Some MATLAB Programs and Functions

By Dr. Huda Alsaud

Bisection Method

```
%Computes approximate solution of f(x)=0
%Input: function handle f; a,b such that f(a)*f(b)<0,
%         and tolerance tol
%Output: Approximate solution xc
function xc=bisect(f,a,b,tol)
if sign(f(a))*sign(f(b)) >= 0
  error('f(a)f(b)<0 not satisfied!') %ceases execution
end
fa=f(a);
fb=f(b);
while (b-a)/2>tol
  c=(a+b)/2;
  fc=f(c);
  if fc == 0
%c is a solution, done
break
end
if sign(fc)*sign(fa)<0  %a and c make the new interval
  b=c;fb=fc;
else                    %c and b make the new interval
  a=c;fa=fc;
end
end
xc=(a+b)/2;
%new midpoint is best estimate
```

Fixed Point Iteration

```
%Computes approximate solution of g(x)=x
%Input: function handle g, starting guess x0,
%        number of iteration steps k
%Output: Approximate solution xc
function xc=fpi(g, x0, k)
x(1)=x0;
for i=1:k
  x(i+1)=g(x(i));
end
xc=x(k+1);
```

Newton's Method

```
Function root=newton(fname,fdname,x,xtol,ftol,n_max,display)
 % Newton's method.
%input:
% fname string that names the function f(x).
% fdname string that names the derivative f'(x).
% x the initial point
% xtol and ftol termination tolerances
 % n_max the maximum number of iteration
% display = 1 if step-by-step display is desired,
% = 0 otherwise
 %output: root is the computed root of f(x)=0
 %
n = 0;
 fx = feval(fname,x);
 if display,
disp(' n x f(x)')
disp('-------------------------------------')
disp(sprintf('%4d %23.15e %23.15e', n, x, fx))
end
if abs(fx) <= xtol
root = x;
return
end
for n = 1:n_max
 fdx = feval(fdname,x);
 d = fx/fdx;
x = x - d;
 fx = feval(fname,x);
if display,
disp(sprintf('%4d %23.15e %23.15e',n,x,fx))
end
 if abs(d) <= xtol | abs(fx) <= ftol
root = x;
return
end
end
```

MATLAB Functions for Root Finding Problem

- `x = fzero(fun,x0)`

For finding a root of a general function. `x = fzero(fun,x0)` tries to find a point `x` where `fun(x) = 0`. This solution is where `fun(x)` changes sign—`fzero` cannot find a root of a function such as x^2.
The `fzero` command is a function file. The algorithm uses a combination of bisection, secant, and inverse quadratic interpolation methods

For more details:
https://www.mathworks.com/help/matlab/ref/fzero.html

- `r = roots(p)`

For finding all roots of the polynomial p. `r = roots(p)` returns the roots of the polynomial represented by p as a column vector. Input `p` is a vector containing `n+1` polynomial coefficients, starting with the coefficient of x^n.

For more details:

https://www.mathworks.com/help/matlab/ref/roots.html

Gaussian Elimination

```
function [x]=Guss(A,b)
N=length(b);
% Gaussian elimination
for j=1:(N-1)
    if A(j,j)==0 %Check if a(i,i)=0 or not
        for k=j+1:N
            if A(k,j)~=0
                [h]=A(k,:);
                A(k,:)=A(j,:);
                A(j,:)=[h];
                t=b(k);
                b(k)=b(j);
                b(j)=t;
                break
            elseif k==N
                    fprintf('The system has no unique
solution')
                    return
            end
        end
    end
    for i=j+1:N
        m=A(i,j)/A(j,j);
        A(i,:)=A(i,:)-m*A(j,:);
        b(i)=b(i)-m*b(j);
    end
    end
    if A(N,N)==0%check if a(n,n)=0 or not
        fprintf('The system has no unique solution')
        return
    else
%back substitution
x(N)=b(N)/A(N,N);
for i=N-1:-1:1
    h=0;
    for j=(i+1):N
        h=h+A(i,j)*x(j);
    end
    x(i)=(b(i)-h)/A(i,i);
end

    end
    end
```

Lu Factorization of a Square Matrix Using No Row Exchanges

```
function [L, U] = slu(A)
% slu  LU factorization of a square matrix using *no row
exchanges*.
%
% [L, U] = slu(A) uses Gaussian elimination to compute a unit
 % lower triangular L and an upper triangular U so that L*U =
A.
 % The algorithm will stop if a pivot entry is very small.
%
 % See also slv, plu, lu.
 [n, n] = size(A);
 for k = 1:n
 if abs(A(k, k)) < sqrt(eps)
   disp(['Small pivot encountered in column ' int2str(k) '.'])
  end
L(k, k) = 1;
 for i = k+1:n
 L(i,k) = A(i, k) / A(k, k);
     for j = k+1:n
    A(i, j) = A(i, j) - L(i, k)*A(k, j);
    end
  end
  for j = k:n
    U(k, j) = A(k, j);
 End
 end
```

MATLAB Functions for solving linear system

- LU matrix factorization

```
Y = lu(A)
[L,U] = lu(A)
[L,U,P] = lu(A)
```

The `lu` function expresses a matrix `A` as the product of two essentially triangular matrices, one of them a permutation of a lower triangular matrix and the other an upper triangular matrix. The factorization is often called the *LU*, or sometimes the *LR*, factorization. `A` can be rectangular.

`Y = lu(A)` returns matrix `Y` that contains the strictly lower triangular `L`, i.e., without its unit diagonal, and the upper triangular U as submatrices. That is, if `[L,U,P] = lu(A)`, then `Y = U+L-eye(size(A))`. The permutation matrix P is not returned.

`[L,U] = lu(A)` returns an upper triangular matrix in U and a permuted lower triangular matrix in `L` such that `A = L*U`. Return value `L` is a product of lower triangular and permutation matrices.

`[L,U,P] = lu(A)` returns an upper triangular matrix in `U`, a lower triangular matrix `L` with a unit diagonal, and a permutation matrix `P`, such that `L*U = P*A`. The statement `lu(A,'matrix')` returns identical output values.

For more details:
https://www.mathworks.com/help/matlab/ref/lu.html

- x = A\B

`x = A\B` solves the system of linear equations `A*x = B`. The matrices `A` and `B` must have the same number of rows. MATLAB® displays a warning message if `A` is badly scaled or nearly singular, but performs the calculation regardless.

For more details:
https://www.mathworks.com/help/matlab/ref/mldivide.html

References:

1. Numerical Analysis by Timothy Sauer, Addison Wesley.
2. http://web.mit.edu/18.06/www/Course-Info/Tcodes.html.