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 21.03.2017

Contents

1 Solution to the given problems **1**

2 Algorithms code **10**

2.1 Code for solving particular tasks 14

Bibliography **15**

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

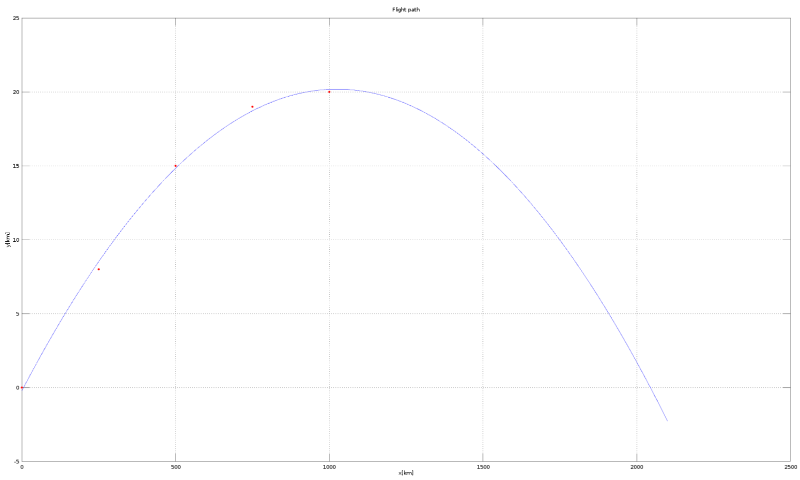

Figure 1.3

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 were also computed, as it was one of the tasks:

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

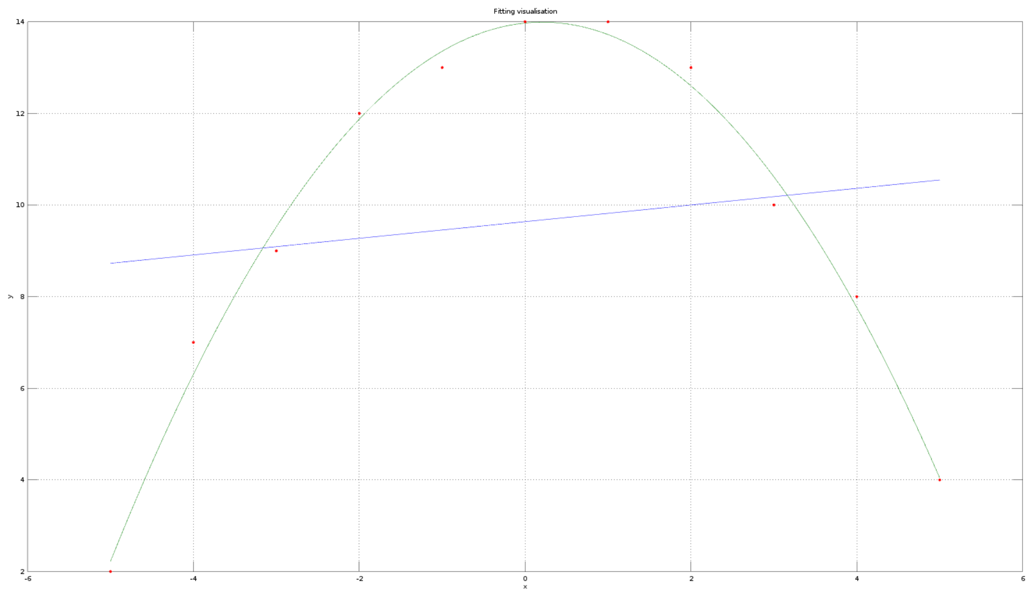

Figure 1.5 Final solutions are presented on a plot according to points, line and quadratic function.

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
```

2.1 Code for solving particular tasks

Task1

```
A1 = [3 -1; 1 2; 2 1];
b1 = [4; 0; 1];

disp(["Classical LS"])
tic
for i = 1:1000
x1 = classicLS(A1,b1);
endfor
time = toc

disp(["SVD LS"])
tic
for i = 1:1000
x11 = svdLS(A1,b1);
endfor
time = toc

disp(["QR LS"])
tic
for i = 1:1000
x111 = qrLS(A1,b1);
endfor
time = toc

disp(["regression1"])
tic
for i = 1:1000
x1111 = regression(A1,b1);
endfor
time = toc

[...]

The similar calculations are performed for the remaining matrices.
```

Bibliography

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

[2] Golub, Gene H., and Charles F. Van Loan. "Matrix computations, 3rd." (1996).

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