EE613 - List of courses

19.09.2019 (JMO) Introduction
26.09.2019 (JMO) Generative I
03.10.2019 (JMO) Generative II
10.10.2019 (JMO) Generative III
17.10.2019 (JMO) Generative IV
24.10.2019 (JMO) Decision-trees
31.10.2019 (SC) Linear regression I
07.11.2019 (JMO) Kernel SVM
14.11.2019 (SC) Linear regression II
21.11.2019 (FF) MLP
28.11.2019 (FF) Feature-selection and boosting
05.12.2019 (SC) HMM and subspace clustering
Outline

Linear Regression I  (Oct 31)
• Least squares
• Singular value decomposition (SVD)
• Kernels in least squares (nullspace)
• Ridge regression (Tikhonov regularization)
• Weighted least squares
• Iteratively reweighted least squares (IRLS)
• Recursive least squares

Linear Regression II  (Nov 14)
• Logistic regression
• Tensor-variate regression

Hidden Markov model (HMM) & subspace clustering  (Dec 5)

Nonlinear Regression I  (Dec 12)
• Locally weighted regression (LWR)
• Gaussian mixture regression (GMR)

Nonlinear Regression II  (Dec 19)
• Gaussian process regression (GPR)
Labs

Teguh Lembono

Python notebooks and labs exercises:
https://github.com/teguhSL/ee613-python
http://www.idiap.ch/software/pbdlib/
LEAST SQUARES

circa 1795
Least squares: a ubiquitous tool

\[ \hat{a} = X^\dagger y \]

- Weighted least squares?
- Regularized least squares?
- L1-norm instead of L2-norm?
- Nullspace structure?
- Recursive computation?
Linear regression

Python notebooks:
demo_LS.ipynb, demo_LS_polFit.ipynb

Matlab codes:
demo_LS01.m, demo_LS_polFit01.m
Linear regression

- **Least squares is everywhere**: from simple problems to large scale problems.

- It was the earliest form of regression, which was published by **Legendre** in 1805 and by **Gauss** in 1809. They both applied the method to the problem of determining the orbits of bodies around the Sun from astronomical observations.

- The term regression was only coined later by **Galton** to describe the biological phenomenon that the heights of descendants of tall ancestors tend to regress down towards a normal average.

- **Pearson** later provided the statistical context showing that the phenomenon is more general than a biological context.
Multivariate linear regression

By describing the input data as \( X \in \mathbb{R}^{N \times D} \) and the output data as \( y \in \mathbb{R}^N \), we want to find \( a \in \mathbb{R}^{D} \) to have \( y = Xa \).

A solution can be found by minimizing the \( \ell_2 \) norm

\[
\hat{a} = \arg \min_a \| y - Xa \|^2 \\
= \arg \min_a (y - Xa)^\top (y - Xa) \\
= \arg \min_a y^\top y - 2a^\top X^\top y + a^\top X^\top X a
\]

By differentiating with respect to \( a \) and equating to zero

\[-2X^\top y + 2X^\top X a = 0 \iff \hat{a} = (X^\top X)^{-1}X^\top y\]

Moore-Penrose pseudoinverse \( X^\dagger \)
Multiple multivariate linear regression

By describing the input data as $\mathbf{X} \in \mathbb{R}^{N \times D_I}$ and the output data as $\mathbf{Y} \in \mathbb{R}^{N \times D_O}$, we want to find $\mathbf{A} \in \mathbb{R}^{D_I \times D_O}$ to have $\mathbf{Y} = \mathbf{X} \mathbf{A}$.

A solution can be found by minimizing the Frobenius norm

$$\hat{\mathbf{A}} = \operatorname{arg\,min}_{\mathbf{A}} \| \mathbf{Y} - \mathbf{X} \mathbf{A} \|^2_F$$

$$= \operatorname{arg\,min}_{\mathbf{A}} \operatorname{tr}\left( (\mathbf{Y} - \mathbf{X} \mathbf{A})^\top (\mathbf{Y} - \mathbf{X} \mathbf{A}) \right)$$

$$= \operatorname{arg\,min}_{\mathbf{A}} \operatorname{tr}(\mathbf{Y}^\top \mathbf{Y} - 2\mathbf{A}^\top \mathbf{X}^\top \mathbf{Y} + \mathbf{A}^\top \mathbf{X}^\top \mathbf{X} \mathbf{A})$$

By differentiating with respect to $\mathbf{A}$ and equating to zero

$$-2\mathbf{X}^\top \mathbf{Y} + 2\mathbf{X}^\top \mathbf{X} \mathbf{A} = 0 \iff \hat{\mathbf{A}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{Y}$$

Moore-Penrose pseudoinverse $\mathbf{X}^\dagger$
Example of multivariate linear regression

\[ \mathbf{x} = [x_1, x_2] \]

\[ N = 40 \]

\[ D^I = 2 \]

\[ D^O = 1 \]
Polynomial fitting with least squares

\[ \widehat{A} = X^\dagger Y \]

Degree 0 (e=24.31)

\[ x = 1 \]

Degree 1 (e=15.65)

\[ x = [1, x] \]

Degree 2 (e=8.53)

\[ x = [1, x, x^2] \]

Degree 3 (e=8.25)

\[ x = [1, x, x^2, x^3] \]

Degree 4 (e=6.48)

\[ x = [1, x, x^2, x^3, x^4] \]

Degree 5 (e=6.47)

\[ x = [1, x, x^2, x^3, x^4, x^5] \]
Singular value decomposition (SVD)

\[ X \in \mathbb{R}^{N \times D^T} \]

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 2 \\
0 & 0 & 3 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 4 & 0 & 0 & 0 \\
\end{bmatrix}
= \begin{bmatrix}
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & -1 \\
1 & 0 & 0 & 0 \\
\end{bmatrix}
\begin{bmatrix}
4 & 0 & 0 & 0 & 0 \\
0 & 3 & 0 & 0 & 0 \\
0 & 0 & \sqrt{5} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\begin{bmatrix}
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
\sqrt{0.2} & 0 & 0 & 0 & \sqrt{0.8} \\
0 & 0 & 0 & 1 & 0 \\
-\sqrt{0.8} & 0 & 0 & 0 & \sqrt{0.2} \\
\end{bmatrix}
\]

Matrix with non-negative diagonal entries (singular values of \( X \))

Unitary matrix (orthogonal)

Unitary matrix (orthogonal)

\[ X = U \Sigma V^T \]
Least squares with SVD

\[ X \in \mathbb{R}^{N \times D^T} \]

\[ \hat{A} = X^\top (X^\top X)^{-1} Y \]

\( X \) can be decomposed with the **singular value decomposition**

\[ X = U \Sigma V^\top \]

where \( U \) and \( V \) are \( N \times N \) and \( D^T \times D^T \) orthogonal matrices, and \( \Sigma \) is an \( N \times D^T \) matrix with all its elements outside of the main diagonal equal to 0. With this decomposition, a solution to the least squares problem is given by

\[ \hat{A} = V \Sigma^\dagger U^\top Y \]

where the pseudoinverse of \( \Sigma \) can be easily obtained by inverting the non-zero diagonal elements and transposing the resulting matrix.
Kernels in least squares (nullspace projection)

Python notebook: demo_LS_polFit.ipynb

Matlab code: demo_LS_polFit_nullspace01.m
Kernels in least squares (nullspace)

The pseudoinverse provides a single least norm solution, but we can sometimes obtain other solutions by employing a nullspace projection operator $N$

$$\hat{A} = X^\dagger Y + \underbrace{(I - X^\dagger X)}_{N} V$$

$V$ can be any vector/matrix (typically, a gradient minimizing a secondary objective function).

The nullspace projection guarantees that $\|Y - X\hat{A}\|_F^2$ is still minimized.
Kernels in least squares (nullspace) \[ \hat{A} = X^\dagger Y + (I - X^\dagger X)V \]

An alternative way of computing the nullspace projection matrix is to exploit the singular value decomposition

\[ X^\dagger = U\Sigma V^\top \]

to compute

\[ N = \tilde{U}\tilde{U}^\top \]

where \( \tilde{U} \) is a matrix formed by the columns of \( U \) that span for the corresponding zero rows in \( \Sigma \).

This can for example be implemented in Matlab/Octave with

\[
[U,S,V] = \text{svd}(\text{pinv}(X))
\]
\[ sp = \text{sum}(S,2) < 1\text{E}-1 \]
\[ N = U(\cdot,sp) \ast U(\cdot,sp)' \]
Example with polynomial fitting

\[ \hat{a} = X^\dagger y + Nv \quad \text{with} \quad x = [1, x, x^2, \ldots, x^6] \]

\[ v \sim \mathcal{N}(0, I) \]

\[ X \in \mathbb{R}^{4 \times 7} \]
\[ y \in \mathbb{R}^4 \]
\[ \hat{a} \in \mathbb{R}^7 \]
Forward kinematics

\[
\begin{bmatrix}
  x_1 \\ x_2
\end{bmatrix} = 
\begin{bmatrix}
  a_1 \cos(q_1) + a_2 \cos(q_1 + q_2) \\ a_1 \sin(q_1) + a_2 \sin(q_1 + q_2)
\end{bmatrix}
\]
Find $q$ to have $f(q)=0$

\[ f'(q) = \frac{f(q)}{\Delta q} \]

\[ \Delta q = \frac{f(q)}{f'(q)} \]
Gauss-Newton algorithm

\[ q \leftarrow q - \frac{f(q)}{f'(q)} \]
Example with robot inverse kinematics

Gauss-Newton algorithm

\[
q \leftarrow q - \alpha J^\dagger(q)f(q)
\]

\[
J(q) = \begin{bmatrix}
\frac{\partial f_1(q)}{\partial q_1} & \frac{\partial f_1(q)}{\partial q_2} \\
\frac{\partial f_2(q)}{\partial q_1} & \frac{\partial f_2(q)}{\partial q_2}
\end{bmatrix}
\]

\( \in \mathbb{R}^{2 \times 2} \)
Example with robot inverse kinematics

Forward kinematics is computed with

\[ \mathbf{x}_t = f(q_t) \quad \iff \quad \dot{\mathbf{x}}_t = \frac{\partial \mathbf{x}_t}{\partial t} = \frac{\partial f(q_t)}{\partial q_t} \frac{\partial q_t}{\partial t} = J(q_t) \dot{q}_t \]

where \( J(q_t) = \frac{\partial f(q_t)}{\partial q_t} \) is a Jacobian matrix.

An inverse kinematics solution can be computed with

\[ \hat{\mathbf{q}}_t = J^+(q_t) \dot{\mathbf{x}}_t + N(q_t) g(q_t) \]
Example with robot inverse kinematics

\[ \hat{q}_t = J^\dagger(q_t) \dot{x}_t + N(q_t) g(q_t) \]

→ Primary constraint: keeping the tip of the robot still

\[ = J^\dagger(q_t) \begin{bmatrix} 0 \\ 0 \end{bmatrix} + N(q_t) \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \]

→ Secondary constraint: trying to move the first joint
Example with robot inverse kinematics

\[ \hat{q}_t = J_{t}^{R\dagger} \dot{x}_t^R + N_{t}^R q_t^L \]

Tracking target with left hand, if possible

\[ \dot{q}_t^L = (J_{t}^L N_{t}^R)\dagger (\dot{x}_t^L - J_{t}^L J_{t}^{R\dagger} \dot{x}_t^R) \]

Tracking target with right hand

\[ \dot{x}_t^R = \kappa_P (\hat{x}_t^R - x_t^R) \]
\[ \dot{x}_t^L = \kappa_P (\hat{x}_t^L - x_t^L) \]

(moving target)

(moving target)
Ridge regression
(Tikhonov regularization, penalized least squares)

Python notebook:
demo_LS_polFit.ipynb

Matlab example:
demo_LS_polFit02.m
Ridge regression (Tikhonov regularization)

The least squares objective can be modified to give preference to a particular solution with

$$\hat{A} = \arg \min_A \|Y - XA\|_F^2 + \|\Gamma A\|_F^2$$

$$= \arg \min_A \text{tr}\left((Y - XA)^\top(Y - XA)\right) + \text{tr}\left((\Gamma A)^\top\Gamma A\right)$$

By differentiating with respect to $A$ and equating to zero, we can see that

$$-2X^\top Y + 2X^\top XA + 2\Gamma^\top\Gamma A = 0$$

yielding

$$\hat{A} = (X^\top X + \Gamma^\top\Gamma)^{-1}X^\top Y$$

If $\Gamma = \lambda I$ with $\lambda \ll 1$ (i.e., giving preference to solutions with smaller norms), the process is known as $\ell_2$ regularization.
Ridge regression (Tikhonov regularization)

Ridge regression can alternatively be computed with augmented matrices

\[
\tilde{X} = \begin{bmatrix} X \\ \Gamma \end{bmatrix} \quad \tilde{Y} = \begin{bmatrix} Y \\ 0 \end{bmatrix}
\]

with \(0 \in \mathbb{R}^{D_I \times D_O}\) and \(\Gamma \in \mathbb{R}^{D_I \times D_I}\), yielding

\[
\hat{A} = (\tilde{X}^\top \tilde{X})^{-1} \tilde{X}^\top \tilde{Y}
\]

\[
= \left( \begin{bmatrix} X^\top \\ \Gamma \end{bmatrix} \begin{bmatrix} X \\ \Gamma \end{bmatrix} \right)^{-1} \begin{bmatrix} X^\top \\ \Gamma \end{bmatrix} \begin{bmatrix} Y \\ 0 \end{bmatrix}
\]

\[
= (X^\top X + \Gamma^\top \Gamma)^{-1} X^\top Y
\]

\(X \in \mathbb{R}^{N \times D_I}\)
\(Y \in \mathbb{R}^{N \times D_O}\)
\(A \in \mathbb{R}^{D_I \times D_O}\)
Ridge regression also has links with SVD. For the singular value decomposition

\[ X = U \Sigma V^\top \]

with \( \sigma_i \) the singular values in the diagonal of \( \Sigma \), a solution to the ridge regression problem is given by

\[ \hat{A} = V \tilde{\Sigma} U^\top Y \]

where \( \tilde{\Sigma} \) has diagonal values

\[ \tilde{\sigma}_i = \frac{\sigma_i}{\sigma_i^2 + \lambda^2} \]

and has zeros elsewhere.
Ridge regression (Tikhonov regularization)

$D^\perp = 7$ (polynomial of degree 7)

\[ \lambda = 10^{-10} \ (e=6.33) \]

\[ \lambda = 10^{10} \ (e=13.67) \]

\[ \lambda = 10^{20} \ (e=24.55) \]
Weighted least squares (Generalized least squares)

Python notebook: demo_LS_weighted.ipynb

Matlab example: demo_LS_weighted01.m
Weighted least squares

By describing the input data as $X \in \mathbb{R}^{N \times D_I}$ and the output data as $Y \in \mathbb{R}^{N \times D_O}$, with a weight matrix $W \in \mathbb{R}^{N \times N}$, we want to minimize

$$
\hat{A} = \arg \min_A \| Y - X A \|_{F, W}^2
$$

$$
= \arg \min_A \text{tr} \left( (Y - X A)^T W (Y - X A) \right)
$$

$$
= \arg \min_A \text{tr} \left( Y^T W Y - 2 A^T X^T W Y + A^T X^T W X A \right)
$$

By differentiating with respect to $A$ and equating to zero

$$
-2X^T W Y + 2X^T W X A = 0 \iff \hat{A} = \left( X^T W X \right)^{-1} X^T W Y
$$
Weighted least squares

\[
\hat{A} = (X^T W X)^{-1} X^T W Y
\]

Ordinary least squares

Weighted least squares

Color intensity proportional to weight
Weighted least squares - Example I

\[ \hat{A} = (X^\top W X)^{-1} X^\top W Y \]

\[ \hat{q}_t = (J^\top W^x J)^{-1} J^\top W^x \dot{x}_t \]

\[
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix}
\]

\[
\begin{pmatrix}
1 & 0 \\
0 & 0.01
\end{pmatrix}
\]

\[
\begin{pmatrix}
0.01 & 0 \\
0 & 1
\end{pmatrix}
\]
Weighted least squares - Example II

\[ \hat{A} = WX^T(XWX^T)^{-1}Y \]

\[ \hat{q}_t = W^Q J^T(JW^QJ^T)^{-1}\dot{\mathbf{x}}_t \]

\[ W^Q = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \]

\[ W^Q = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 1 \end{bmatrix} \]

\[ W^Q = \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \]
Iteratively reweighted least squares (IRLS)

Python notebook: demo_LS_weighted.ipynb

Matlab code: demo_LS_IRLS01.m
Iteratively reweighted least squares (IRLS)

- **Iteratively Reweighted Least Squares** generalizes least squares by raising the error to a power that is less than 2: → can no longer be called “least squares”

- The strategy is that an error $|e|^p$ can be rewritten as
  $|e|^p = |e|^{p-2} e^2$.

- $|e|^{p-2}$ can be interpreted as a weight, which is used to minimize $e^2$ with **weighted least squares**.

- $p=1$ corresponds to **least absolute deviation regression**.
Iteratively reweighted least squares (IRLS)

$$|e|^p = |e|^{p-2} e^2$$

For an $\ell_p$ norm objective defined by

$$\hat{A} = \arg\min_A \|Y - X A\|_{F,p}^2$$

$\hat{A}$ is estimated by starting from $W = I$ and iteratively computing

$$\hat{A} \leftarrow (X^T W X)^{-1} X^T W Y$$

$$W_{t,t} \leftarrow |Y_t - X_t A|^{p-2} \quad \forall t \in \{1, \ldots, T\}$$
IRLS as regression robust to outliers

Ordinary least squares (e=14.6)

Iteratively reweighted least squares (e=12.6)

Color darkness proportional to weight
Recursive least squares

Python notebook: 
demo_LSRecursive.ipynb

Matlab code: 
demo_LSRecursive01.m
Recursive least squares

Sherman-Morrison-Woodbury relation:

\[
(B + UV)^{-1} = B^{-1} - \underbrace{B^{-1}U (I + VB^{-1}U)^{-1} V B^{-1}}_{E}
\]

with \( U \in \mathbb{R}^{n \times m} \) and \( V \in \mathbb{R}^{m \times n} \).

When \( m \ll n \), the correction term \( E \) can be computed more efficiently than inverting \( B + UV \).

By defining \( B = XX^\top \), the above relation can be exploited to update a least squares solution when new datapoints are available.
Recursive least squares

\[(B + UV)^{-1} = B^{-1} - B^{-1}U(I + VB^{-1}U)^{-1}VB^{-1}\]

If \(X_{\text{new}} = [X^\top, V^\top]^\top\) and \(Y_{\text{new}} = [Y^\top, C^\top]^\top\), we then have

\[B_{\text{new}} = X_{\text{new}}^\top X_{\text{new}}\]
\[= X^\top X + V^\top V\]
\[= B + V^\top V\]

whose inverse can be computed with

\[B_{\text{new}}^{-1} = B^{-1} - B^{-1}V^\top(I + VB^{-1}V^\top)^{-1}VB^{-1}\]

which is exploited to efficiently compute the update as

\[\hat{A}_{\text{new}} = \hat{A} + K(C - V\hat{A})\]

with Kalman gain \(K = B^{-1}V^\top(I + VB^{-1}V^\top)^{-1}\)
Recursive least squares

Ordinary least squares (e=11.0)

Recursive least squares (e=11.0)
Linear regression:
Examples of applications
Koopman operators in control

\[ \dot{x} = f(x) \]
\[
\begin{bmatrix}
\dot{x}_1 \\
\dot{x}_2 
\end{bmatrix}
= \begin{bmatrix}
\lambda_1 x_1 \\
\lambda_2 (x_2 - x_1^2)
\end{bmatrix}
\]

**Main challenge in Koopman analysis:** How to find these basis functions?

\[ \dot{y} = A y \]
\[
\begin{bmatrix}
\dot{y}_1 \\
\dot{y}_2 \\
\dot{y}_3 
\end{bmatrix}
= \begin{bmatrix}
\lambda_1 & 0 & 0 \\
0 & \lambda_2 & -\lambda_2 \\
0 & 0 & 2\lambda_1 
\end{bmatrix}
\begin{bmatrix}
y_1 \\
y_2 \\
y_3 
\end{bmatrix}
\]

with
\[
\begin{bmatrix}
y_1 \\
y_2 \\
y_3 
\end{bmatrix}
= \begin{bmatrix}
x_1 \\
x_2 \\
x_1^2 
\end{bmatrix}
\]

\[ y_3 = \frac{\partial y_3}{\partial x_1} \dot{x}_1 \]
\[ = 2x_1 \lambda_1 x_1 \]
\[ = 2\lambda_1 y_3 \]
Linear quadratic tracking (LQT)

\[ \min_u \sum_{t=1}^{T} \frac{1}{Q_t} \| \mu_t - x_t \|^2 + \frac{1}{R_t} \| u_t \|^2 \]

s.t. \[ x_{t+1} = Ax_t + Bu_t \] System dynamics

- \( x_t \) state variable (position+velocity)
- \( \mu_t \) desired state
- \( u_t \) control command (acceleration)
- \( Q_t \) precision matrix
- \( R_t \) control weight matrix

\[ Q_T = \sum_{T}^{-1} \]

Track path! Use low control commands!
How to solve this objective function?

\[ \min_u \sum_{t=1}^{T} \left( \| \mu_t - x_t \|^2_{Q_t} + \| u_t \|^2_{R_t} \right) \]

s.t. \[ x_{t+1} = Ax_t + Bu_t \] System dynamics

- Pontryagin’s max. principle, Riccati equation, Hamilton-Jacobi-Bellman (the Physicist perspective)
- Dynamic programming (the Computer Scientist perspective)
- Linear algebra (the Algebraist perspective)

Track path! Use low control commands!
Let’s first re-organize the objective function...

\[
c = \sum_{t=1}^{T} \left( (\mu_t - x_t)^T Q_t (\mu_t - x_t) + u_t^T R_t u_t \right)
\]

\[
= (\mu - x)^T Q (\mu - x) + u^T R u
\]

\[
Q = \begin{bmatrix}
Q_1 & 0 & \cdots & 0 \\
0 & Q_2 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & Q_T
\end{bmatrix}
\]

\[
R = \begin{bmatrix}
R_1 & 0 & \cdots & 0 \\
0 & R_2 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & R_T
\end{bmatrix}
\]

\[
\mu = \begin{bmatrix}
\mu_1 \\
\mu_2 \\
\vdots \\
\mu_T
\end{bmatrix}
\]

\[
x = \begin{bmatrix}
x_1 \\
x_2 \\
\vdots \\
x_T
\end{bmatrix}
\]

\[
u = \begin{bmatrix}
u_1 \\
u_2 \\
\vdots \\
u_T
\end{bmatrix}
\]
Let's then re-organize the constraint...

\[ x_{t+1} = A x_t + B u_t \]

\[ x_2 = A x_1 + B u_1 \]
\[ x_3 = A x_2 + B u_2 = A(A x_1 + B u_1) + B u_2 \]
\[ \vdots \]
\[ x_T = A^{T-1} x_1 + A^{T-2} B u_1 + A^{T-3} B u_2 + \cdots + B_{T-1} u_{T-1} \]

\[
\begin{bmatrix}
  x_1 \\
  x_2 \\
  x_3 \\
  \vdots \\
  x_T
\end{bmatrix} =
\begin{bmatrix}
  I \\
  A \\
  A^2 \\
  \vdots \\
  A^{T-1}
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  x_2 \\
  x_3 \\
  \vdots \\
  x_T
\end{bmatrix} +
\begin{bmatrix}
  0 & 0 & \cdots & 0 & 0 \\
  B & 0 & \cdots & 0 & 0 \\
  AB & B & \cdots & 0 & 0 \\
  \vdots & \vdots & \ddots & \vdots \\
  A^{T-2} B & A^{T-3} B & \cdots & B & 0
\end{bmatrix}
\begin{bmatrix}
  u_1 \\
  u_2 \\
  \vdots \\
  u_T
\end{bmatrix}
\]

\[ x = S^x x_1 + S^u u \]
Linear quadratic tracking (LQT)

The constraint can then be put into the objective function:

\[ x = S^x x_1 + S^u u \]

\[ c = (\mu - x)^\top Q (\mu - x) + u^\top R u \]

\[ = (\mu - S^x x_1 - S^u u)^\top Q (\mu - S^x x_1 - S^u u) + u^\top R u \]

Solving for \( u \) results in the analytic solution:

\[ \hat{u} = (S^u^\top Q S^u + R)^{-1} S^u^\top Q (\mu - S^x x_1) \]
Linear quadratic tracking (LQT)

\[
\hat{u} = (S^u Q S^u + R)^{-1} S^u Q (\mu - S^x x_1)
\]
\[
\hat{\Sigma}^u = (S^u Q S^u + R)^{-1}
\]

\[
\hat{x} = S^x x_1 + S^u \hat{u}
\]
\[
\hat{\Sigma}^x = S^u (S^u Q S^u + R)^{-1} S^u^T
\]

The distribution in control space can be projected back to the state space.

Passing through 3 keypoints with varying precision.

\begin{align*}
  &t=0 \quad \text{Initial state} \\
  &t=0.3 \quad \text{Third state} \\
  &t=0.6 \quad \text{Sixth state} \\
  &t=1 \quad \text{Final state}
\end{align*}
References

Regression

