Basic Exercises for Scientific Programming

Hongzhou Ye

January 1, 2018

Abstract

Scientific programming involves many topics of applied math, varying from basic linear algebra to optimization algorithms. This file is a collection of several basic exercises that help you practice some of the essential skills.

1 Linear least square fitting

In the most general sense, the task of least square fitting is to find an approximate mapping $\tilde{f}$ to a given mapping

$$\begin{aligned} f : \mathbb{X} &\to \mathbb{Y} \\ \boldsymbol{x} &\to \boldsymbol{y} = f(\boldsymbol{x}) \end{aligned} \tag{1}$$

by minimizing the least square loss

$$\mathcal{L} = \sum_i \|\tilde{f}(\boldsymbol{x}^{(i)}) - \boldsymbol{y}^{(i)}\|_2^2 \tag{2}$$

based on a set of known data $\{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}$, where

$$\|\boldsymbol{a}\|_2 = \sqrt{\boldsymbol{a} \cdot \boldsymbol{a}} = \left(\sum_\mu a_\mu^2\right)^{1/2} \tag{3}$$

is the 2-norm of vector $\boldsymbol{a}$.

As a special case, linear least square fitting assumes a linear functional form for the approximate mapping $\tilde{f}$, i.e.

$$\tilde{f}(\boldsymbol{x}) = \mathbf{A}^{\mathrm{T}}\boldsymbol{x} + \boldsymbol{b}, \tag{4}$$

where $\mathbf{A}$ and $\boldsymbol{b}$ are determined from experimental data. In this exercise, we are going to explore the properties of linear least square fitting.

1. First let us consider an even simpler case, $\boldsymbol{b} = \mathbf{0}$. Show that $\mathbf{A}$ is determined by the following equation

$$\mathbf{X}\mathbf{X}^{\mathrm{T}}\mathbf{A} = \mathbf{X}\mathbf{Y}^{\mathrm{T}} \tag{5}$$

where

$$\mathbf{X} = [\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \cdots, \boldsymbol{x}^{(N)}], \quad \mathbf{Y} = [\boldsymbol{y}^{(1)}, \boldsymbol{y}^{(2)}, \cdots, \boldsymbol{y}^{(N)}]. \tag{6}$$

Hint: $\mathcal{L}$ is a function of $\mathbf{A}$

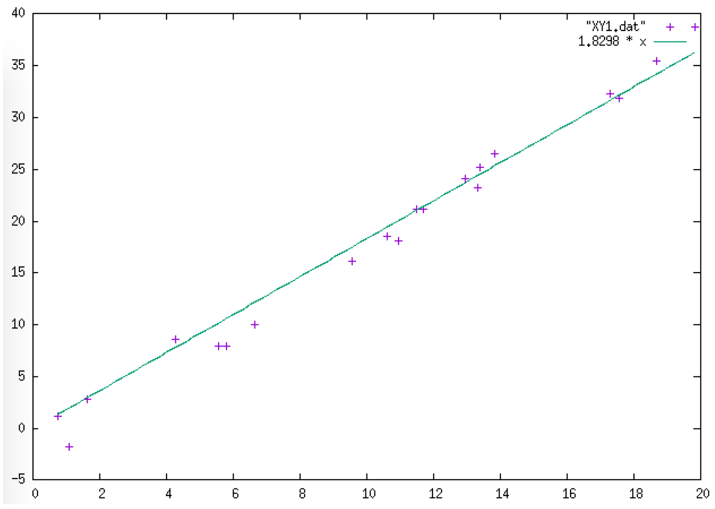
$$\mathcal{L} = \sum_i \|\mathbf{A}^{\mathrm{T}}\boldsymbol{x}^{(i)} - \boldsymbol{y}^{(i)}\|_2^2 = \sum_i (\mathbf{A}^{\mathrm{T}}\boldsymbol{x}^{(i)} - \boldsymbol{y}^{(i)})^{\mathrm{T}}(\mathbf{A}^{\mathrm{T}}\boldsymbol{x}^{(i)} - \boldsymbol{y}^{(i)}) \tag{7}$$

You might need the following trick

$$\begin{aligned} \sum_i \boldsymbol{x}^{(i)\mathrm{T}}\mathbf{A}\mathbf{A}^{\mathrm{T}}\boldsymbol{x}^{(i)} &= \sum_i \sum_{\mu\nu\lambda} x_\mu^{(i)} A_{\mu\nu} A^{\mathrm{T}}_{\nu\lambda} x_\lambda^{(i)} = \sum_{\mu\nu\lambda} A^{\mathrm{T}}_{\nu\lambda}\left[\sum_i x_\lambda^{(i)} x_\mu^{(i)}\right] A_{\mu\nu} \\ &= \sum_{\mu\nu\lambda} A^{\mathrm{T}}_{\nu\lambda}(XX^{\mathrm{T}})_{\lambda\mu} A_{\mu\nu} = \mathbf{A}^{\mathrm{T}}\mathbf{X}\mathbf{X}^{\mathrm{T}}\mathbf{A} \end{aligned} \tag{8}$$

Same trick can be played to all other terms in the expansion. Then you can take derivative with $\mathbf{A}^{\mathrm{T}}$ and obtain the desired equation (you can treat $\mathbf{A}$ and $\mathbf{A}^{\mathrm{T}}$ two independent variables).

2. We now have the equation and let us see how it works! In `data/least_square` you can find the data file `X1.txt` and `Y1.txt`, which contains 20 data points. Fit a linear equation with zero $\boldsymbol{b}$ and plot the fitted curve along with the original data. Your result should be something like the following



3. The example above is one-dimensional and trivial. A non-trivial multi-dimensional data set can be found in `X2.txt` and `Y2.txt`, where $\boldsymbol{x}^{(i)} \in \mathbb{R}^5$ and $\boldsymbol{y}^{(i)} \in \mathbb{R}^3$ and there are 20 of them. Find an $\mathbf{A} \in \mathbb{R}^{5\times 3}$ that minimizes the square loss. If you do the math correctly, the least square loss as defined in eqn (7) should be 3.431402129330746.

2 Non-linear fitting in terms of LLS

The reason why linear least square (LLS) is important is that one can go beyond linear fitting by introducing basis functions.

To be specific, let us first specify the dimension for each quantity:

$$\boldsymbol{x} \in \mathbb{R}^n, \quad \boldsymbol{y} \in \mathbb{R}^m, \quad \text{data set: } \{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_{i=1}^N. \tag{9}$$

Now we introduce a basis function:

$$\begin{aligned} \phi : \mathbb{R}^n &\to \mathbb{R}^l \\ \boldsymbol{x} &\to \boldsymbol{\phi} = \phi(\boldsymbol{x}) \end{aligned} \tag{10}$$

For example, $\phi(x) = [x_1^2, x_2^2, x_1 x_2]^{\mathrm{T}}$ is a basis function transforming a $\mathbb{R}^2$ vector $\boldsymbol{x} = [x_1, x_2]^{\mathrm{T}}$ to a $\mathbb{R}^3$ vector $\boldsymbol{\phi}$. With this in hands, our ansatz in eqn (4) needs to be modified

$$\tilde{f}(\boldsymbol{x}) = \mathbf{A}^{\mathrm{T}} \phi(\boldsymbol{x}) + \boldsymbol{b} \tag{11}$$

and $\mathbf{A} \in \mathbb{R}^{l\times m}$.

Though eqn (11) is still linear in $\boldsymbol{\phi}$, it does not need to be linear in $\boldsymbol{x}$ because the basis function could be non-linear. In this way, one can achieve non-linearity within the framework of LLS.

1. In a way similar to the derivation of eqn (5), show that $\mathbf{A}$ in eqn (11) is determined by solving

$$\mathbf{\Phi}\mathbf{\Phi}^{\mathrm{T}}\mathbf{A} = \mathbf{\Phi}\mathbf{Y}^{\mathrm{T}} \tag{12}$$

where $\mathbf{Y}$ is defined in eqn (6), and

$$\mathbf{\Phi} = [\boldsymbol{\phi}^{(1)}, \boldsymbol{\phi}^{(2)}, \cdots, \boldsymbol{\phi}^{(N)}]. \tag{13}$$

2. *Polynomial basis.* LLS with basis function

$$\phi(x) = [1, x, x^2, \cdots, x^{l-1}] \tag{14}$$

is equivalent to fitting a polynomial of order $l-1$. Perform LLS fit with $l = 2, 4, 6$ and 8 for data sets `X1.txt` and `Y1.txt` in `data/non_linear_least_square`. Then

(a) report the loss for each l

(b) report the RMS of $\mathbf{A}$ for each l

$$\mathrm{RMS}\,\mathbf{A} = \frac{1}{nm}\sum_{\mu}^{n}\sum_{\nu}^{m} A_{\mu\nu}^2 = \texttt{mean(sum(sum(A.*A)))} \qquad \text{(In MATLAB)} \tag{15}$$

(c) plot all fitted curves along with the data points

What trend(s) do you observe?

Hint: wrapping the fitting and plotting processes each in a function would make your life much easier. Specifically, they could be like

```
A = lls_poly(X, Y, l)
    % X/Y: data sets; l: polynomial order
    % A: LLS coefficient matrix.
```

and

```
y = lls_poly_plot(x, A, l)
    % x: grid points for plotting (e.g. x = [-1.5:0.1:2.5])
    % A: optimized LLS coefficient matrix
    % l: polynomial order
    % y: function values on grid x, so that you can plot with plot(x, y)
```

3. *Polynomial basis (cnt'd).* The data set above is generated as follows

$$y = 1 - 2x + x^3 + g \tag{16}$$

for x randomly drawn from -1.5 to 2, where $g \sim \mathcal{N}(0,1)$ is a Gaussian random variable that mimics a random noise (from, e.g. measure error). Now plot your fitted curves and the original data points along with the "real" solution (set $g = 0$).

(a) Does it approach the "real" solution for larger l?

(b) Does a small loss always mean a *good* fitting?

4. *Polynomial basis (cnt'd).* In the example above, we see how increasing the *model complexity* (in this case, l) can result in *overfitting.* However, by using more data points, we can have a more complex model without overfitting. Data sets `X2.txt` and `Y2.txt` are generated in the same way, but of a much larger size! Now re-do all fittings with the new data and plot the fitted curves along with the "real" solution. What new trend(s) do you observe?

5. *Sinusoidal basis.* Copy everything from `lls_poly` to a new function named `lls_sin` that uses the following basis

$$[1, \sin x, \cos x, \cdots, \sin(l-1)x, \cos(l-1)x]^{\mathrm{T}} \tag{17}$$

for an order-l fitting. Then re-do the fitting for $l = 2, 4, 6$ and 8. Report variation of the loss as the model complexity increases. Plot all fitted curves along with the "real" solution. Do you notice any artificial feature for large l?