

WROCLAW UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF ELECTRONICS

FIELD: Electronics
SPECIALITY: Advanced Applied Electronics

**Numerical Methods:
Eigenproblems**

AUTHOR:
Jaroslaw M. Szumega

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

GRADE:

Contents

1	Solution to the given problems	1
2	Algorithms code	7
2.1	Code for solving particular tasks	11
	Bibliography	12

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:

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

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.

```
1 Matrix A
2 Classical LS
3     time = 0.12578
4 SVD LS
5     time = 0.78760
6 QR LS
7     time = 0.061208
8 regression
9     time = 0.087359
10
11 Matrix B
12 Classical LS
13     time = 0.043440
14 SVD LS
15     time = 0.86493
16 QR LS
17     time = 0.059717
18
19 Matrix C
20 Classical LS
21     time = 0.041433
22 SVD LS
23     time = 0.81831
24 QR LS
25     time = 0.057312
```

Classical LS x1 =	Classical LS x2 =	Classical LS x3 =
1.04819 -0.67470	-1.6667 3.8333 7.9167	1.80722 0.55854 -0.38097
b =	b =	b =
3.81928 -0.30120 1.42169	6.75000 0.25000 0.75000 7.25000	2.746728 0.770074 0.045985 6.919703
SVD LS x11 =	SVD LS x22 =	SVD LS x33 =
1.04819 -0.67470	-1.6667 3.8333 7.9167	1.80722 0.55854 -0.38097
b =	b =	b =
3.81928 -0.30120 1.42169	6.75000 0.25000 0.75000 7.25000	2.746728 0.770074 0.045985 6.919703
QR LS x111 =	QR LS x222 =	QR LS x333 =
1.04819 -0.67470	-1.6667 3.8333 7.9167	1.80722 0.55854 -0.38097
b =	b =	b =
3.81928 -0.30120 1.42169	6.75000 0.25000 0.75000 7.25000	2.746728 0.770074 0.045985 6.919703
regression1 x1111 =		
2.02857 -0.54286		
b =		
6.62857 0.94286 3.51429		

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

Problem 2: Find the least squares approximating function of the form $a_0 + a_1x^2 + a_2 \sin\left(\frac{\pi x}{2}\right)$

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:

```

1 A1= [1 0 sin(3.14*0/2);
2 1 1 sin(3.14*1/2);
3 1 1 sin(3.14*1/2);
4 1 1 sin(3.14*(-1)/2); ]
5
6 b1 = [3; 0; -1; 2]
7
8
9 # b)
10 A2 = [1 1 sin(3.14*(-1)/2);
11 1 0 sin(3.14*0/2);
12 1 4 sin(3.14*2/2);
13 1 9 sin(3.14*3/2); ]
14
15 b2 = [0.5; 1; 5; 9]
```

Then the algorithms were applied to get the solution.

<p>Classical LS x1 =</p> <pre> 3.0000 -2.2500 -1.2500 b = 3.00000 -0.50000 -0.50000 2.00000</pre> <p>SVD LS x11 =</p> <pre> 3.0000 -2.2500 -1.2500 b = 3.00000 -0.50000 -0.50000 2.00000</pre> <p>QR LS x111 =</p> <pre> 3.0000 -2.2500 -1.2500 b = 3.00000 -0.50000 -0.50000 2.00000</pre>	<p>Classical LS x2 =</p> <pre> 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</pre>
--	---

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:

```
1
2 Case A:
3
4 Classical LS
5     time = 0.050778
6 SVD LS
7     time = 0.73374
8 QR LS
9     time = 0.051985
10
11
12 Case B:
13
14 Classical LS
15     time = 0.034670
16 SVD LS
17     time = 0.71804
18 QR LS
19     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 alpha1 =	SVD LS alpha2 =	QR LS alpha3 =	Tikhonov, alpha = 1 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:

```

1 Classical LS
2 time = 0.064596
3
4
5 SVD LS
6 time = 0.75261
7
8
9 QR LS
10 time = 0.052665
11
12
13 Gen. Tikhonov, alpha = 1
14 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:

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

Chapter 2

Algorithms code

Algorithm 1 – The classical LS fitting.

```
1 function [x] = classicLS(A, b)
2
3 [m,n] = size(A)
4
5 if(m >= n || rank(A) == n)
6     disp(["There is an unique solution"])
7     x = inv(A' * A) * A' * b;
8 else if ( m < n)
9     disp(["Underdetermined system"])
10    x = A' * inv(A * A') * b;
11 end
12 endfunction
```

Algorithm 2 – Pseudoinverse

```
1 function [x] = pseudoinverse(A)
2
3 [m,n] = size(A);
4 r = rank(A);
5 [u,s,v] = svdqr(A,20);
6
7 x = zeros(n,m);
8
9 for i = 1:r
10    x = x + inv(s(i,i)) * v(:,i) * u(:,i)';
11 endfor
12
13 endfunction
```

Algorithm 3 – The orthogonal projectors

```

1 function [Pra, Prah, Pna, Pnah] = projectors(A)
2
3 [m,n] = size(A)
4
5 Pra = A * pseudoinverse(A)
6 Prah = pseudoinverse(A) * A
7
8 Pnah = eye(size(Pra)) - Pra
9 Pna = eye(size(Prah)) - Prah
10
11 endfunction

```

Algorithm 4 – The LS solution by SVD

```

1 function [x] = svdLS(A,b)
2
3 [u,s,v] = svd(A);
4
5 x = (v*pseudoinverse(s) * u') * b;
6
7 endfunctiong

```

Algorithm 5 – The LS solution by QR factorization

```

1 function [U, S, V] = svdQR(A,iterations)
2
3 [n, m]= size(A); # can be rectangular matrix
4 U=eye(n);
5 V=eye(m);
6
7 R=A';
8
9 for i = 0:iterations
10     [Q,R]=qr(R'); # qr decompositions and updating
11     U=U*Q;
12     [Q,R]=qr(R');
13     V=V*Q;
14 endfor
15 S=R'; # S is R transposed
16
17 endfunction

```

Algorithm 6 – The linear regression

```

1 function [x] = regression(A, b)
2
3 [m,n] = size(A)
4 [p,r] = size(b);
5
6 if(n != 2 || r != 1)
7     disp(["The matrix does not describe the polynomial of first degree"
8         ])
9 else
10     meanY = sum(b)/p
11     meanT = sum(A(:,n))/m
12
13     #beta = (sum(b.*A(:,n)) - m*meanY*meanT)/(sum(A(:,n).^2) - m * meanT
14         *meanT)
15     #alpha = meanY - beta*meanT
16
17     #more accurate b calculation
18     first = (b.- meanY).*(A(:,n).-meanT)
19     second = (b.-meanT).^2
20     beta = sum(first)/sum(second)
21     alpha = meanY - beta*meanT
22     x = [alpha;beta]
23 endif
24 endfunction

```

Algorithm 7 – The TSVD algorithm

```

1 function [x] = tsvd(A,b, it)
2
3 [u,s,v] = svdqr(A,it);
4
5 c = u'*b
6 r = rank(A)
7 x=zeros(size(A,2),1);
8
9 for i = 1:r
10     x = x + (c(i)*v(:,i))/s(i,i);
11 endfor
12 endfunction

```

Algorithm 8 – The iterative refinement

```

1 function [x] = refinement(A,x,b,it)
2
3 for s = 1:it
4     r = b - A*x
5
6     #extended refinement
7     delta = qrLS(A,r);
8     x = x + delta
9 endfor
10 endfunction

```

Algorithm 10 – The General Cross-Validation

```
1 function [x] = crossvalidation(A,b, mi)
2
3 C = inv(A'*A);
4 M = A' * A + (mi.^2).^C'*C;
5
6 x = inv(M)*A'*b;
7
8 endfunction
```

Algorithm 11 – The Iterative Tikhonov Regularization

```
1 function [x] = tikhonovIt(A,b, it, mi)
2
3 [m,n] = size(A);
4
5 x=zeros(size(A,2),1);
6 for i = 1:it
7     x = x + inv(A' * A + eye(size(A'*A)).*mi) * A' * (b-A*x);
8 endfor
9
10 endfunction
```

Algorithm ** – The General Tikhonov Regularization

```
1 function [x] = tikhonovGen(A,b, alpha)
2
3 x = inv(A' * A + alpha.*eye(size(A'*A))) * A' * b;
4
5 endfunction
```

2.1 Code for solving particular tasks

Task1

```
1 A1 = [3 -1; 1 2; 2 1];
2 b1 = [4; 0; 1];
3
4 disp(["Classical LS"])
5 tic
6 for i = 1:1000
7 x1 = classicLS(A1,b1);
8 endfor
9 time = toc
10
11 disp(["SVD LS"])
12 tic
13 for i = 1:1000
14 x11 = svdLS(A1,b1);
15 endfor
16 time = toc
17
18 disp(["QR LS"])
19 tic
20 for i = 1:1000
21 x111 = qrLS(A1,b1);
22 endfor
23 time = toc
24
25 disp(["regression1"])
26 tic
27 for i = 1:1000
28 x1111 = regression(A1,b1);
29 endfor
30 time = toc
31
32 [...]
33
34 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.