CMPS251 - Assignment 3

Kinan Dak Al Bab, 201205052

September 26, 2014

Note: MATLAB R2010b (the version in the labs) was used to solve this assignment.

1 Inverse matrix

1.1 Question

Following the procedure outlined in Example 5.5 (pages 103-104), write a matlab function to compute the inverse of a square matrix. (you may use $lu()$ from matlab or your own $mylu()$ from the last assignment). The function will have the following header:

$function\ B = myinverse(A)$

Test your solution by:

- comparing with the built-in function $inv()$; and
- multiplying a matrix by its inverse that you compute.

1.2 Code

```
function B = myinverse(A)
    E = eye(size(A));
    [L U] = lu(A); % lu decomposition

    Y = L \ E; %Forward pass
    B = U \ Y; %Backwards pass
end



%%% Code for testing myinverse %%%
A = rand(50, 50); % random 50x50 matrix
A1 = myinverse(A);
A2 = inv(A);
norm(A1 - A2); % compare myinverse to built-in inv

e = A * A1;
norm(e - eye(50)) % multiply A by its inverse, result should be ...
    the identity vector
```

1.3 Output

```
ans =

  1.7043e-014


ans =

  1.3354e-013
```

1.4 Comment

Both norm are close to machine precision, which tells us that myinverse and inv yield nearly the same result, and that A * A1 is almost the identity matrix. with some precision errors.

2 Pentadiagonal solver

2.1 Question

Write a function to solve $Ax = b$ where A is pentadiagonal. Do not perform pivoting, as it destroys the structure of the matrix. The header of the function should be:

$function\ x = pentasolve(A, b)$

- Test your solution by generating a pentadiagonal matrix with random entries, and a right hand side vector. You may use the following script:

```
n = 10;
d = rand(n, 1); % entries on  the diagonal
sup1 = rand(n-1, 1); sup2 = rand(n-2, 1); % entries in ...
    the two superdiagonals
sub1 = rand(n-1, 1); sub2 = rand(n-2, 1) % entries in ...
    the two subdiagonals

% Build the matrix A %
A = diag(d) + diag(sup1, 1) + diag(sup2, 2) + ...
    diag(sub1, -1) + diag(sub2, -2);
b = rand(n, 1); % generate a rhs with random ...
    coefficients %
x = pentasolve(A, b); % solve %
norm(A * x - b) % check that solution is correct
```

- What is the cost of solving a pentadiagonal system?

2.2 Code

```
function x = pentasolve(A, b)
[n ¬] = size(A); % get size of matrix
L = eye(n);
for i = 1:n-1 % from 1 to n-1
    for j = i+1:min([n i+2]) % from i + 1 to min(i + 2, n)

        L(j, i)=A(j, i)/A(i, i);

        for k = i:min([n i+2]) % from i + 1 to min(i + 2, n)
            A(j, k)=A(j, k)-L(j, i)*A(i, k);
        end
    end
end

% forward pass %
y = zeros(n, 1); % create a vector for unknowns %
for i = 1:n
    sum = 0;

    for j = max([1 i-2]):i-1
        sum = sum + L(i, j) * y(j);
    end

    y(i) = (b(i) - sum) / L(i, i);
end

% backwards pass %
x = zeros(n, 1); % create a vector for unknowns %
for i = n:-1:1
    sum = 0;

    for j = min([n i+1]):n
        sum = sum + A(i, j) * x(j);
    end

    x(i) = (y(i) - sum) / A(i, i);
end
end
```

2.3 Output

```
ans =

  7.9140e-016
```

2.4 Comment

norm is nearly zero (machine precision), the solution is correct.

2.5 Cost of Solving a Pentadiagonal System

- Forward and Backwards pass: They have the same cost:
  The outter loop has n iterations and 2 flops in each of them, the inner most loop has at max 2 iterations, and it has 2 flops in each iteration. total $2 \times n + 2 \times 2 \times n = 6 \times n$ flops for each pass. $O(n)$.

- LU decomposition:
  The outer most loop is from 1 to n-1.
  The loop after it can iterate two times, it contains one flop inside it, so there are $2 \times (n-1)$ flops.
  The inner most loop iterates three times, and contains two flops (multiplication and addition). so there are $2 \times (n-1) \times 2 \times 3 = 12 \times (n-1)$.
  So for the lu decomposition we would have in total $14 \times (n-1)$ flops. so in order of $O(n)$.

- So the total cost of solving the system falls in $O(n)$.

3 Effect of poor conditioning of A

3.1 Question

- Use the hilb() function to generate a 20×20 Hilbert matrix
- Use *cond*() to compute the condition number of A.
- Generate a right-hand side vector b as $b = A * ones(20, 1)$.
- Use backslash to "solve" $Ax = b$. Is the result you get expected? Comment.

3.2 Code

```
1 A = hilb(20); % generate hilbert matrix
2 cond(A) % print condition of A
3
4 b = A * ones(20, 1);
5 x = A \ b; % solve A x = b
6
7 norm(A*x - b) % print the norm
8 norm(x - ones(20, 1)) % norm of the solution to the vector of ones
```

3.3 Output

```
ans =

  1.6441e+018

Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 1.155429e-019.
```

```
> In conditionin at 5

ans =

  4.0747e-015

ans =

  201.1761
```

3.4 Comment

We know that the solution of this System is a vector of 20 ones (by construction). notice that the condition of A is fairly High (whether the one returned by the cond function, or the one in matlab's warning).

After solving the system, we get some solution x. this solution is a valid solution of the system (the norm is close to 0, machine precision). however the norm of its difference with the ones vector is quite high indicating that it is not a vector of ones.

So we have "two solutions" for the system in some sense, that is due to the fact that A is highly singular. which is shown by the high condition of A.

4 Bisection

4.1 Question

Write a script that uses the bisection method to solve a scalar nonlinear equation $f(x) = 0$

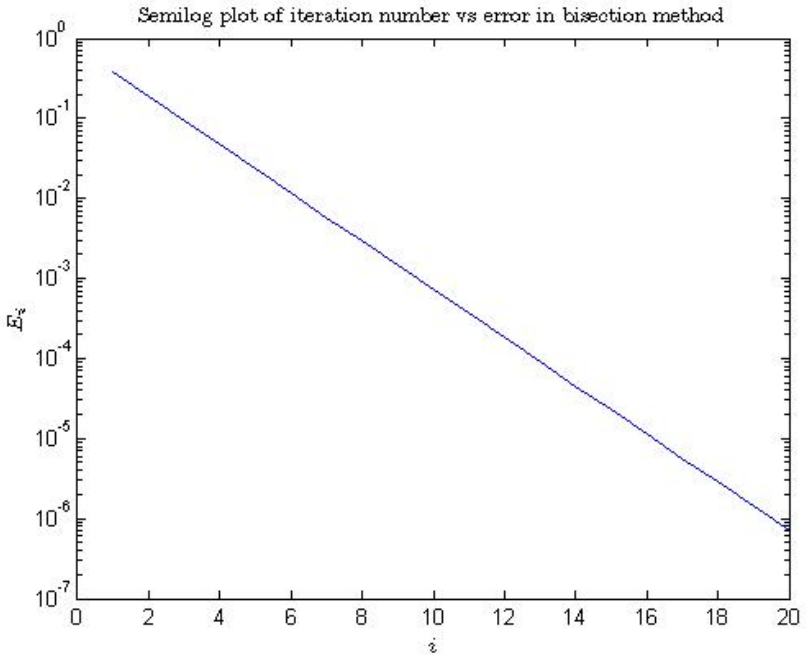
- Print each iterate and its corresponding function value. (use format long)
- Plot (on a semilog graph) the error as a function of iteration number. Comment.

4.2 Code

```
1  format long
2
3  a = 0.5;
4  b = 2;
5
6  f_a = a^3 - sin(a);
7  f_b = b^3 - sin(b);
8
9  atol = 10^-6;
10
11 x = (a + b) / 2;
12 f_x = x^3 - sin(x);
```

```
13 tmp = b;
14 err = [];
15 while abs(f_x) > atol
16     if f_a * f_x < 0 % solution in left segment
17         b = x;
18         f_b = f_x;
19     else % solution in right segment
20         a = x;
21         f_a = f_x;
22     end
23
24     err = [err,(b - a) / 2];
25
26     tmp = x;
27     x = (a + b) / 2
28     f_x = x^3 - sin(x)
29     i = i + 1;
30 end
31
32 i = 1:length(err);
33 semilogy(i, err);
34
35 xlabel('$i$', 'Interpreter', 'latex');
36 ylabel('$E_i$', 'Interpreter', 'latex');
37 title('Semilog plot of iteration number vs error in bisection ...
       method', 'Interpreter', 'latex');
```

4.3 Plot



4.4 Output

```
x =

   0.875000000000000


f_x =

  -0.097621627236027

.
.
.
.
.

x =

   0.928626298904419


f_x =

   -1.953774331209246e-008
```

4.5 Comment

The algorithm converges slowely, by a factor of two, because the error is decreasing by a factor of 2 each time, in a sense the algorithm is close to binary search. the error appears as a straight line in semilog scale.

5 Newton's Method

5.1 Question

Write Matlab code that implements Newton's method to nd the root of the equation $x^3 - 2x - 5 = 0$ in the interval $2 \leq x \leq 3$ to an accuracy of 10^{-6}.

- Print each iterate and its corresponding function value. (use format long)
- Use the function values at the last three iterates to compute the observed convergence rate.

5.2 Code

```
format long

x = 2.5;
f_x = (x^3 - 2*x - 5);
while abs(f_x) > 10^-6
    x = x - (f_x / (3*x^2 - 2));
    f_x = (x^3 - 2*x - 5)
end
```

5.3 Output

```
f_x =

   0.807945126228955


f_x =

   0.028881721218617


f_x =

    4.186494613200864e-005


f_x =

    8.840039811275346e-011
```

5.4 Comment

The algorithm converges very quickly. taking only 4 iterations. The last iteration had an error in the order of 10^{-11}, The previous iteration had an error in the order of 10^{-5}, The last iteration had an error in the order of 10^{-2}. the decrease in error have quadratic behavior, which is what the theory tells us.

6 Cube root by Newton's method

6.1 Question

Solve Exercise 5 on page 59 of your text.

6.2 Code

```
format long
aa = [0 2 10];
```

```
3
4  for i = 1:3
5      a = aa(i)
6
7      x = a/3;
8      f_x = (x^3 - a)
9      while abs(f_x(1)) > 10^-6
10         x = x - (f_x / (3*x^2));
11         f_x = (x^3 - a)
12     end
13
14     x
15 end
```

6.3 Output

```
a =

     0


f_x =

     0


x =

     0


a =

     2


f_x =

  -1.703703703703704


f_x =

   5.351680384087794


f_x =

   1.193556355407582
```

```
f_x =

   0.142518127663455

f_x =

   0.003136697703048

f_x =

    1.636959448880759e-006

f_x =

    4.463096558993129e-013

x =

   1.259921049894967

a =

    10

f_x =

  27.037037037037038

f_x =

   6.045381344307270

f_x =

   0.727450893157128

f_x =

   0.016319427316404
```

```
f_x =

    8.861388550940319e-006


f_x =

    2.616573624436569e-012


x =

   2.154434690032072
```

6.4 Comment

The algorithm converges very quickly. in the first value $a = 0$ it didnt even iterate (the inital guess was the right solution). in the second value $a = 2$ it took 6 iterations, and for the last value it took 5 iterations. the values of f show quadratic behavior in convergance.