WROCLAW UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF ELECTRONICS

FIELD: Electronics
SPECIALITY: Advanced Applied Electronics

Numerical Methods: Underdetermined systems

AUTHOR:
Jaroslaw M. Szumega

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

GRADE:

WROCŁAW 25.04.2017

Contents

Chapter 1

Solution to the given problems

Problem 1: Find a condition on the numbers a, b, c such that the following system of equations is consistent. When that condition is satisfied, find all solutions (in terms of a, b, and c):

$$\begin{cases} x+3y+z=a \\ -x-2y+z=b \\ 3x+7y-z=c \end{cases}$$

Then choose the numbers a, b, c to satisfy the conditions:

(a) $x=0$, $y=3$, $z=1$

(b) $x=0$, $y=0$, $z=1$

(c) $x=$ -2, $y=$ 1, $z=0$.

For the computed numbers a, b, c, estimate the solutions using the Regularized FOCUSS algorithm.

To find the mentioned condition, the RREF (Row Reduced Echelon Form) will be used.

$$\begin{bmatrix} 1 & 3 & 1 & | & a \\ -1 & -2 & 1 & | & b \\ 3 & 7 & -1 & | & c \end{bmatrix} --RREF--> \begin{bmatrix} 1 & 0 & -5 & | & -2a-3b \\ 0 & 1 & 2 & | & a+b \\ 0 & 0 & 0 & | & -a+2b+c \end{bmatrix}$$

Regarding to the obtained result, the system is consistent only when the condition is fulfilled:

$$-a+2b+c=0$$

The solution interpreted from the RREF matrix is:

$$\begin{aligned} x_1 &= -2a-3b \\ x_2 &= a+b-2x_3 \\ x_3 &= freevariable \end{aligned}$$

In above mentioned solution, the x_3 is a free variable, which means that it is not bounded to any specific value. We can interpret is as a solution's parameter.
To estimate it, we should have some kind of knowledge that will bound it even before the solving process.

```
 b =
 10
 -5
 20

 x_calcA =
 0.00000
 3.00000
 1.00000

 Elapsed time is 0.022619 seconds.

 b_calculatedA =
 10.0000
 -5.0000
 20.0000

solution_error =    4.4052e-07
residual_error =    6.9201e-07

 b =
 1
 1
 -1

 x_calcB =
 0.00000
 0.00000
 1.00000

 Elapsed time is 0.00672603 seconds.

 b_calculatedB =
 1.00000
 1.00000
 -1.00000

solution_error =    3.3345e-07
residual_error =    5.7755e-07

 b =
 1
 0
 1

 x_calcC =
 0.00000
 0.20000
 0.40000

 Elapsed time is 0.00638103 seconds.

 b_calculatedC =
 1.0000e+00
 -5.2708e-07
 1.0000e+00

solution_error =    2.1909
residual_error =    1.1900e-06
```

The calculated solution for cases 'a' and 'b' are corresponding to the data pointed out for them. For case 'c' however, the calculated solution differs quite much.
The 'c' solution is not exact, but the estimate errors are rather small. Knowing the system purpose, it would be possible to assess if the model is usefull.

Problem 2: Perform the forward projection of the exact solution $\mathbf{x} = [1 \quad 0 \quad 1 \quad 1 \quad 0]^T$ onto the range space spanned by the columns in the matrix:

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 2 & 3 & 1 \\ 2 & 4 & 4 & 6 & 2 \\ 3 & 6 & 6 & 9 & 6 \\ 1 & 2 & 4 & 5 & 3 \end{bmatrix}.$$

Assuming the linear forward projection model, try to estimate the true solution to $\mathbf{Ax} = \mathbf{b}$ given $\mathbf{A}$ and $\mathbf{b}$. Then change the a_{21} entry from 2 to 0, and repeat the estimation. Explain the difference. Compute the residual and solution errors. Which algorithm gives the best estimate and why? Which metrics are best to solve this problem?

For the selected problem the multiple algorithms were used for calculations. Both scenarios, indicated in the introduction were taken under consideration.

$a_{21} = 2$

```
Regularized focuss
Elapsed time is 0.00539589 seconds.

x_transposed =
0.00000    1.00000    2.00000    0.00000    0.00000

solution_error =  2.0000
residual_error =    2.3452e-07



MFocuss
Elapsed time is 0.00734401 seconds.

x_transposed =
0.00000    0.00000    0.00000    1.00000    0.00000

solution_error =  1.4142
residual_error =  12.288



Tikhonov
Elapsed time is 0.000128984 seconds.

x_transposed =
0.193639    0.387279    0.877795    1.071435    0.038132

solution_error =  0.90647
residual_error =  0.084187



QR LS
(warning during algorithms operation: matrix singular to machine precision)
Elapsed time is 0.000256062 seconds.

x_transposed =
-0.660375    0.894607    0.894607    1.050346    -0.069089

solution_error =  1.8909
residual_error =  0.32342



SVD LS
Elapsed time is 0.000138998 seconds.

x_transposed =
1.8182e-01    3.6364e-01    9.0909e-01    1.0909e+00    2.3870e-15

solution_error =  0.90453
residual_error =    1.5511e-14
```

$a_{21} = 0$

```
2. Regularized focuss
Elapsed time is 0.0136371 seconds.

ans =
1.00000  -0.50000   0.00000   2.00000   0.00000

solution_error =  1.5000
residual_error =    1.6575e-06



2. MFocuss
Elapsed time is 0.012074 seconds.

ans =
1.00000   0.00000   1.00000   1.00000   0.00000

solution_error =    9.5263e-10
residual_error =    1.3265e-08



2. Tikhonov
Elapsed time is 0.000120878 seconds.

ans =
0.764147   0.092481   0.899610   0.945850   0.214162

solution_error =  0.35079
residual_error =  0.23123



2. QR LS
Elapsed time is 0.000169039 seconds.

ans =
1.0000e+00  -6.6613e-16   1.0000e+00   1.0000e+00  -1.8328e-15

solution_error =    3.4265e-15
residual_error =    4.6998e-15



2. SVD LS
Elapsed time is 0.000123024 seconds.

ans =
1.0000e+00   7.7716e-16   1.0000e+00   1.0000e+00  -2.6645e-15

solution_error =    5.2791e-15
residual_error =    6.9369e-15
```

In the first case, the results did not correspond to the original x vector. In second scenario, the algorithms were able to calculate precise or almost exact value of x. It is mainly due to the condition number of both matrices:

- $a_{21} = 2$: cond(A) $= 2e16$
- $a_{21} = 0$: cond(A) $= 28$

In the second case (which gave better result), the algorithms that calculated the most optimal solution were MFOCUSS and variations of LS for under-determined systems (QR, SVD options).

Problem 3: Generate 5 sparse instantaneous signals such that at each time sample at most 3 signals are non-zeros. Then perform the forward projection of each time sample onto the range space spanned by the columns of the matrix:

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 2 & 3 & 1 \\ 0 & 4 & 4 & 6 & 2 \\ 3 & 6 & 6 & 9 & 6 \\ 1 & 2 & 4 & 5 & 3 \end{bmatrix}.$$

Then estimate these signals using the Regularized FOCUSS and M-FOCUSS algorithms. Which algorithm gives better results and why? How to choose the regularization parameter?

To start, the 5 signals had to be generated. The Octave code shown below did this, the comments are attached.

```
#generation of five signals , now 100 samples each
x = randn(5,100);

#convert to "discrete", it will help with estimating the "non-zeros" condition
x(x < 0) = 0;
x(x > 0) = 1;

#sum to find when condition "at most 3 non zeros at the time" is exceeded
E = sum(x,1);
n = find(E>3);

#and wipe when more than 3 are non-zeros
x(:,n) = [];
```

At this point, at least quite a few signals are generated according to the condition indicated in the task. However, we only need 5 of them.

The results are presented below:

```
Signal no. 1
0    0    1    1    0

Focuss
xFOCUSS_transposed =
0.00000    0.00000    1.00277    1.01478    0.00000

solution_error =  0.015034
residual_error =  0.20503


MFocuss
xMFOCUSS_transposed =
0.00000    0.00000    0.99895    1.00187    0.00000

solution_error =  0.0021402
residual_error =  0.014051


Signal no. 2
1    0    1    0    1

Focuss
xFOCUSS_transposed =
0.99968    0.00000    1.00127    0.00000    1.00182

solution_error =  0.0022456
residual_error =  0.022535

MFocuss
xMFOCUSS_transposed =
1.00048    0.00000    1.00146    0.00000    1.00190

solution_error =  0.0024394
residual_error =  0.027004
```

```
Signal no. 3
0   1   0   0   0

Focuss
xFOCUSS_transposed =
0.00000   0.99784   0.00000   0.00000   0.00000

solution_error =  0.0021606
residual_error =  0.016736

MFocuss
xMFOCUSS_transposed =
0.00000   0.99833   0.00000   0.00000   0.00000

solution_error =  0.0016659
residual_error =  0.012904


Signal no. 4
0   0   1   0   0

Focuss
xFOCUSS_transposed =
0.00000   0.00000   1.00224   0.00000   0.00000

solution_error =  0.0022401
residual_error =  0.019007

MFocuss
xMFOCUSS_transposed =
0.00000   0.00000   0.99708   0.00000   0.00000

solution_error =  0.0029241
residual_error =  0.024811


Signal no. 5
1   1   0   0   1

Focuss
xFOCUSS_transposed =
1.00025   0.99820   0.00000   0.00000   0.99932

solution_error =  0.0019382
residual_error =  0.017824

MFocuss
xMFOCUSS_transposed =
1.00014   1.00462   0.00000   0.00000   1.00146

solution_error =  0.0048473
residual_error =  0.046094
```

The signals were estimated using regularization matrix $L = I(identity)$.
Comparing both methods for each single case, there is no any spectacular difference between obtained solutions. Both of the errors are quite small in their value and solution is fairly estimated.

Problem 4: Solve the problem:

$$\min_{\mathbf{x}} \|\mathbf{x}\|_p \qquad \text{s.t.} \qquad \mathbf{Ax} = \mathbf{b},$$

where $\mathbf{A} = \begin{bmatrix} 2 & 3 & -1 & 10 & 21 & 44 & -9 & 1 & -1 \\ 1 & 2 & 2 & 8 & 15 & 35 & 8 & -3 & 1 \\ 3 & 1 & 1 & 6 & 16 & 53 & -7 & 2 & 2 \end{bmatrix}$ and $\mathbf{b} = [118 \quad 77 \quad 129]^T$, and $p \in [0,1]$.

Compare two cases, when p = 0 and p = 1, with respect to the residual error.

The tasks goal is to compare cases with parameter p = 0 and p = 1, which clearly indicates using of Focuss-family algorithms.
For the calculation below the parameter $\lambda = 1e - 8$ was fixed and used during the computations.

```
A =

2     3    −1    10    21    44    −9     1    −1
1     2     2     8    15    35     8    −3     1
3     1     1     6    16    53    −7     2     2

b =

118
77
129

x_exact =
0.0191812    0.0174549    0.0045213    0.0704225    0.1561079    0.4034636   −0.0369650
    0.0039733    0.0059462

x_focuss0 =
0.00000    0.00000    0.00000    0.00000    1.00000    2.00000   −1.00000    0.00000    0.00000

x_focuss1 =
2.6276e−140     8.8686e−69    −6.9865e−63     1.9074e−30     1.0000e+00     2.0000e+00
    −1.0000e+00     3.0741e−87    −7.6512e−62

x_mfocuss0 =
0.00000    0.00000    0.00000    0.00000    1.00000    1.00000   −1.00000    0.00000    0.00000

x_mfocuss1 =
0.00000    0.00000    0.00000    0.00000    1.00000    1.00000   −1.00000    0.00000    0.00000



Solution Errors:

e_focuss0 =     1.5787e−09
e_focuss1 =     1.2990e−09
e_mfocuss0 =   77.266
e_mfocuss1 =   77.266
```

While the solutions itself may not be very meaningful, the errors values usually are.
In this particular case, the computations vere conducted a few times, mostly because the MFOCUSS algorithm result were very similar in both cases of parameter p value.

To sum up: the results again were not exact, but very well approximated. The residual erors for FOCUSS algorithm were small. In case of MFOCUSS quite larger (in case of any mistake in Octave code or algorithm's coded, there were multiple checks, but nothing incorrect was found).

Problem 5: Solve the problem:

$$\min_{\mathbf{x}} \|\mathbf{b}-\mathbf{A}\mathbf{x}\|_2 \qquad \text{s.t.} \qquad \mathbf{B}\mathbf{x}=\mathbf{d},$$

where

$$\mathbf{A}=\begin{bmatrix}1 & 1 & 1 & 1\\ 1 & 3 & 1 & 1\\ 1 & -1 & 3 & 1\\ 1 & 1 & 1 & 3\\ 1 & 1 & 1 & -1\end{bmatrix}, \qquad \mathbf{b}=\begin{bmatrix}2\\ 1\\ 6\\ 3\\ 1\end{bmatrix}, \qquad \mathbf{B}=\begin{bmatrix}1 & 1 & 1 & -1\\ 1 & -1 & 1 & 1\\ 1 & 1 & -1 & 1\end{bmatrix} \qquad \text{and} \qquad \mathbf{d}=\begin{bmatrix}1\\ 3\\ -1\end{bmatrix}$$

To solve above-mentioned problem, the LS algorithm with equality constraints was chosen to be compared with built-in Octave function **lsqlin**.

```
LS algorithm with equality constraints:

x =
0.75000
-0.75000
1.25000
0.25000

Elapsed time is 0.017941 seconds.
e_Ax =     1.7092e-15
e_Bd =     6.6613e-16



Lsqlin algoritm:

x_octave =
0.50000
-0.50000
1.50000
0.50000

Elapsed time is 0.00419497 seconds.
e_OctaveAx =     1.7092e-15
e_OctaveBd =     6.6613e-16
```

What can be noticed is that in both cases the residual errors (because these were calculated) are the same.
The execution time seems to be better in case of built in function. And in fact there is not much more to comment.

Problem 6: Solve the NNLS problem:

$$\min_{\mathbf{x}} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|_2 \qquad \text{s.t. } \mathbf{x} \geq \mathbf{0},$$

where

$$\mathbf{A} = \begin{bmatrix} 73 & 71 & 52 \\ 87 & 74 & 46 \\ 72 & 2 & 7 \\ 80 & 89 & 71 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 49 \\ 67 \\ 68 \\ 20 \end{bmatrix}.$$

Which methods gives the best NNLS solution?

In task 6, the NNLS algorithm (interior point version) was coded and used. For the task, the hardcoded limit of 1000 iterations was set.
To have a comparison, also the built-in **lsqnonneg** function was used for computations.

```
Elapsed time is 0.00615311 seconds.
x_nnls =
3.3062e-01
4.1365e-05
7.2307e-01

error_nnls =  71.122



Elapsed time is 0.00321317 seconds.
x_octave =
0.64954
0.00000
0.00000

error_octave =  39.813
```

The performance in case of error value was much better in Octave's lsqnonneg function. The same is for the computation time.

Problem 7: For the matrix $\mathbf{A} = \begin{bmatrix} -4 & -2 & -4 & -2 \\ 2 & -2 & 2 & 1 \\ -4 & 1 & -4 & -2 \end{bmatrix}$ and $\mathbf{b} = \begin{bmatrix} -12 & 3 & -9 \end{bmatrix}^T$,

find the NNLS solution to $\mathbf{Ax} = \mathbf{b}$ and compare it with the ordinary LS solution. Compute the residual errors for each solution. Then, perturb the vector $\mathbf{b}$ with an additive zero-mean Gaussian noise with SNR = 20 dB, and compute the same solutions. How does a noise level affect the NNLS and LS solutions?

```
x_nnls =
-0.053757
1.041759
1.702508
1.616885

e_nnls =  0.28882


General Crossvalidation cannot be used due to "machine's precision" error.

x_QR =
2.2204e-16
1.0000e+00
2.0000e+00
1.0000e+00

Elapsed time is 0.000808954 seconds.
e_qr =    2.5121e-15

x_SVD =
-8.8818e-16
1.0000e+00
2.0000e+00
1.0000e+00

Elapsed time is 0.000778913 seconds.
e_svd =    8.9811e-15


b perturbed by noise

x_nnls =
5.4629e-01
7.1885e-01
8.6128e-05
4.4108e+00

e_nnls =  3.1961

x_QR =
0.023284
1.004230
2.006133
1.003067

Elapsed time is 0.000202894 seconds.
e_qr =    2.6645e-15

x_SVD =
0.023284
1.004230
2.006133
1.003067

Elapsed time is 0.000172138 seconds.
e_svd =    1.0986e-14
```

According to the result – the noise introduces a change in solution estimation, however, that change is larger in case of computing NNLS solution. The classical LS (using svd, qr) were not affected that much.

Problem 8: Applying the Galerkin discretization to the Fredholm integral equation of the first kind, we get the matrix operator:

$$a_{ii} = h^2\left(h\left(i^2 - i + \frac{1}{4}\right) - \left(i - \frac{2}{3}\right)\right),$$

$$a_{ij} = h^2\left(j - \frac{1}{2}\right)\left(h\left(i - \frac{1}{2}\right) - 1\right),$$

where $h = \frac{1}{n}$ and $i = 1, \ldots, n$. Let $b_i = \sum_{j=1}^{n} a_{ij} x_j^*$ where $x_j^* = \begin{cases} |s_j| & \text{if } s_j > 0 \\ 0 & \text{otherwise} \end{cases}$, $s_j = \sin(4\pi h j)$ for $j = 1, \ldots, n$. For $n = 10, 10^2, 10^3, 10^4 \ldots$

(a) estimate the minimal norm LS solution with the selected regularization methods and the NNLS solution,

(b) estimate the solution error $\|\mathbf{x} - \mathbf{x}_*\|_2$ and the residual error $\|\mathbf{b} - \mathbf{A}\mathbf{x}\|_2$ for each solution.

Perturb the vector **b** with an additive zero-mean Gaussian noise $N(0, \sigma^2)$ with the standard deviation adapted to have SNR = 0, 10, 20, 30 [dB]. Repeat the tasks (a) – (b) for the noisy data. Use the generalized Tikhonov regularization:

$$\min_{\mathbf{x}} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|_2 \qquad \text{s.t.} \qquad \|\mathbf{L}\mathbf{x}\|_2 \le \gamma \text{ and } 0 \le \mathbf{x} \le 1,$$

and find the right value of γ for each noisy case. Try the following cases of the matrix **L**:

- $\mathbf{L} = \mathbf{I}_n$ (identity),
- discrete approximation to the first derivative operator,
- discrete approximation to the second derivative operator.

To filfill Task8, the function to initialize the testing environment (meaningful name of the function – **galerkin_init**) was prepared.
All the calculations were dune using regularization matrix
$L = I(identity)$, as well as the first derivative.

Due to the problem size and solution space (the following powers of 10), only the numerical results of solution and residual errors will be shown.

L = Identity Matrix	L = First Derivative
N = 10	**N = 10**
b exact	**b exact**
e_solution_nnls = 7.5293	e_solution_nnls = 7.5293
e_residual_nnls = 1.1155	e_residual_nnls = 1.1155
e_solution_tikhonov = 7.5293	e_solution_tikhonov = 7.5293
e_residual_tikhonov = 0.028983	e_residual_tikhonov = 2.8637e-04
b perturbed by noise 10dB	**b perturbed by noise 10dB**
e_solution_nnls = 14.409	e_solution_nnls = 122.96
e_residual_nnls = 2.4311	e_residual_nnls = 18.594
e_solution_tikhonov = 14.409	e_solution_tikhonov = 122.96
e_residual_tikhonov = 1.3001	e_residual_tikhonov = 0.037848
b perturbed by noise 20dB	**b perturbed by noise 20dB**
e_solution_nnls = 11.839	e_solution_nnls = 10.329
e_residual_nnls = 1.7614	e_residual_nnls = 1.5201
e_solution_tikhonov = 11.839	e_solution_tikhonov = 10.329
e_residual_tikhonov = 0.38907	e_residual_tikhonov = 0.014126
b perturbed by noise 30dB	**b perturbed by noise 30dB**
e_solution_nnls = 12.018	e_solution_nnls = 7.3704
e_residual_nnls = 1.7388	e_residual_nnls = 1.0972
e_solution_tikhonov = 12.018	e_solution_tikhonov = 7.3704
e_residual_tikhonov = 0.094894	e_residual_tikhonov = 0.0041121

Figure 1.1 Computations for n = 10

By looking at the results, the interesting facts can be noticed.
1. In case of using NNLS algorithm, there is no difference between using Identity Matrix or First Derivative operator as a regularization matrix. The Solution and Residual errors are quite similar.
2. The very interesting case is for N = 10 and L = First derivative.
The error calculated from both NNLS and Tikhonov solutions suddenly increased.
3. The Tikhonov regularization performed better in measure of residual error of the

L = Identity Matrix	L = First Derivative
N = 100	**N = 100**
b exact	**b exact**
e_solution_nnls = 4.5857	e_solution_nnls = 4.5857
e_residual_nnls = 0.36383	e_residual_nnls = 0.36383
e_solution_tikhonov = 4.5857	e_solution_tikhonov = 4.5857
e_residual_tikhonov = 0.10359	e_residual_tikhonov = 3.2663e-05
b perturbed by noise 10dB	**b perturbed by noise 10dB**
e_solution_nnls = 4.6550	e_solution_nnls = 4.7040
e_residual_nnls = 3.1954	e_residual_nnls = 2.9751
e_solution_tikhonov = 4.6550	e_solution_tikhonov = 4.7040
e_residual_tikhonov = 3.1990	e_residual_tikhonov = 0.12531
b perturbed by noise 20dB	**b perturbed by noise 20dB**
e_solution_nnls = 4.5767	e_solution_nnls = 4.5880
e_residual_nnls = 1.0546	e_residual_nnls = 0.99470
e_solution_tikhonov = 4.5767	e_solution_tikhonov = 4.5880
e_residual_tikhonov = 0.97328	e_residual_tikhonov = 0.042137
b perturbed by noise 30dB	**b perturbed by noise 30dB**
e_solution_nnls = 4.5774	e_solution_nnls = 4.5807
e_residual_nnls = 0.50121	e_residual_nnls = 0.50303
e_solution_tikhonov = 4.5774	e_solution_tikhonov = 4.5807
e_residual_tikhonov = 0.29993	e_residual_tikhonov = 0.013966

Figure 1.2 Computations for n = 100

L = Identity Matrix	L = First Derivative
N = 1000	N = 1000
b exact	**b exact**
e_solution_nnls = 13.692	e_solution_nnls = 13.692
e_residual_nnls = 0.89060	e_residual_nnls = 0.89060
e_solution_tikhonov = 13.692	e_solution_tikhonov = 13.692
e_residual_tikhonov = 0.33199	e_residual_tikhonov = 3.2894e-05
b perturbed by noise 10dB	**b perturbed by noise 10dB**
e_solution_nnls = 13.693	e_solution_nnls = 13.692
e_residual_nnls = 9.5561	e_residual_nnls = 10.220
e_solution_tikhonov = 13.693	e_solution_tikhonov = 13.692
e_residual_tikhonov = 9.5277	e_residual_tikhonov = 0.46320
b perturbed by noise 20dB	**b perturbed by noise 20dB**
e_solution_nnls = 13.692	e_solution_nnls = 13.692
e_residual_nnls = 3.3141	e_residual_nnls = 3.2708
e_solution_tikhonov = 13.692	e_solution_tikhonov = 13.692
e_residual_tikhonov = 3.2104	e_residual_tikhonov = 0.14556
b perturbed by noise 30dB	**b perturbed by noise 30dB**
e_solution_nnls = 13.692	e_solution_nnls = 13.692
e_residual_nnls = 1.3675	e_residual_nnls = 1.3483
e_solution_tikhonov = 13.693	e_solution_tikhonov = 13.692
e_residual_tikhonov = 1.0343	e_residual_tikhonov = 0.045857

Figure 1.3 Computations for n = 1000

solution. It had even better result, when the regularization was First Derivative operator.

Chapter 2

Algorithms code

Algorithm 1 – FOCUSS algorithm.

```
function [x] = focuss(A,b,p,lambda, regul)

[m,n] = size(A);

x = rand(n,1);

#when reg = I, then we have generalized focuss
if(regul == 1)
regul = eye(m);
endif

for k = 1:100
W=diag(abs(x)).^(1-p/2);
W2 = W.*W;
x=W2*A'*inv(A*W2*A'+lambda*regul)*b;
endfor

endfunction
```

Algorithm 2 – MFOCUSS

```
function [x] = mfocuss(A,b,p,lambda)

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

x = ones(n,1);
I = eye(m);

for k = 1:100
% norm of each row
w = sqrt(sum(abs(x).^2,2));
W=diag(w.^(1-p/2));

A = A * W;
Q = A'*inv(A*A'+lambda*I)*b;

x=W*Q;

endfor
endfunction
```

Algorithm 3 – LS by QR for underdetermined problems.

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

[m,n] = size(A);
[Q, R] = qr(A');

z = inv(R(1:m,1:m)')*b;
x = Q(:,1:m)*z;

endfunction
```

Algorithm 4 – The LS solution by SVD for underdetermined problems.

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

[m,n] = size(A);

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

x = (s*v) \ (u'*b);
endfunction
```

Algorithm 5 – The LS by equality constraints (nullspace)

```
function [x] = equalityLS(A,b,C,d)

#according to
# https://stanford.edu/class/ee103/lectures/constrained-least-squares/constrained-least-squares_slides.pdf
# http://www.seas.ucla.edu/~vandenbe/133A/lectures/cls.pdf

x_exact = pseudoinverse(C)*d;

C_z = C';
d_z = A'*b - A'*A;;

z = pseudoinverse(C_z)*d_z;

if(isempty(z) == false)
x = x_exact;
endif


endfunction
```

Algorithm 6 – The interior-point NNLS

```
function [x] = nnls(A,b, iterations)

[m,n] = size(A);

AtA = A'*A;
Atb = A'*b;

e = ones(n,1);

xk = ones(n,1);
yk = xk;

completed_iterations = 0;
for i = 1:iterations

completed_iterations = i;

Xk= diag(xk);
Yk= diag(yk);

mi = (xk' * yk) ./ (n^2);

#we have diagonal matrices, so we can spare some time
Xk_inv = diag(1./xk);
Xk_sqrt_inv = diag(1./sqrt(xk));

Yk_sqrt = diag(sqrt(yk));
Yk_sqrt_inv = diag(1./sqrt(yk));

#taking advantage that sqrt(a*b) = sqrt(a)*sqrt(b)
Fac1 = [A; Xk_sqrt_inv * Yk_sqrt];
Fac2 = [b - A * Xk * e; Xk_sqrt_inv * Yk_sqrt_inv * mi * e ];

uk = Fac1\Fac2;

vk = -Yk*e + Xk_inv *mi*e - Xk_inv *Yk*uk;

T1 = min(-xk(uk < 0) ./ uk(uk < 0));
T2 = min(-yk(vk < 0) ./ vk(vk < 0));

#exact book condition
theta = 0.99995 * max(T1,T2);

#if no further improvemeent, then... break
if isempty(theta)
break;
endif

xk = xk + theta*uk;
yk = yk + theta*vk;
end

x = xk;
completed_iterations;
endfunction
```

Algorithm 7 – The General Crossvalidation

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

C = inv(A'*A);

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

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

endfunction
```

Algorithm 8 – The Tikhonov regularization

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

[m,n] = size(A);

if(L == 1)
    L = eye(n);
endif

x = inv(A' * A + alpha.*2*L'*L) * A' * b;

endfunction
```

Algorithm 9 – 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
```

Bibliography

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

[2] Golub, G. H., & Van Loan, C. F. (2012). Matrix computations (Vol. 3). JHU Press.

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