WROCLAW UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF ELECTRONICS

FIELD: Electronics
SPECIALITY: Advanced Applied Electronics

Numerical Methods:
Least Squares

AUTHOR:
Jaroslaw M. Szumega

SUPERVISOR:
Rafal Zdunek, D.Sc, K-4/W4

GRADE:

WROCŁAW 06.04.2017

Contents

Chapter 1

Solution to the given problems

The solutions below are the requested numerical results of requested problems as well as the execution time's comparison between different algorithms. To ensure proper timings, all measurements are made for one thousands rounds/repetition – it will help to reduce the impact of setting up the Octave engine and other overheads.

Problem 1: Find the solution that best approximates the system of inconsistent linear equations:

$$\text{(a)} \begin{cases} 3x_1 - x_2 = 4 \\ x_1 + 2x_2 = 0 \\ 2x_1 + x_2 = 1 \end{cases}, \quad \text{(b)}, \begin{cases} 3x_1 + x_2 + x_3 = 6 \\ 2x_1 + 3x_2 - x_3 = 1 \\ 2x_1 - x_2 + x_3 = 0 \\ 3x_1 - 3x_2 + 3x_3 = 8 \end{cases}, \quad \text{(c)} \begin{cases} x_1 + x_2 - x_3 = 5 \\ 2x_1 - x_2 + 6x_3 = 1 \\ -x_1 + 4x_2 + x_3 = 0 \\ 3x_1 + 2x_2 - x_3 = 6 \end{cases}.$$

Each system was solved using set of algorithms dedicated for solving Least Squares problem: there was LS approximation, solving LS using Singular Vector Decomposition (SVD) or QR factorization. In addition the first system is solved also by linear regression.

```
Matrix A
Classical LS
    time =   0.12578
SVD LS
    time =   0.78760
QR LS
    time =   0.061208
regression
    time =   0.087359

Matrix B
Classical LS
    time =   0.043440
SVD LS
    time =   0.86493
QR LS
    time =   0.059717

Matrix C
Classical LS
    time =   0.041433
SVD LS
    time =   0.81831
QR LS
    time =   0.057312
```

```
Classical LS
x1 =

   1.04819
  -0.67470

b =

   3.81928
  -0.30120
   1.42169

SVD LS
x11 =

   1.04819
  -0.67470

b =

   3.81928
  -0.30120
   1.42169

QR LS
x111 =

   1.04819
  -0.67470

b =

   3.81928
  -0.30120
   1.42169

regressionl
x1111 =

   2.02857
  -0.54286

b =

   6.62857
   0.94286
   3.51429
```

```
Classical LS
x2 =

  -1.6667
   3.8333
   7.9167

b =

   6.75000
   0.25000
   0.75000
   7.25000

SVD LS
x22 =

  -1.6667
   3.8333
   7.9167

b =

   6.75000
   0.25000
   0.75000
   7.25000

QR LS
x222 =

  -1.6667
   3.8333
   7.9167

b =

   6.75000
   0.25000
   0.75000
   7.25000
```

```
Classical LS
x3 =

   1.80722
   0.55854
  -0.38097

b =

   2.746728
   0.770074
   0.045985
   6.919703

SVD LS
x33 =

   1.80722
   0.55854
  -0.38097

b =

   2.746728
   0.770074
   0.045985
   6.919703

QR LS
x333 =

   1.80722
   0.55854
  -0.38097

b =

   2.746728
   0.770074
   0.045985
   6.919703
```

Figure 1.1 Calculations results. Also verification in form b = Ax

Problem 2: Find the least squares approximating function of the form $a_0 + a_1 x^2 + a_2 \sin\left(\frac{\pi x}{2}\right)$ for each of the following sets of data pairs:

(a) (0,3), (1,0), (1,-1), (-1,2)

(b) (-1, 0.5), (0,1), (2,5), (3,9)

The selected systems was written in the form of matrices. In Octave code it is like this:

```
A1= [1 0 sin(3.14*0/2);
1 1 sin(3.14*1/2);
1 1 sin(3.14*1/2);
1 1 sin(3.14*(-1)/2); ]

b1 = [3; 0; -1; 2]


# b)
A2 = [1 1 sin(3.14*(-1)/2);
1 0 sin(3.14*0/2);
1 4 sin(3.14*2/2);
1 9 sin(3.14*3/2); ]

b2 = [0.5; 1; 5; 9]
```

Then the algorithms were applied to get the solution.

```
Classical LS
x1 =

   3.0000
  -2.2500
  -1.2500

b =

   3.00000
  -0.50000
  -0.50000
   2.00000

SVD LS
x11 =

   3.0000
  -2.2500
  -1.2500

b =

   3.00000
  -0.50000
  -0.50000
   2.00000

QR LS
x111 =

   3.0000
  -2.2500
  -1.2500

b =

   3.00000
  -0.50000
  -0.50000
   2.00000
```

```
Classical LS
x2 =

   0.89905
   1.04988
   1.39836

b =

   0.55057
   0.89905
   5.10079
   8.94959

SVD LS
x22 =

   0.89905
   1.04988
   1.39836

b =

   0.55057
   0.89905
   5.10079
   8.94959

QR LS
x222 =

   0.89905
   1.04988
   1.39836

b =

   0.55057
   0.89905
   5.10079
   8.94959
```

As the picture shows, the calculated coefficients gives quite a good approximation, when using them in computing the "b" vector.

The listing below show also timings of selected algorithms:

```

Case A:

Classical LS
    time =  0.050778
SVD LS
    time =  0.73374
QR LS
    time =  0.051985


Case B:

Classical LS
    time =  0.034670
SVD LS
    time =  0.71804
QR LS
    time =  0.051468
```

Problem 3: The yield y of wheat in quintals per hectare appears to be a linear function of the number of days x_1 of sunshine, the number of centimeters x_2 of rainfall, and the number of kilograms x_3 of fertilizer per hectare. Find the best fit to the data in the table with an equation in the form: $y = a_0 + a_1x_1 + a_2x_2 + a_3x_3$.

y	x_1	x_2	x_3
28	50	18	10
30	40	20	16
21	35	14	10
23	40	12	12
23	30	16	14

The problem mentioned above should be presented in form of matrix: Once again, the

$$A = \begin{bmatrix} 1 & 50 & 18 & 10 \\ 1 & 40 & 20 & 16 \\ 1 & 35 & 14 & 10 \\ 1 & 40 & 12 & 12 \\ 1 & 30 & 16 & 14 \end{bmatrix}, b = \begin{bmatrix} 28 \\ 30 \\ 21 \\ 23 \\ 23 \end{bmatrix}, x = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

set of selected algorithms will be used. This time there also will be selected the Thikonov regularization.

```
Classical LS        SVD LS              QR LS               Tikhonov, alpha = 1
alpha1 =            alpha2 =            alpha3 =            alpha4 =

  -11.10958           -11.10958           -11.10958           -0.24434
    0.32523             0.32523             0.32523            0.18006
    0.73297             0.73297             0.73297            0.77874
    0.99586             0.99586             0.99586            0.49767

b =                 b =                 b =                 b =

   28.304              28.304              28.304              27.753
   29.240              29.240              29.240              28.695
   20.494              20.494              20.494              21.937
   22.645              22.645              22.645              22.275
   24.317              24.317              24.317              24.585
```

Figure 1.2 Results.

Looking at 1.2 we can see, that all the solutions during the verification give quite good approximation of "b". The first three has exactly the same values and are much better than last algorithm. However, Tikhonov regularization gives the coefficients that stand out of the rest of solutions, but this solution is still correct.

And the timing comparison:

```
Classical LS
time =  0.064596


SVD LS
time =  0.75261


QR LS
time =  0.052665


Gen. Tikhonov, alpha = 1
time =  0.041787
```

Problem 4: Using least squares find the "best" straight-line fit and the error estimates for the slope and intercept of that line for the following set of data:

x_i	1	2	3	4	5	6	7	8
y_i	1.5	2.0	2.8	4.1	4.9	6.3	5.0	11.5

At first, the problem presented in matrices: The results below show the solution computed

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \\ 1 & 5 \\ 1 & 6 \\ 1 & 7 \\ 1 & 8 \end{bmatrix}, b = \begin{bmatrix} 1.5 \\ 2 \\ 2.8 \\ 4.1 \\ 4.9 \\ 6.3 \\ 5 \\ 11.5 \end{bmatrix}, x = \begin{bmatrix} a_0 \\ a_1 \end{bmatrix}$$

by different methods. The first three methods are pretty the same answers.

Classical LS	SVD LS	QR LS	Tikhonov, alpha = 1	Linear regression
alpha1 =	alpha2 =	alpha3 =	alpha4 =	alpha 5 =
-0.39643	-0.39643	-0.39643	-0.17322	0.75045
1.14643	1.14643	1.14643	1.10164	0.89638
b =	b =	b =	b =	b =
0.75000	0.75000	0.75000	0.92842	1.6468
1.89643	1.89643	1.89643	2.03005	2.5432
3.04286	3.04286	3.04286	3.13169	3.4396
4.18929	4.18929	4.18929	4.23333	4.3360
5.33571	5.33571	5.33571	5.33497	5.2323
6.48214	6.48214	6.48214	6.43661	6.1287
7.62857	7.62857	7.62857	7.53825	7.0251
8.77500	8.77500	8.77500	8.63989	7.9215

There were also calculated the timings and the errors of the selected methods:

```
Classical LS
    error =  3.8985
    time =  0.043488

SVD LS
    error =  3.8985
    time =  0.72264

QR LS
    error =  3.8985
    time =  0.054575

Tikhonov, alpha = 1
    error =  3.9098
    time =  0.057088

Linear regression
    error =  4.9385
    time =  0.071316
```

As it turned out, the best algorithm considering both time and error is the classical LS fitting. The SVD and QR based algorithms gave the same solution, but were slower (especially SVD, which also in previous tasks was slow).

Problem 5: A missile is fired from enemy territory, and its position in flight is observed by radar tracking devices at the following positions:

x_i [km]	0	250	500	750	1000
y_i [km]	0	8	15	19	20

Suppose our intelligence sources indicate that enemy missiles are programmed to follow a parabolic flight path. Predict how far down range the missile will land.

In this case, as the position of the missle can be described using parabola, so here we are considering the polynomial of second degree.

$$y = a_0 + a_1 x + a_2 x^2$$

$$A = \begin{bmatrix} 1 & 0 & 0^2 \\ 1 & 250 & 250^2 \\ 1 & 500 & 500^2 \\ 1 & 750 & 750^2 \\ 1 & 1000 & 1000^2 \end{bmatrix}, b = \begin{bmatrix} 0 \\ 8 \\ 15 \\ 19 \\ 20 \end{bmatrix}, x = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}$$

Using different methods, the following solutions were obtained:

```
Classical LS        SVD LS             QR LS              Tikhonov, alpha = 1   TSVD
alpha1 =            alpha2 =           alpha3 =           alpha4 =              alpha6 =

  -2.2857e-01         -2.2857e-01        -2.2857e-01        -1.2115e-01           -2.2857e-01
   3.9829e-02          3.9829e-02         3.9829e-02         3.9454e-02            3.9829e-02
  -1.9429e-05         -1.9429e-05        -1.9429e-05        -1.9151e-05           -1.9429e-05

bc =                bc =               bc =               bc =                  bc =

   -0.22857            -0.22857           -0.22857           -0.12115              -0.22857
    8.51429             8.51429            8.51429            8.54541               8.51429
   14.82857            14.82857           14.82857           14.81810              14.82857
   18.71429            18.71429           18.71429           18.69692              18.71429
   20.17143            20.17143           20.17143           20.18187              20.17143
```

We need to compare the results according to the error and computation time. Then we choose one of them.

```
Classical LS
    error =  0.67612
    time =  0.049904

SVD LS
    error =  0.67612
    time =  0.71043

QR LS
    error =  0.67612
    time =  0.053302

Tikhonov, alpha = 1
    error =  0.68569
    time =  0.041042

TSVD
    error =  0.67612
    time =  0.44019

General-Cross Validation
    error =  0.68445
    time =  0.045696
```

There also was a try to refine the result a little (we choose the classical LS result):

```
x = refinement(A, alpha1, b, 100);
bc = A*x;
disp(["Solution refinement:"])
error = norm(bc - b)
```

As it can be seen, after 100 rounds of refinement algorithm, there was not any better solution computed.

```
Solution refinement:
error =  0.67612
```

Therefore the polynomial can be stated as:
$y = -0.22857 + 0.039829x - 0.000019429x^2$

The plot of the line described by y is presented on the figure: The rocket should land

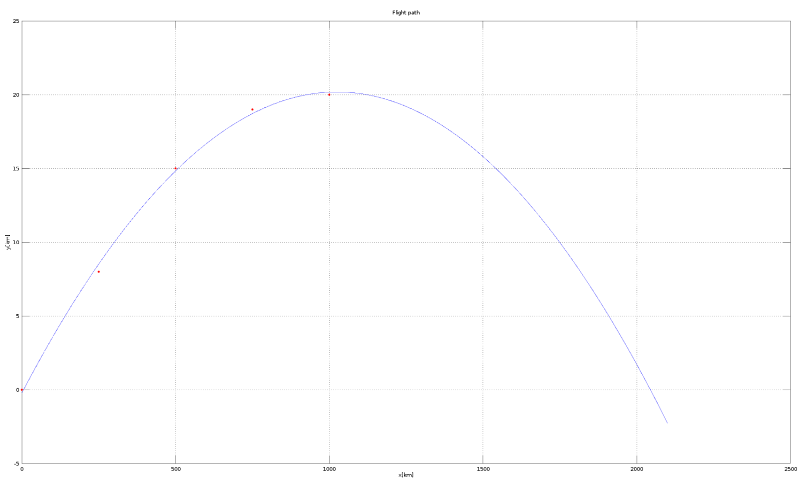

Figure 1.3

around 2100 km away from point zero. However, we can check the equation roots to determine the more accurate landing position: The roots of the presented equation are:

$x_1 = 5.7550 x_2 = 2044.2450$

According to LS solution, the land-point is around 2044km from point zero.

Problem 6: Using least squares techniques, fit the following data

x	-5	-4	-3	-2	-1	0	1	2	3	4	5
y	2	7	9	12	13	14	14	13	10	8	4

with a line $y = a_0 + a_1 x$ and then fit the data with a quadratic $y = a_0 + a_1 x + a_2 x^2$. Determine which of these two curves best fits the data by computing the l_2 norm of the errors in each case.

$$Aq = \begin{bmatrix} 1 & -5 & -5^2 \\ 1 & -4 & -4^2 \\ 1 & -3 & -3^2 \\ 1 & -2 & -2^2 \\ 1 & -1 & -1^2 \\ 1 & 0 & 0^2 \\ 1 & 1 & 5^2 \\ 1 & 2 & 4^2 \\ 1 & 3 & 3^2 \\ 1 & 4 & 2^2 \\ 1 & 5 & 1^2 \end{bmatrix}, \quad A = \begin{bmatrix} 1 & -5 \\ 1 & -4 \\ 1 & -3 \\ 1 & -2 \\ 1 & -1 \\ 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \\ 1 & 5 \end{bmatrix}, b = \begin{bmatrix} 2 \\ 7 \\ 9 \\ 12 \\ 13 \\ 14 \\ 14 \\ 13 \\ 10 \\ 8 \\ 4 \end{bmatrix}, x = \begin{bmatrix} a_0 \\ a_1 \end{bmatrix}$$

Figure 1.4 Task's input data, is presented in the required form - matrices.

For the given data, the solutions were calculated for bot line and quadratic functions:
$alpha = \ [9.63636 \quad 0.18182]$
$alphaq = [13.97203 \ \ 0.18182 \quad -0.43357]$

According to results, the functions can be written.

$y = 0.1818x + 9.63636 \qquad y_q = -0.43357x^2 + 0.18182x + 13.97203$

L_2 -norms of the errors were also computed, as it was one of the tasks:

$l_{2line} = 12.76 \qquad\qquad l_{2quad} = 1.27$

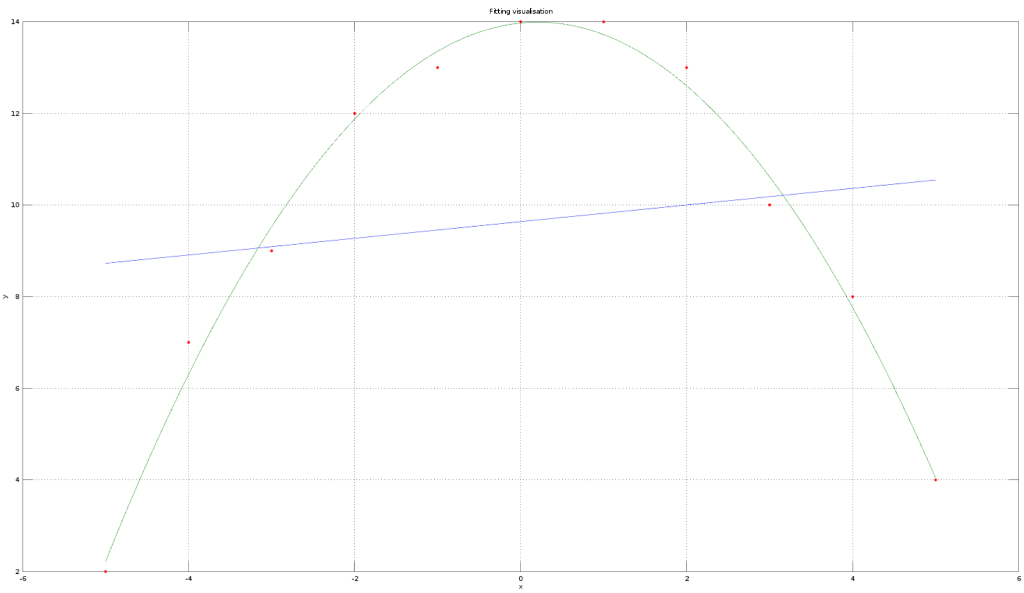

Figure 1.5 Final solutions are presented on a plot according to points, line and quadratic function.

Problem 7: Fit the cosine polynomial $g(x)=\sum_{j=0}^{n} c_j \cos jx$ to the function $f(x)=\pi^2-x^2$ to minimize the error $\|f(x)-g(x)\|_2$ in the range $[0 \quad \pi]$. Estimate the error for n = 1, 2, 3, …, 10.

The following Octave code was prepared to solve the following task:

```
x = 0:0.01:pi;

F = pi.^2 - x.^2;
f = F';

#we assume the largest polynomial
n = 10;
for i = 1:n
G(:,i)=cos(i*x);
endfor

#Ax = b, so to this task GC = f

C = svdLS(G,f)
y = G*C;

plot(x,y, "b", x, f, "r");

grid on
xlabel('x')
ylabel('y')
title('Fitting visualisation')
```

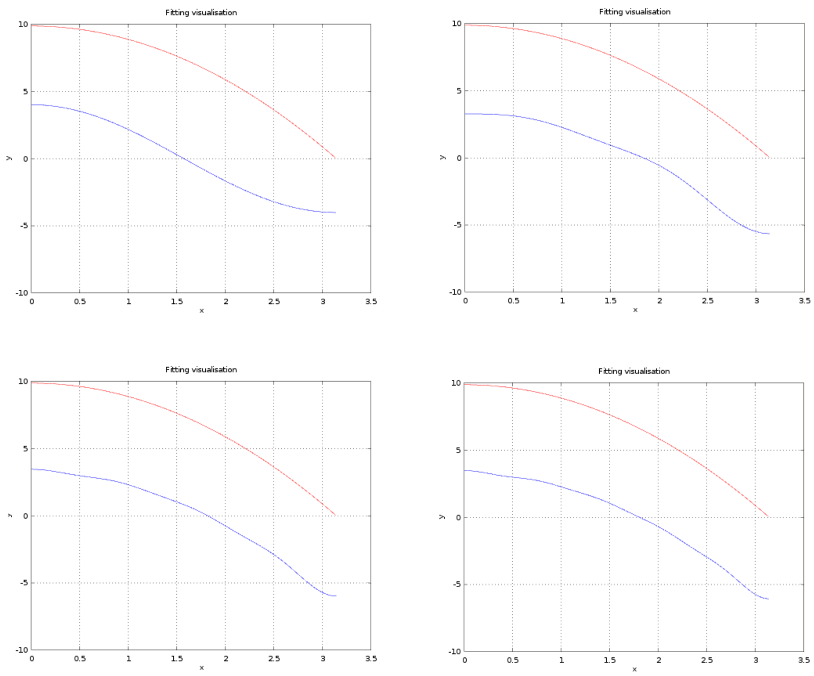

Figure 1.6 Different line fitting (for N = 1,4,7,10)

The error for different n:

```
Error for n = 1
    error =  117.60
Error for n = 2
    error =  116.97
Error for n = 3
    error =  116.83
Error for n = 4
    error =  116.80
Error for n = 5
    error =  116.78
Error for n = 6
    error =  116.78
Error for n = 7
    error =  116.77
Error for n = 8
    error =  116.77
Error for n = 9
    error =  116.77
Error for n = 10
    error =  116.77
```

The error does not vary much according to increasing n value, but of course slight change can be observed.

Line is fitted quite well, but there is certainly a constant missing. However according to g(x) function, we weren't supposed to find a solution including constant value.

Problem 8: Let $f(x) = \sin(\pi x)$ on [0, 1]. The object of this problem is to determine the coefficients α_i of the cubic polynomial $p(x) = \sum_{i=1}^{3} \alpha_i x^i$ that is as close to $f(x)$ as possible in the sense that $r = \int_0^1 |f(x) - p(x)|^2 \, dx$ is as small as possible.

This task objective is to find the cubic polynomial coefficients to minimize the presented integral.
To achieve this, the following file was prepared:

```
#preparing the input data
x = 0:0.1:1;
f = sin(pi*x)';

for i = 1:1:3
p(:,i)=x.^i;
endfor

#different methods
x1 = classicLS(p, f);
x2 = qrLS(p,f);
x3 = svdLS(p,f);
x4 = tikhonovGen(p,f,1);
x5 = tikhonovIt(p,f,100,0.2);
x6 = tsvd(p,f,100);

#calculating the functions according to computed coefficients
Y1 = p.*x1'
Y2 = p.*x2'
Y3 = p.*x3'
Y4 = p.*x4'
Y5 = p.*x5'
Y6 = p.*x6'

#minimizing condition
integral1 = quad(@(x)fun(x,Y1),0,1)
integral2 = quad(@(x)fun(x,Y2),0,1)
integral3 = quad(@(x)fun(x,Y3),0,1)
integral4 = quad(@(x)fun(x,Y4),0,1)
integral5 = quad(@(x)fun(x,Y5),0,1)
integral6 = quad(@(x)fun(x,Y6),0,1)
```

And the following results were obtained:

```
x1 =                x4 =

   3.81784             0.767551
  -3.65750             0.042592
  -0.18863            -0.274183

x2 =                x5 =

   3.81784             3.55956
  -3.65750            -2.85815
  -0.18863            -0.75088

x3 =                x6 =

   3.81784             3.81784
  -3.65750            -3.65750
  -0.18863            -0.18863

    integral1 =  0.31847
    integral2 =  0.31847
    integral3 =  0.31847
    integral4 =  0.57266
    integral5 =  0.33999
    integral6 =  0.31847
```

As it can be seen, the results of every method beside Tikhonov-based, are the same. In case of their equivalence, also the error will be just the same.

Problem 9: For the matrix $\mathbf{A} = \begin{bmatrix} -4 & -2 & -4 & -2 \\ 2 & -2 & 2 & 1 \\ -4 & 1 & -4 & -2 \end{bmatrix}$ and $\mathbf{b} = \begin{bmatrix} -12 & 3 & -9 \end{bmatrix}^T$,

(a) find the solution of $\mathbf{Ax} = \mathbf{b}$ that has the minimum Euclidean norm,

(b) estimate $rank(\mathbf{A})$, $cond(\mathbf{A})$, and $\mathbf{A}^+$,

(c) compute orthogonal projectors onto each of the four fundamental subspaces associated with $\mathbf{A}$,

(d) find the point in $N(A)^\perp$ that is closest to $\mathbf{b}$.

The following code was designed to solve task no. :

```
A = [-4 -2 -4 -2; 2 -2 2 1; -4 1 -4 -2]
b = [-12; 3; 9]

#Task a)
#x1 = classicLS(A,b)
#x2 = qrLS(A,b)
x3 = svdLS(A,b)
x4 = tikhonovGen(A,b,1)
x5 = tikhonovIt(A,b,100,0.2)
x6 = tsvd(A,b,100)

#n1 and n2 cannot be used because of precision
#n1 = norm(A*x1 - b,2)
#n2 = norm(A*x2 - b,2)
n3 = norm(A*x3 - b,2)
n4 = norm(A*x4 - b,2)
n5 = norm(A*x5 - b,2)
n6 = norm(A*x6 - b,2)

#Task b)
rref(A)
rank(A)
Aplus = pseudoinverse(A)

#Task c)
[Pra, Prah, Pna, Pnah] = projectors(A)
```

Calculations of the LS and the norm of the solution:

```
x3 =                norm3 =  12

   0.22222
   3.00000
   0.22222
   0.11111

x4 =                norm4 =  12.034

   0.21951
   2.70000
   0.21951
   0.10976

x5 =                norm5 =  12

   0.22222
   3.00000
   0.22222
   0.11111

x6 =                norm6 =  12

   0.22222
   3.00000
   0.22222
   0.11111
```

According to the RREF, the matrix looks like that:

```
echelo =

   1.00000   0.00000   1.00000   0.50000
   0.00000   1.00000  -0.00000  -0.00000
   0.00000   0.00000   0.00000   0.00000
```

Figure 1.7 RREF of matrix A.

As the figure can idicate, the matrix rank should be equal to 2 (which is exactly the answer compiuted by Octave).

We were also requested to compute the pseudoinverse of Matrix A and its orthonormal projectors:

```
Aplus =

  -0.049383   0.024691  -0.049383
  -0.222222  -0.222222   0.111111
  -0.049383   0.024691  -0.049383
  -0.024691   0.012346  -0.024691

Pra =

   0.88889   0.22222   0.22222
   0.22222   0.55556  -0.44444
   0.22222  -0.44444   0.55556

Prah =

   4.4444e-01  -3.4694e-17   4.4444e-01   2.2222e-01
   5.5511e-17   1.0000e+00   5.5511e-17   2.7756e-17
   4.4444e-01  -3.4694e-17   4.4444e-01   2.2222e-01
   2.2222e-01  -1.7347e-17   2.2222e-01   1.1111e-01

Pna =

   5.5556e-01   3.4694e-17  -4.4444e-01  -2.2222e-01
  -5.5511e-17   3.3307e-16  -5.5511e-17  -2.7756e-17
  -4.4444e-01   3.4694e-17   5.5556e-01  -2.2222e-01
  -2.2222e-01   1.7347e-17  -2.2222e-01   8.8889e-01

Pnah =

   0.11111  -0.22222  -0.22222
  -0.22222   0.44444   0.44444
  -0.22222   0.44444   0.44444
```

Problem 10: For the matrix $\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \\ 13 & 14 & 15 \end{bmatrix}$ and the exact solution $\mathbf{x}_* = [1 \quad 2 \quad 3]^T$,

(a) determine the data vector for the model $\mathbf{A}\mathbf{x} = \mathbf{b}$, then estimate the minimal norm LS solution with the selected regularization methods,
(b) estimate $rank(\mathbf{A})$, $cond(\mathbf{A})$, and $\mathbf{A}^+$,
(c) find $N(\mathbf{A})$, and the missing components that went onto $N(\mathbf{A})$,
(d) estimate the solution error $\|\mathbf{x} - \mathbf{x}_*\|_2$ and the residual error $\|\mathbf{b} - \mathbf{A}\mathbf{x}\|_2$.

The following code was prepared to solve all the requested tasks:

```
A = [1 2 3; 4 5 6; 7 8 9; 10 11 12; 13 14 15]
x_exact = [1 2 3]'
b = A*x_exact

x1 = tsvd(A,b,1000)
x2 = tikhonovIt(A,b,1000,0.1)

Echelon = rref(A)
RankA = rank(A)
APlus = pseudoinverse(A)

error1 = sum((x1-x_exact).^2)
Rerror1 = sum((b-A*x1).^2)
error2 = sum((x2-x_exact).^2)
Rerror2 = sum((b-A*x2).^2)
```

```
A =                           x_exact =      b =

     1     2     3               1              14
     4     5     6               2              32
     7     8     9               3              50
    10    11    12                              68
    13    14    15                              86

x1 =

   1.0000
   2.0000
   3.0000

x2 =

   1.00000
   2.00000
   3.00000

Echelon =

   1   0  -1
   0   1   2
   0   0   0
   0   0   0
   0   0   0

RankA =  2
APlus =

  -3.8889e-01  -2.4444e-01  -1.0000e-01   4.4444e-02   1.8889e-01
  -2.2222e-02  -1.1111e-02   5.6379e-17   1.1111e-02   2.2222e-02
   3.4444e-01   2.2222e-01   1.0000e-01  -2.2222e-02  -1.4444e-01

error1 =    6.1926e-29
Rerror1 =    2.1255e-26
error2 =    4.0820e-24
Rerror2 = 0
```

During computations we obtained exact solution, so all the errors are a result of computer arithmetic and numbers rounding.

Problem 11: Generate the values of the polynomial $y = 40 + 10x + 5x^2 + 3x^3 + 2x^4 + x^5 + x^6$ for $x = 1, 2, \ldots, 14$. First fit the polynomial $y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6$ to the generated data in the LS sense and compare the estimated coefficients $a_0, a_1, \ldots, a_6$ to the true ones. Then perturb the observed data with an additive Gaussian noise $N(0, \sigma^2)$, and illustrate the fitting error (the Euclidean norm) versus the standard deviation σ. Estimate the maximal value of σ for which the fitting error exceeds 10^{-6}, assuming double precision arithmetic operations.

```
a=[40; 10; 5; 3; 2; 1; 1];

%Coefficients
x=1:1:14;
X = x';
A = [X.^0 X.^1 X.^2 X.^3 X.^4 X.^5 X.^6];
%sample x values
y=40+10*x+5*x.^2+3*x.^3+2*x.^4+x.^5+x.^6;

C = tsvd(A,y',10)
C_exact = A\y'
```

```
C =

   40.00000
   10.00000
    5.00000
    3.00000
    2.00000
    1.00000
    1.00000

C_exact =

   40.00000
   10.00000
    5.00000
    3.00000
    2.00000
    1.00000
    1.00000
```

Figure 1.8 Using computations the solution that corresponds to exact coefficients was obtained.

Problem 13: Let $\mathbf{c} = [0 \quad 1 \quad \dots \quad N-1]^T \in \Re^N$ and $\mathbf{r} = [0 \quad -1 \quad \dots \quad -N+1] \in \Re^N$. Assuming $\mathbf{c}$ is the first column and $\mathbf{r}$ is the first row of the Toeplitz, and $\mathbf{x}^* = [1 \quad 2 \quad \dots \quad N]^T \in \Re^N$, perform the forward projection: $\mathbf{Ax}^* = \mathbf{b}$, and

(a) solve the system of linear equations with the selected regularization methods. Estimate the minimal norm LS solution. Which method gives the best approximations?

(b) draw the errors: $r^{(k)} = \left\|\mathbf{b} - \mathbf{Ax}^{(k)}\right\|_2$ and $\left\|\mathbf{x}^* - \mathbf{x}^{(k)}\right\|_2$,

(c) estimate $rank(\mathbf{A})$, $cond(\mathbf{A})$,

```
A =

   0  -1  -2  -3  -4  -5  -6  -7  -8  -9
   1   0  -1  -2  -3  -4  -5  -6  -7  -8
   2   1   0  -1  -2  -3  -4  -5  -6  -7
   3   2   1   0  -1  -2  -3  -4  -5  -6
   4   3   2   1   0  -1  -2  -3  -4  -5
   5   4   3   2   1   0  -1  -2  -3  -4
   6   5   4   3   2   1   0  -1  -2  -3
   7   6   5   4   3   2   1   0  -1  -2
   8   7   6   5   4   3   2   1   0  -1
   9   8   7   6   5   4   3   2   1   0
```

```
x3 =

    1.00000        norm3 =    8.3470e-14
    2.00000
    3.00000
    4.00000
    5.00000
    6.00000
    7.00000
    8.00000
    9.00000
   10.00000
```

```
x5 =

    1.00000        norm5 =    3.5527e-14
    2.00000
    3.00000
    4.00000
    5.00000
    6.00000
    7.00000
    8.00000
    9.00000
   10.00000
```

```
x4 =

   0.99879         norm4 =  0.68230
   1.99758
   2.99637
   3.99516
   4.99395
   5.99274
   6.99153
   7.99031
   8.98910
   9.98789
```

```
x6 =

    1.00000        norm6 =    2.1006e-13
    2.00000
    3.00000
    4.00000
    5.00000
    6.00000
    7.00000
    8.00000
    9.00000
   10.00000
```

```
rankn =  2
condition =    2.3259e+18
Aplus =

  -0.000000   0.001212   0.002424   0.003636   0.004848   0.006061   0.007273   0.008485   0.009697   0.010909
  -0.001212   0.000000   0.001212   0.002424   0.003636   0.004848   0.006061   0.007273   0.008485   0.009697
  -0.002424  -0.001212   0.000000   0.001212   0.002424   0.003636   0.004848   0.006061   0.007273   0.008485
  -0.003636  -0.002424  -0.001212   0.000000   0.001212   0.002424   0.003636   0.004848   0.006061   0.007273
  -0.004848  -0.003636  -0.002424  -0.001212   0.000000   0.001212   0.002424   0.003636   0.004848   0.006061
  -0.006061  -0.004848  -0.003636  -0.002424  -0.001212   0.000000   0.001212   0.002424   0.003636   0.004848
  -0.007273  -0.006061  -0.004848  -0.003636  -0.002424  -0.001212   0.000000   0.001212   0.002424   0.003636
  -0.008485  -0.007273  -0.006061  -0.004848  -0.003636  -0.002424  -0.001212   0.000000   0.001212   0.002424
  -0.009697  -0.008485  -0.007273  -0.006061  -0.004848  -0.003636  -0.002424  -0.001212   0.000000   0.001212
  -0.010909  -0.009697  -0.008485  -0.007273  -0.006061  -0.004848  -0.003636  -0.002424  -0.001212   0.000000
```

Figure 1.9 (Norm on the image are the l2 norm calculated from errors.)

Chapter 2

Algorithms code

Algorithm 1 – The classical LS fitting.

```
function [x] = classicLS(A, b)

[m,n] = size(A)

if(m >= n || rank(A) == n)
    disp(["There is an unique solution"])
    x = inv(A' * A) * A' * b;
else if ( m < n)
    disp(["Underdetermined system"])
    x = A' * inv(A * A') * b;
end
endfunction
```

Algorithm 2 – Pseudoinverse

```
function [x] = pseudoinverse(A)

[m,n] = size(A);
r = rank(A);
[u,s,v] = svdqr(A,20);

x = zeros(n,m);

for i = 1:r
    x = x + inv(s(i,i)) *v(:,i)* u(:,i)';
endfor

endfunction
```

Algorithm 3 – The orthogonal projectors

```
function [Pra, Prah, Pna, Pnah] = projectors(A)

[m,n] = size(A)

Pra = A * pseudoinverse(A)
Prah = pseudoinverse(A) * A

Pnah = eye(size(Pra)) - Pra
Pna = eye(size(Prah)) - Prah

endfunction
```

Algorithm 4 – The LS solution by SVD

```
function [x] = svdLS(A,b)

[u,s,v] = svd(A);

x = (v*pseudoinverse(s) * u') * b;

endfunctiong
```

Algorithm 5 – The LS solution by QR factorization

```
function [U, S, V] = svdQR(A,iterations)

[n, m]= size(A); # can be rectangular matrix
U=eye(n);
V=eye(m);

R=A';

for i = 0:iterations
    [Q,R]=qr(R');   # qr decompositions and updating
    U=U*Q;
    [Q,R]=qr(R');
    V=V*Q;
endfor
S=R';               # S is R transposed

endfunction
```

Algorithm 6 – The linear regression

```
function [x] = regression(A, b)

[m,n] = size(A
[p,r] = size(b);

if(n != 2 || r != 1)
    disp(["The matrix does not describe the polynomial of first degree"
       ])

else
    meanY = sum(b)/p
    meanT = sum(A(:,n))/m

    #beta = (sum(b.*A(:,n)) - m*meanY*meanT)/(sum(A(:,n).^2) - m * meanT
       *meanT)
    #alpha = meanY - beta*meanT

    #more accurate b calculation
    first = (b.- meanY).*(A(:,n).-meanT)
    second = (b.-meanT).^2
    beta = sum(first)/sum(second)
    alpha = meanY - beta*meanT

    x = [alpha;beta]
endif
endfunction
```

Algorithm 7 – The TSVD algorithm

```
function [x] = tsvd(A,b, it)

[u,s,v] = svdqr(A,it);

c = u'*b
r = rank(A)
x=zeros(size(A,2),1);

for i = 1:r
    x = x + (c(i)*v(:,i))/s(i,i);
endfor
endfunction
```

Algorithm 8 – The iterative refinement

```
function [x] = refinement(A,x,b,it)

for s = 1:it
    r = b - A*x

    #extended refinement
    delta = qrLS(A,r);
    x = x + delta
endfor
endfunction
```

Algorithm 10 – The General Cross-Validation

```
function [x] = crossvalidation(A,b, mi)

C = inv(A'*A);
M = A' * A + (mi.^2).^C'*C;

x = inv(M)*A'*b;

endfunction
```

Algorithm 11 – The Iterative Tikhonov Regularization

```
function [x] = tikhonovIt(A,b, it, mi)

[m,n] = size(A);

x=zeros(size(A,2),1);
for i = 1:it
    x = x + inv(A' * A + eye(size(A'*A)).*mi) * A' * (b-A*x);
endfor

endfunction
```

Algorithm ** – The General Tikhonov Regularization

```
function [x] = tikhonovGen(A,b, alpha)

x = inv(A' * A + alpha.*eye(size(A'*A))) * A' * b;

endfunction
```

Bibliography

[1] Björck, Åke. Numerical methods for least squares problems. Society for Industrial and Applied Mathematics, 1996.

[2] C. D. Meyer, Matrix Analysis and Applied Linear Algebra, SIAM, 2000,

[3] Ch. Zarowski, An Introduction to Numerical Analysis for Electrical and Computer Engineers, Wiley, 2004

[4] Zdunek R., Numerical Methods - lecture slides.