ECE-GY 9243/ME-GY 7973 Spring 2019

Optimal and Learning Control for Robotics

Homework 1

Prof. Ludovic Righetti *Name: Yunian Pan*

1 exercise 1 [Convex optimization with linear equalities]

The optimization problem is:

$$\begin{aligned} \min_{\mathbf{x}} \ & \frac{1}{2}\mathbf{x}^\top \mathbf{Q}\mathbf{x} \\ \text{subject to } & \mathbf{A}\mathbf{x} = \mathbf{b} \end{aligned} \tag{1}$$

The Lagrangian of the optimization problem is:

$$\mathcal{L} = \frac{1}{2}\mathbf{x}^\top \mathbf{Q}\mathbf{x} + \lambda^\top(\mathbf{A}\mathbf{x} - \mathbf{b}) \tag{2}$$

Use the KKT condition we get:

$$\frac{1}{2}(\mathbf{Q} + \mathbf{Q}^\top)\mathbf{x} + \mathbf{A}^\top\lambda = 0 \tag{3}$$

$$\mathbf{A}\mathbf{x} - \mathbf{b} = 0 \tag{4}$$

Solve the KKT system, it's known that $\mathbf{Q} \in R^{n\times n} > 0 \implies \mathbf{Q}$ is invertible, $\mathbf{A} \in R^{m\times n}$ is full rank $\implies \mathbf{A}(\mathbf{Q} + \mathbf{Q}^\top)^{-1}\mathbf{A}^\top$ is invertible, thus:

$$\lambda = -\frac{1}{2}(\mathbf{A}(\mathbf{Q} + \mathbf{Q}^\top)^{-1}\mathbf{A}^\top)^{-1}\mathbf{b} \tag{5}$$

$$\mathbf{x} = (\mathbf{Q} + \mathbf{Q}^\top)^{-1}\mathbf{A}^\top(\mathbf{A}(\mathbf{Q} + \mathbf{Q}^\top)^{-1}\mathbf{A}^\top)^{-1}\mathbf{b} \tag{6}$$

now our problem is:

$$\begin{aligned} \min_{\mathbf{x}} \ & \frac{1}{2}\mathbf{x}^\top \begin{bmatrix} 10 & 1 & 0 \\ 1 & 100 & 2 \\ 0 & 2 & 1 \end{bmatrix} \mathbf{x} \\ \text{subject to } & \mathbf{1}^\top\mathbf{x} = 1 \end{aligned} \tag{7}$$

where $\mathbf{A} = \mathbf{1}^\top = [1\ 1\ 1]$ and $\mathbf{Q} = \begin{bmatrix} 10 & 1 & 0 \\ 1 & 100 & 2 \\ 0 & 2 & 1 \end{bmatrix}$, apply formula 5 and 6, we get the result $\mathbf{x}$:

$$\mathbf{x}^\top = \begin{bmatrix} 0.09090909, & -0.01030928, & 0.91940019 \end{bmatrix}$$

2 exercise 2 [Netwon's method]

- As implemented in jupyter notebook, Newton's method converges much faster than gradient descent.
- As implemented in jupyter notebook, the results are as follows:
  - As shown in 2.1, $f(x)$ *with* $x \in [-10, 10]$, start from $x_0 = 6.5$ the iteration went all the way down to the convergence extremal point.

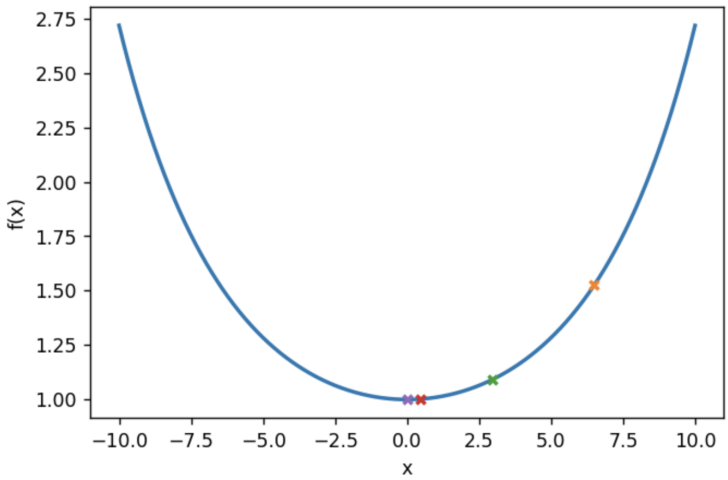

Figure 2.1

  - As shown in 2.2, $f(\mathbf{x})$ $\mathbf{x}$ in the box $\{(x, y)|x \in [-10, 10], y \in [-10, 10]\}$, the converging process goes in almost one direction.

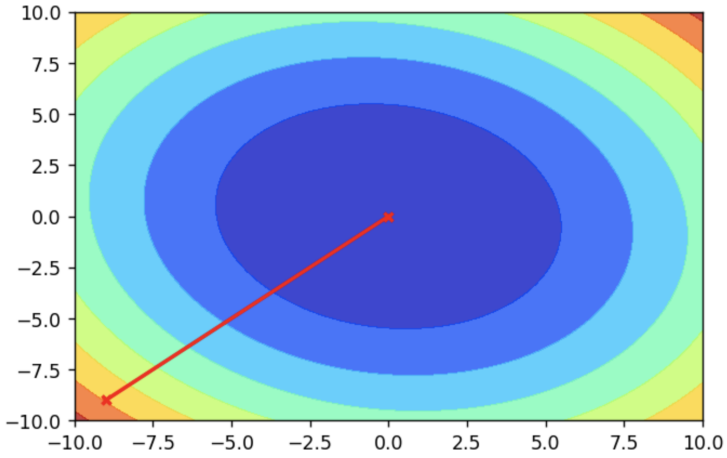

Figure 2.2

- comparison shown in 2.3 and 2.4, since the points of convergence can be easily computed that $x_c = 0$ and $\mathbf{x}_c = (0, 0)$, I compared the converging process with these extremal points instead

of the final results, as there's a tolerance below which the iteration stops, the true error will never goes to 0.

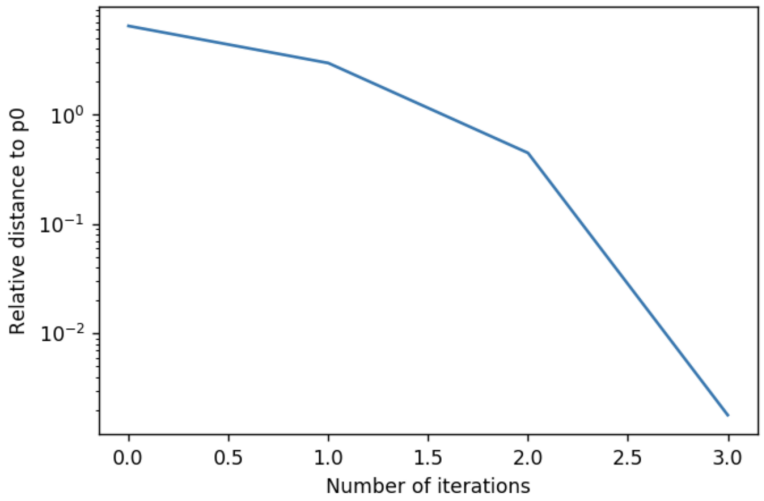

Figure 2.3: $f(x) = e^{\frac{x^2}{100}}$

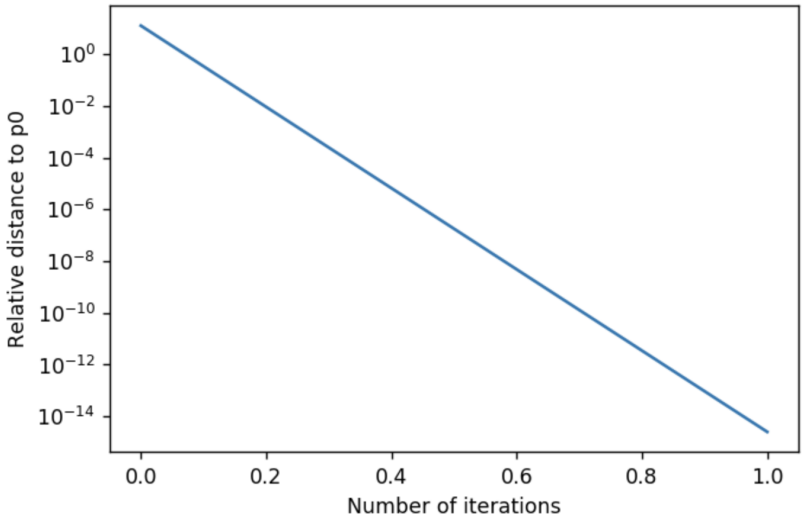

Figure 2.4: $f(x) = x^\top Qx$

Note that as tolerance goes down to 10^{-10} or even smaller, the number of iteration increases to some degree.

3 exercise 3 [Linear Least Squares]

- As implemented in jupyter notebook, I modified the function so that it can process cross-validation to find the best polynomial order without overfitting.

– The order of polynomial that best fits the data is uncertain because the split of training and testing set is random, it usually takes on a value between 3 and 6 as I tried from 1 to 10, 3.1 shows an example of best order.

```
iteration 1
## the model is array([ 1.31880712, 19.04432667])
** fitting error 80090.749143 ** testing error 12761.636030
iteration 2
## the model is array([1.17851011e+00, 1.90453381e+01, 1.84352655e-02])
** fitting error 80087.674410 ** testing error 12875.067926
iteration 3
## the model is array([0.69949605, 4.55815354, 0.08024354, 1.01915273])
** fitting error 18961.131545 ** testing error 2423.782280
iteration 4
## the model is array([ 1.22515415,  4.52150839, -0.14003611,  1.02175701,  0.01063706])
** fitting error 18917.936732 ** testing error 2364.004758
iteration 5
## the model is array([ 1.23057831,  6.32419967, -0.17486166,  0.67588389,  0.01307481,
       0.01271192])
** fitting error 18524.096760 ** testing error 2519.638213
iteration 6
## the model is array([ 1.54398423e+00,  6.30747424e+00, -4.47161123e-01,  6.79093822e-01,
       4.64510031e-02,  1.25605325e-02, -9.93328162e-04])
** fitting error 18508.282956 ** testing error 2539.797070
iteration 7
## the model is array([ 1.54420286e+00,  6.67332574e+00, -4.56852426e-01,  5.46292585e-01,
       4.77035258e-02,  2.43370629e-02, -1.03608488e-03, -2.93114500e-04])
** fitting error 18499.408077 ** testing error 2533.753299
iteration 8
## the model is array([-3.91837550e-01,  6.99255019e+00,  2.40805864e+00,  4.16870717e-01,
       -5.89128962e-01,  3.62675796e-02,  4.33289433e-02, -6.04405373e-04,
       -9.52120325e-04])
** fitting error 17915.642847 ** testing error 2765.818573
iteration 9
## the model is array([-3.92416210e-01,  6.94347996e+00,  2.41159239e+00,  4.46054108e-01,
       -5.90125976e-01,  3.16659697e-02,  4.34089920e-02, -3.39860519e-04,
       -9.54031168e-04, -5.00916148e-06])
** fitting error 17915.545418 ** testing error 2764.817989
iteration 10
## the model is array([-1.50691266e+00,  7.25022405e+00,  4.94617274e+00,  2.40505155e-01,
       -1.48507558e+00,  6.71818070e-02,  1.52396313e-01, -2.57872769e-03,
       -6.30459888e-03,  4.11651332e-05,  9.10053395e-05])
** fitting error 17732.800840 ** testing error 2898.482875
*****************
best_d = 4
*****************
the best model is array([ 1.22515415,  4.52150839, -0.14003611,  1.02175701,  0.01063706]) selected through cross-val
idation
```

Figure 3.1

The best order is 4 in this case because the model with $degree = 4$ has the lowest testing error, which means it predicts the data more reasonably while fits the dataset well.

– The polynomial coefficients are in the selected model, which is [1.22515415, 4.52150839, -0.14003611, 1.02175701, 0.01063706].

– Plot the overlaid function as shown in 3.2. The fit looks good, despite the fact that some of the data points are out of the curve, those data points can be viewed as noise.

- Since it is in fact possible to use functions to mapp the data as long as the mapping is of the form $y = \sum_k a_k f_k(x)$ where f_k is an arbitrary function. Now that we want to fit periodic functions of the form:

$$y_n = a_0 + \sum_{k=1}^{K} a_k \cos(kT2\pi x_n) + \sum_{k=1}^{K} b_k \sin(kT2\pi x_n)$$

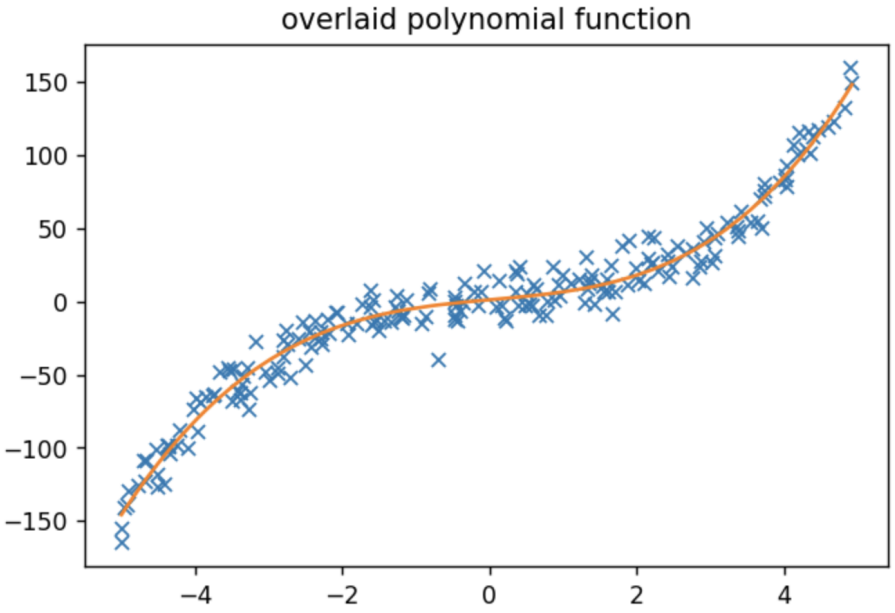

Figure 3.2

– The optimization problem can be written as:

$$\begin{aligned} &\min \ ||y - \hat{y}||^2 \\ &\text{subject to } \hat{y} = F(\mathbf{x})\mathbf{w} \end{aligned}$$

$$\text{where } F(\mathbf{x}) = \begin{bmatrix} 1 & \cos(2\pi T x_1) & \cdots & \cos(2\pi K T x_1) & \sin(2\pi T x_1) & \cdots & \sin(2\pi K T x_1) \\ 1 & \cos(2\pi T x_2) & \cdots & \cos(2\pi K T x_2) & \sin(2\pi T x_2) & \cdots & \sin(2\pi K T x_2) \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 1 & \cos(2\pi T x_n) & \cdots & cos(2\pi K T x_n) & \sin(2\pi T x_n) & \cdots & \sin(2\pi K T x_n) \end{bmatrix}$$

$$\mathbf{w}^\top = [a_0 \ a_1 \dots a_K \ b_1 \ \dots b_K]$$

– As implemented in jupyter notebook. We take derivative to obtain the parameters:

$$\begin{aligned} \mathcal{L} &= ||y - \hat{y}||^2 \\ \frac{\partial \mathcal{L}}{\partial \mathbf{w}} &= \frac{\partial}{\partial \mathbf{w}} (\mathbf{y} - F(\mathbf{x})\mathbf{w})^\top (\mathbf{y} - F(\mathbf{x})\mathbf{w}) \\ &= -2F(\mathbf{x})^\top \mathbf{y} + 2F(\mathbf{x})^\top F(\mathbf{x})\mathbf{w} \end{aligned}$$

Let $\dfrac{\partial \mathcal{L}}{\partial \mathbf{w}} = 0 \implies \mathbf{w} = (F(\mathbf{x})^\top F(\mathbf{x}))^{-1} F(\mathbf{x})^\top \mathbf{y}$, which is quite similar to the polynomial solution.

– Assuming $T = 1$, start from $K = 2$, the testing error and training error already decrease to 0, as shown in 3.3.

– 3.4 gives the overlaid function, which looks good.

```
iteration 1
## the model is array([ 1.97253699,  1.92582147, -0.01222802])
** fitting error 164.801043 ** testing error 73.044848
iteration 2
## the model is array([ 2.00000000e+00,  2.00000000e+00,  1.35207180e-16, -1.77216311e-16,
        1.50000000e+00])
** fitting error 0.000000 ** testing error 0.000000
iteration 3
## the model is array([ 2.00000000e+00,  2.00000000e+00,  1.09708947e-16,  4.04438030e-17,
       -1.60390644e-16,  1.50000000e+00,  2.14314281e-16])
** fitting error 0.000000 ** testing error 0.000000
iteration 4
## the model is array([ 2.00000000e+00,  2.00000000e+00,  3.10485408e-17,  1.64076633e-16,
       -2.15906830e-16, -1.41269215e-16,  1.50000000e+00,  3.44434019e-17,
       -1.47298646e-16])
** fitting error 0.000000 ** testing error 0.000000
iteration 5
## the model is array([ 2.00000000e+00,  2.00000000e+00,  3.07448698e-17,  1.75087526e-16,
       -2.22207407e-16,  2.20970940e-18, -1.25037990e-16,  1.50000000e+00,
        1.18151111e-17, -8.92544975e-17,  9.48674861e-17])
** fitting error 0.000000 ** testing error 0.000000
*****************
best_k = 2
*****************
the best model is array([ 2.00000000e+00,  2.00000000e+00,  1.35207180e-16, -1.77216311e-16,
        1.50000000e+00]) selected through cross-validation
```

Figure 3.3

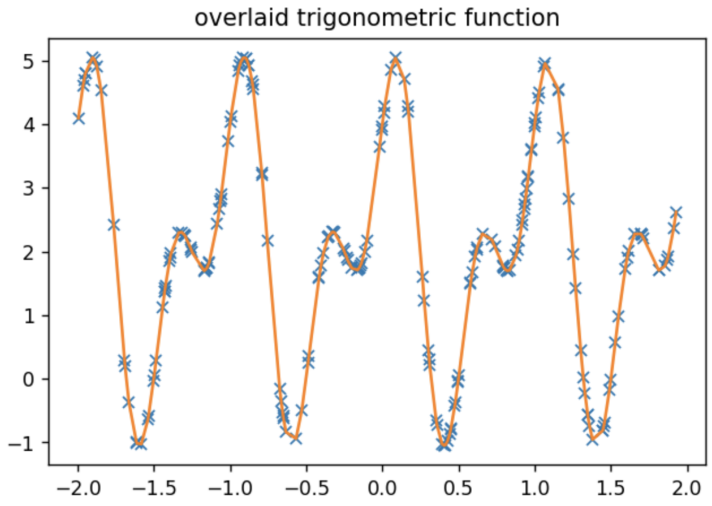

Figure 3.4