Data Smoothing

Exercise 3

Data Smoothing Report

Professor
Annie Cuyt

Author
Cesare De Cal

Assistant Professor
Ferre Knaepkens

November 7, 2019

1 Introduction

This exercise asks to use the linearly independent basis function:

$$\phi_{3,i}(x) = \binom{3}{i} x^i (1-x)^{3-i} \qquad i = 0, 1, 2, 3$$

to find the optimal combination

$$\phi(x) = \lambda_0 \phi_{3,0}(x) + \lambda_1 \phi_{3,1}(x) + \lambda_2 \phi_{3,2}(x) + \lambda_3 \phi_{3,3}(x)$$

that minimizes the Euclidean norm

$$\sqrt{\sum_{j=0}^{19} (\phi(x_j) - y_j)^2}$$

for the 20 data points (x_j, y_j) given in

j	x_j	y_j
0	0.0	-0.80
1	0.6	-0.34
2	1.5	0.59
3	1.7	0.59
4	1.9	0.23
5	2.1	0.10
6	2.3	0.28
7	2.6	1.03
8	2.8	1.50
9	3.0	1.44
10	3.6	0.74
11	4.7	-0.82
12	5.2	-1.27
13	5.7	-0.92
14	5.8	-0.92
15	6.0	-1.04
16	6.4	-0.79
17	6.9	-0.06
18	7.6	1.00
19	8.0	0.00

This is an overdetermined system. To solve this exercise, I'll first find the coefficient matrix A and then use QR decomposition to solve the system. I'll also calculate the l_2-norm of the residual, and the condition number of the coefficient matrix. Finally, I'll plot the data points with the graph of the function $\phi(x)$.

2 Tools

The following programming language and libraries have been used in this exercise:

- C
- GSL (GNU Scientific Library)
- C Math Library
- Python (for plotting)
- SciPy (for plotting)

The following double-precision GSL data types have been used in the exercise:

- `gsl_matrix`
- `gsl_vector`

The following GSL methods have been used in the exercise:

- `gsl_matrix_alloc(size1, size2)`
- `gsl_matrix_set(matrix, row, column, value)`
- `gsl_matrix_get(matrix, row, column)`
- `gsl_vector_alloc(size)`
- `gsl_vector_set(vector, index, value)`
- `gsl_vector_set_zero(vector)`
- `gsl_vector_get(vector, index)`
- `gsl_matrix_memcpy(destinationMatrix, matrixToCopyFrom)`
- `gsl_linalg_SV_decomp(A, V, S, workspaceVector)`
- `gsl_vector_minmax(vector, minInVector, maxInVector)`
- `gsl_sf_fact(number)` to calculate the factorial
- `gsl_sf_pow_int(base, exponent)`
- `gsl_blas_dnrm2(vector)` to calculate the Euclidean norm
- `gsl_matrix_free(matrixToDeallocate)`
- `gsl_vector_free(vectorToDeallocate)`

In order to decompose the coefficient matrix into its QR decomposition, and then solve the square system $A\vec{x} = \vec{b}$, I've used the following methods:

- `gsl_linalg_QR_decomp(matrixQR, vectorTau)`
- `gsl_linalg_QR_lssolve(matrixQR, vectorTau, vectorB, vectorX, vectorResidual)`

- `gsl_permutation_alloc(size)`

The following method from the C Math library was used in this exercise to calculate the absolute value of a number:

- `fabs(x)`

3 Computation

First of all, I compute the coefficients A of the linear system by using the linearly independent basis function. For the (i, j) cell in the matrix, I call the basis function by passing the i-th element of the x_j data points, and by using the column j for the i parameter of the basis function. This results in a 20×4 matrix. The following is a representation of A:

$$\begin{bmatrix}
1.000000000000000\,e+00 & 0.000000000000000\,e+00 & 0.000000000000000\,e+00 & 0.000000000000000\,e+00 \\
6.400000000000002\,e-02 & 2.880000000000000\,e-01 & 4.320000000000001\,e-01 & 2.160000000000000\,e-01 \\
-1.250000000000000\,e-01 & 1.125000000000000\,e+00 & -3.375000000000000\,e+00 & 3.375000000000000\,e+00 \\
-3.429999999999999\,e-01 & 2.499000000000000\,e+00 & -6.068999999999998\,e+00 & 4.912999999999999\,e+00 \\
-7.289999999999998\,e-01 & 4.616999999999998\,e+00 & -9.747000000000000\,e+00 & 6.858999999999999\,e+00 \\
-1.331000000000000\,e+00 & 7.623000000000002\,e+00 & -1.455300000000000\,e+01 & 9.261000000000001\,e+00 \\
-2.196999999999999\,e+00 & 1.166100000000000\,e+01 & -2.063099999999999\,e+01 & 1.216700000000000\,e+01 \\
-4.096000000000001\,e+00 & 1.996800000000001\,e+01 & -3.244800000000000\,e+01 & 1.757600000000000\,e+01 \\
-5.831999999999998\,e+00 & 2.721599999999999\,e+01 & -4.233599999999999\,e+01 & 2.195199999999999\,e+01 \\
-8.000000000000000\,e+00 & 3.600000000000000\,e+01 & -5.400000000000000\,e+01 & 2.700000000000000\,e+01 \\
-1.757600000000000\,e+01 & 7.300800000000001\,e+01 & -1.010880000000000\,e+02 & 4.665600000000001\,e+01 \\
-5.065300000000001\,e+01 & 1.930290000000000\,e+02 & -2.451990000000000\,e+02 & 1.038230000000000\,e+02 \\
-7.408800000000001\,e+01 & 2.751840000000000\,e+02 & -3.407040000000000\,e+02 & 1.406080000000000\,e+02 \\
-1.038230000000000\,e+02 & 3.777390000000001\,e+02 & -4.581090000000000\,e+02 & 1.851930000000000\,e+02 \\
-1.105920000000000\,e+02 & 4.008960000000000\,e+02 & -4.844160000000000\,e+02 & 1.951120000000000\,e+02 \\
-1.250000000000000\,e+02 & 4.500000000000000\,e+02 & -5.400000000000000\,e+02 & 2.160000000000000\,e+02 \\
-1.574640000000000\,e+02 & 5.598720000000002\,e+02 & -6.635520000000001\,e+02 & 2.621440000000001\,e+02 \\
-2.053790000000000\,e+02 & 7.205670000000001\,e+02 & -8.426970000000001\,e+02 & 3.285090000000001\,e+02 \\
-2.874960000000000\,e+02 & 9.931679999999998\,e+02 & -1.143648000000000\,e+03 & 4.389759999999999\,e+02 \\
-3.430000000000000\,e+02 & 1.176000000000000\,e+03 & -1.344000000000000\,e+03 & 5.120000000000000\,e+02
\end{bmatrix}$$

Then, I calculate the condition number of the coefficient matrix A. In GSL there is no direct function that calculates the condition number, but it's possible to use the ratio of the largest singular value of matrix A, $\sigma_n(A)$, to the smallest $\sigma_1(A)$:

$$\kappa(A) := \frac{\sigma_n(A)}{\sigma_1(A)} = \frac{\|A\|}{\|A^{-1}\|^{-1}}$$

I proceed to factorize A into its singular value decomposition SVD using the `gsl_linalg_SV_decomp` method, and then use `gsl_vector_minmax` to extract the minimum and maximum singular values out of the vector S that contains the diagonal elements of the singular value matrix. The condition number of the matrix A is equal to $3.741019262503867e+03$.

The column vector $\vec{b}$ is formed by the input y_j values:

$$\begin{bmatrix} -8.000000000000000e-01 \\ -3.400000000000000e-01 \\ 5.900000000000000e-01 \\ 5.900000000000000e-01 \\ 2.300000000000000e-01 \\ 1.000000000000000e-01 \\ 2.800000000000000e-01 \\ 1.030000000000000e+00 \\ 1.500000000000000e+00 \\ 1.440000000000000e+00 \\ 7.400000000000000e-01 \\ -8.200000000000000e-01 \\ -1.270000000000000e+00 \\ -9.200000000000000e-01 \\ -9.200000000000000e-01 \\ -1.040000000000000e+00 \\ -7.900000000000000e-01 \\ -6.000000000000000e-02 \\ 1.000000000000000e+00 \\ 0.000000000000000e+00 \end{bmatrix}$$

I proceed in solving the system (finding the λ values that approximate to a solution of the system). I use QR decomposition to solve the system (`gsl_linalg_QR_decomp` and `gsl_linalg_QR_lssolve` GSL methods). The following is the QR decomposition of A:

$$\begin{bmatrix}
-5.607137694902810e+02 & 1.956202843752372e+03 & -2.276212763756861e+03 & 8.833928944999552e+02 \\
1.139370324819986e-04 & -4.671135706796601e+01 & 1.122030535262658e+02 & -6.782418228480103e+01 \\
-2.225332665664034e-04 & 1.460497799957120e-02 & 7.581872706318327e+00 & -8.735870723873337e+00 \\
-6.106312834582108e-04 & 2.762429988027977e-02 & 1.797466673315370e-01 & -2.470454024049311e+00 \\
-1.297814010615264e-03 & 4.400918925292650e-02 & 2.113118885654055e-01 & 1.354991299702775e-01 \\
-2.369534222399064e-03 & 6.326913656703972e-02 & 2.350053228690535e-01 & 9.686987975376644e-02 \\
-3.911244693171105e-03 & 8.491363227214763e-02 & 2.513310790520081e-01 & 6.050043190845661e-02 \\
-7.291970078847909e-03 & 1.207784144212678e-01 & 2.631080548709738e-01 & 1.049772432830278e-02 \\
-1.038251208492211e-02 & 1.462383018205487e-01 & 2.632200917938682e-01 & -1.959354810540480e-02 \\
-1.424212906024982e-02 & 1.723659541841733e-01 & 2.577288314294048e-01 & -4.692208258102275e-02 \\
-3.128995754536886e-02 & 2.498504055563917e-01 & 2.126763547071380e-01 & -1.108942994397425e-01 \\
-9.017582041110428e-02 & 3.520968788593226e-01 & 5.605686650054093e-02 & -1.473630562460556e-01 \\
-1.318963572269736e-01 & 3.635568863958127e-01 & -2.831753402090372e-02 & -1.240982343374392e-01 \\
-1.848325706777897e-01 & 3.424022086740725e-01 & -1.099043818739413e-01 & -7.278910904340619e-02 \\
-1.968831921288935e-01 & 3.335830602668363e-01 & -1.251940955112117e-01 & -5.896438574071358e-02 \\
-2.225332665664034e-01 & 3.108660410390103e-01 & -1.542417580429807e-01 & -2.760834500928011e-02 \\
-2.803278262928973e-01 & 2.434003295032724e-01 & -2.044456433018340e-01 & 5.043304854412397e-02 \\
-3.656292780331310e-01 & 1.127004233192019e-01 & -2.472054281433762e-01 & 1.782091144383195e-01 \\
-5.118193920381977e-01 & -1.705668773785795e-01 & -2.556299463550832e-01 & 4.175378294298211e-01 \\
-6.106312834582110e-01 & -3.921595584433181e-01 & -2.261060282111244e-01 & 5.880648466528077e-01
\end{bmatrix}$$

The solutions vector is directly given by the `gsl_linalg_QR_lssolve` method.

$$\begin{bmatrix} -1.118262027055321e+00 \\ -4.515724133992401e-01 \\ 2.972563550105292e-03 \\ 2.972882574075207e-01 \end{bmatrix}$$

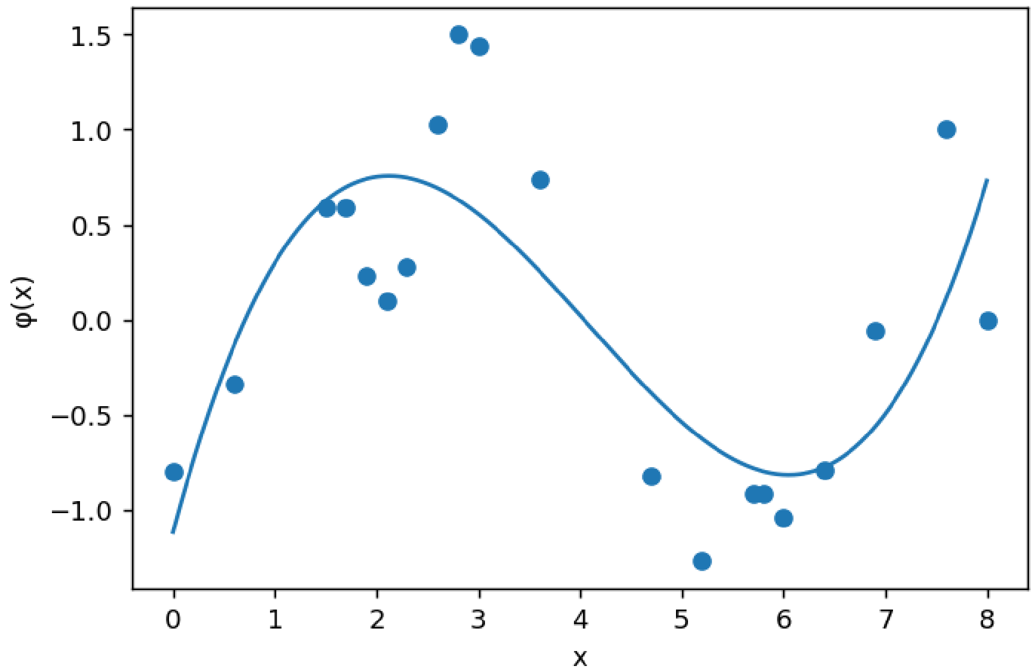
The residual vector $\vec{r}$ is also directly given by the `gsl_linalg_QR_lssolve` method.

$$\begin{bmatrix} 3.182620270553753e-01 \\ -2.038767862631474e-01 \\ -3.507925507654632e-02 \\ -1.076211346528342e-01 \\ -5.104297656943454e-01 \\ -6.559970851745927e-01 \\ -4.668150300670836e-01 \\ 3.379120178166642e-01 \\ 8.680652851344475e-01 \\ 8.842461476326978e-01 \\ 4.840349364751467e-01 \\ -4.332442092900336e-01 \\ -6.426370573997744e-01 \\ -1.387527086448619e-01 \\ -1.218089886087358e-01 \\ -2.242466352247385e-01 \\ -1.314405461740510e-02 \\ 5.037445845512266e-01 \\ 8.885672158267567e-01 \\ -7.311795037780225e-01 \end{bmatrix}$$

The Euclidean norm of the residual vector, which can be easily calculated with `gsl_blas_dnrm2(vector)`, is equal to $2.282876480420795e+00$.

4 Plot

I will plot the computed function together with the data points. In order to plot the graph of the $\phi(x)$ function, I used the λ values and plugged them into the linear combination. Plus, I created an array of equidistant x values and calculated the function value for each of the values.



5 Observations

We can observe that the graph isn't very accurate.