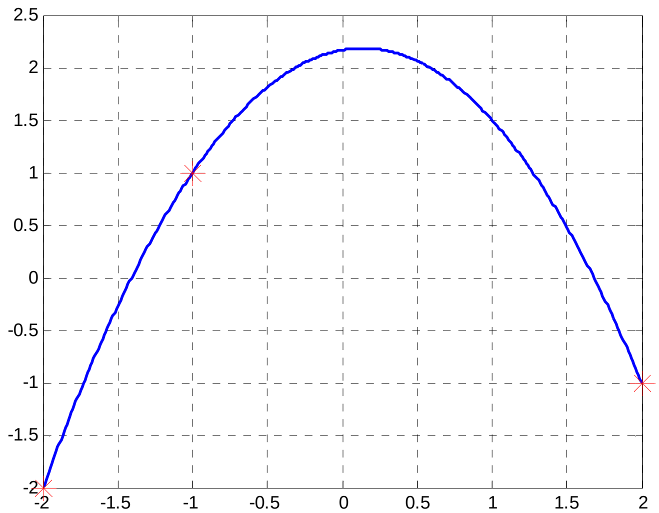
(a)

```
>> x=[-2 1 2]';
>> y=[-1 -1 0]';
>> c=parabola(x,y)

c =
    0.2500
    0.2500
   -1.5000
```



(b)

```
>> x=[-2 -1 2]';
>> y=[-2 1 -1]';
>> c=parabola(x,y)

c =
   -0.9167
    0.2500
    2.1667
```



2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2