NUMERICAL LINEAR ALGEBRA

HW - 4.1
HW - 4.2
HW - 4.3
HW - 4.4

Murat Çabuk

HW - 4.1

Explain the difference between accuracy and precision.

Explanation

We can explain with this example (from Wikipedia).

Imagine a study evaluating a new test that screens people for a disease. Each person taking the test either has or does not have the disease. The test outcome can be positive (classifying the person as having the disease) or negative (classifying the person as not having the disease). The test results for each subject may or may not match the subject's actual status. In that setting:

True positive: Sick people correctly identified as sick

False positive: Healthy people incorrectly identified as sick

True negative: Healthy people correctly identified as healthy

False negative: Sick people incorrectly identified as healthy

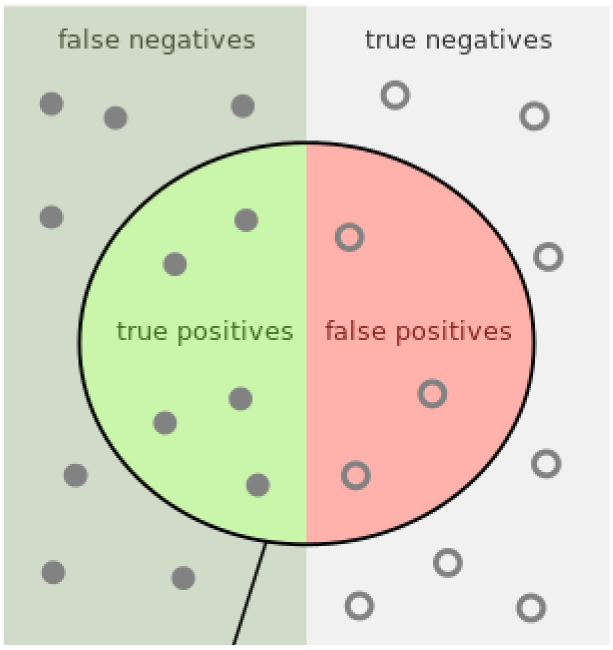
In general, Positive = identified and negative = rejected. Therefore:

True positive = correctly identified

False positive = incorrectly identified

True negative = correctly rejected

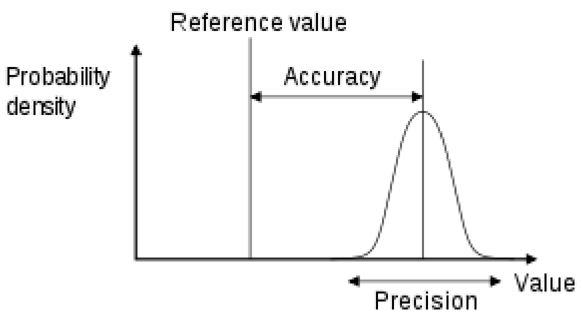
False negative = incorrectly rejected



$$\text{Precision} = \frac{tp}{tp + fp}$$

$$\text{Accuracy} = \frac{tp + tn}{tp + tn + fp + fn}$$

Accuracy indicates proximity of measurement results to the true value, precision to the repeatability or reproducibility of the measurement.



HW - 4.2

Write a Matlab program to solve a set of nonlinear equations by Newton-Ramphson Method.

Explanation

```
function result= newton_raphson(func,initial,max_count, tolerance)
%
% Purpose : To solve the non-linear equation using newton raphson
% Input(func) : non-linear function (f = @(x)
% Input(initial) : initial value of f(x)
% Input(max_count) : maximum iteration
% Input(tolerance) : acceptable tolerance for result
% Output : result
% Remarks : %sample func = f = @(x) x^3 -x -1;
%call sample : t = newton_raphson(f, 1, 10, 0.01)


syms x;
myFunc = func(x)
diffFunc = diff(myFunc);
y = initial;
y_old=0;
tolerance_new = tolerance+1;
count = 0;

while tolerance_new > tolerance && count < max_count
    tolerance_new = 0; % in the begining we suppose our result is correct
    count= count + 1;
    a = subs(myFunc,x,y);
    b = subs(diffFunc,x,y);
    y_old = y;
    y = y - double(a)/double(b); % Newton Raphson

    % if difference between new result and old is not really clouse to zero
    if((abs((y_old(1))-(y(1)))) > (1e-4))
    tolerance_new = abs((y_old(1))-(y(1)))
    end

result = y;
end
```

HW - 4.3

Write a Matlab program to compare all splitting methods.

X = mySplittingMethods(A,b,X,i)

Answers

There are two functions.

- mySplittingMethods(A,b,x_init,tolerance, maxcount)
this function tries to solve our problem in three method (jacobian, seidel and relaxation).

For Ax=b

A: Sequare Matrix

b: Result Vector

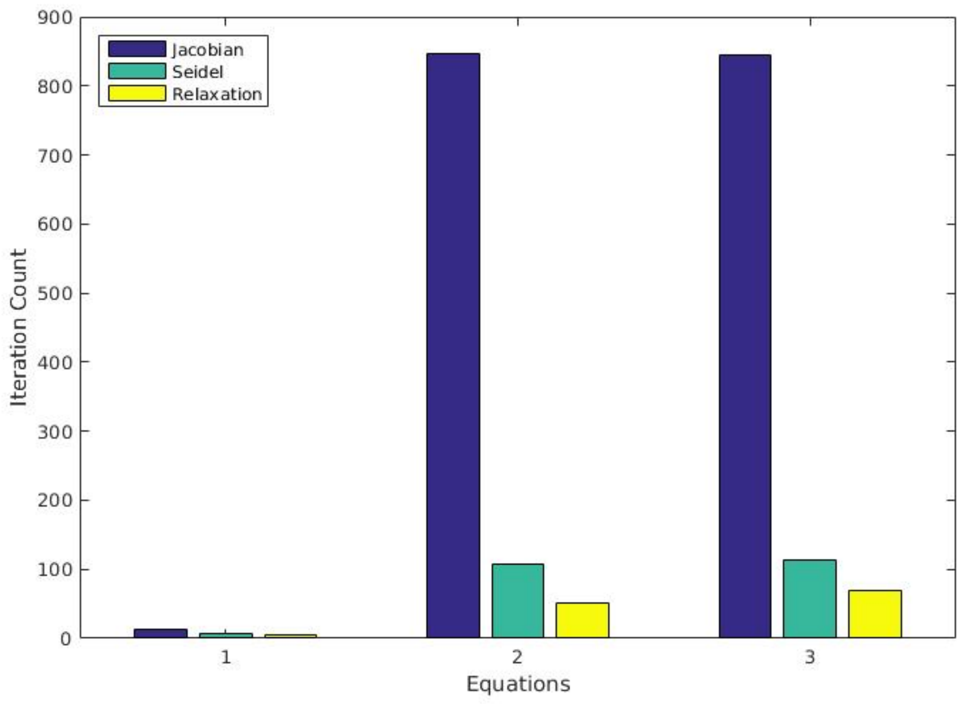
x_init : initial vector

tolerance : acceptable error (float)

`maxcount : maximum iteration (integer)`

-mySplittingMethodsTest()
this function tests our function and crates a plot.

As you can see jacobian always takes long time.



-

mySplittingMethodsTest.m

```
function mySplittingMethodsTest()

A=[2,1,0;
      1,3,1;
      0,1,1];

b=[7,8,5];

x_init=[0;0;0]

[time1 count1]= mySplittingMethods(A,b,x_init,0.01,20000)

A=[-1,1,0, 3,5,6;
   -2,3,1,-2,3,4;
    0,1,1,-5 2,4;
   -1,1,0, 3,5,6;
   -2,3,1,-2,3,4;
    0,1,1,-5 2,4];

x_init=[-1;-2;-3;2;5;6]

b=[7;8;5;6;4;3]

[time2 count2]=mySplittingMethods(A,b,x_init,0.01,20000)

x_init=[-4;0;-3;-10;4;5]

[time3 count3]=mySplittingMethods(A,b,x_init,0.01,20000)

y=[count1;count2;count3];
t=[time1 time2 time3]


bar1 = bar(y);

xlabel('Equations');

ylabel('Iteration Count');

set(bar1(3),'DisplayName','Jacobian');
set(bar1(2),'DisplayName','Seidel');
set(bar1(1),'DisplayName','Relaxation');

legend('Jacobian','Seidel','Relaxation','Location','northwest');
```

mySplittingMethods.m

```matlab
function [time,count] = mySplittingMethods(A,b,x_init,tolerance, maxcount)
x0=x_init;

jacobi_count = 0;
seidel_count = 0;
relaxation_count = 0;

tic;

%_____________________________________ Jacobi method
xnew=x0;
error=1;
while error>tolerance
   xold=xnew;
   jacobi_count = jacobi_count + 1;
   for i=1:length(xnew)
      off_diag = [1:i-1 i+1:length(xnew)];
      xnew(i) = 1/A(i,i)*( b(i)-sum(A(i,off_diag)*xold(off_diag)) );
   end
   error=norm(xnew-xold)/norm(xnew);
end
x_jacobian = xnew

jacobian_time = toc;

tic;

%___________________________________________Gauss Seidel
relax_parameter=1;
n=length(x0);
x=x0;
error=1;
iter = 0;
while (error>tolerance & iter<maxcount)
   xold=x;

   seidel_count= seidel_count + 1;

   for i=1:n
      I = [1:i-1 i+1:n];
      x(i) = (1-relax_parameter)*x(i)+relax_parameter/A(i,i)*( b(i)-A(i,I)*x(I) );
   end
   error = norm(x-xold)/norm(x);
   iter = iter+1;
end

x_seidel=x

seidel_time=toc;

tic;

%______________________________________________relaxation

relax_parameter=1.2;
n=length(x0);
x=x0;
error=1;
iter = 0;
while (error>tolerance & iter<maxcount)
```

```
    xold=x;

    relaxation_count = relaxation_count + 1;

    for i=1:n
        I = [1:i-1 i+1:n];
        x(i) = (1-relax_parameter)*x(i)+relax_parameter/A(i,i)*( b(i)-A(i,I)*x(I) );
    end
    error = norm(x-xold)/norm(x);
    iter = iter+1;
end

x_relaxation=x;

relaxation_time = toc;

time(1)=jacobian_time;
time(2)=seidel_time;
time(3)=relaxation_time;

count(1)= jacobi_count;
count(2)=seidel_count;
count(3)=relaxation_count;

return
```