EE 263 Homework 3

Luis A. Perez

The smoothest input that takes the state to zero

Solution:

(a) We beging by trying to express the problem in a way that's more susceptible to the techniques that we're learned in the course. Firstly, let us simply expand $x(20)$. We have:

$$\begin{aligned}
x(20) &= Ax(19) + Bu(19) && \text{(Using our recurrence)} \\
&= A(Ax(18) + Bu(18)) + Bu(19) && \text{(Re-use reccurence)} \\
&= A^2x(18) + ABu(18) + Bu(19) && \text{(Simplify)} \\
&= A^2(Ax(17) + Bu(17)) + ABu(18) + Bu(19) && \text{(Apply reccurrences again)} \\
&= A^3x(17) + A^2Bu(17) + ABu(18) + Bu(19) \\
&\vdots \\
&= A^{20}x(0) + \sum_{i=0}^{19} A^iBu(19-i)
\end{aligned}$$

We know that $x(20) = 0$, we re-arranginh the above, we have:

$$\sum_{i=0}^{19} A^iBu(19-i) = -A^{20}x(0)$$

$$\begin{bmatrix} B & AB & A^2B & A^3B & \cdots & A^{19}B \end{bmatrix} \begin{bmatrix} u(19) \\ u(18) \\ u(17) \\ u(16) \\ \vdots \\ u(0) \end{bmatrix} = -A^{20}x(0)$$

$$Cu = y$$

where we have

$$C = \begin{bmatrix} B & AB & A^2B & \cdots & A^{19}B \end{bmatrix} \in \mathbb{R}^{3\times 20}$$
$$u = \begin{bmatrix} u(19) \\ u(18) \\ u(17) \\ \vdots \\ u(0) \end{bmatrix} \in \mathbb{R}^{20}$$
$$y = -A^{20}x(0) \in \mathbb{R}^3$$

with C and y known, and u as the parameters we're trying to find. Note that the above is an underdetermined system of linear equations with 3 equations and 20 variables.

The problem above would be straight-forward to solve if what our objective did was minimize the norm of u. However, we're actually tryining to minize the the smoothesness. So we'll need to do a change of variables. In fact, we propose defining our variables as:

$$\begin{aligned}
\begin{bmatrix} u(19) \\ u(18) \\ u(17) \\ \vdots \\ u(1) \\ u(0) \end{bmatrix} &= \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & 1 \\ 0 & 1 & 1 & \cdots & 1 & 1 \\ 0 & 0 & 1 & \cdots & 1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} u(19) - u(18) \\ u(18) - u(17) \\ u(17) - u(16) \\ \vdots \\ u(1) - u(0) \\ u(0) - u(-1) \end{bmatrix} \\
&= \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & 1 \\ 0 & 1 & 1 & \cdots & 1 & 1 \\ 0 & 0 & 1 & \cdots & 1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} \delta(19) \\ \delta(18) \\ \delta(17) \\ \vdots \\ \delta(1) \\ \delta(0) \end{bmatrix} \\
&= T\delta
\end{aligned}$$

where $T \in \mathbb{R}^{20\times 20}$ is an upper triangular matrix of 1s and $\delta \in \mathbb{R}^{20}$ is the differences between each subsequent value. Using these, we can write our problem as:

$$(CT)\delta = y$$

And looking at our objective, we have:

$$\begin{aligned} J_{\text{smooth}} &= \sqrt{\frac{1}{20}\sum_{t=0}^{19}(u(t)-u(t-1))^2} \\ &= \frac{1}{\sqrt{20}}\sum_{t=0}^{19}\delta(t)^2 \\ &= \frac{1}{\sqrt{20}}||\delta|| \end{aligned}$$

So we can easily find $\hat{\delta}$ such that $||del\hat{t}a||$ is minimal (so our objective is satisfied) and $(CT)\delta = y$ (so our equations hold). In fact, the solution is given by:

$$\hat{\delta} = (CT)^T(CTT^TC)^{-1}y = -(CT)^T(CTT^TC)^{-1}A^{20}x(0)$$

Once we have $\hat{\delta}$, we can use T to obtain $\hat{u}$ as:

$$\hat{u} = T\hat{\delta}$$

which will have a smoothness factor of:

$$J_{\text{smooth}} = \frac{1}{\sqrt{20}}||\hat{\delta}||$$

Solving as per the above (see attached code), the plot of $\hat{u}$ is provided in Figure 1, with the corresponding $J_{\text{smooth}} = 1.124615543256077$.

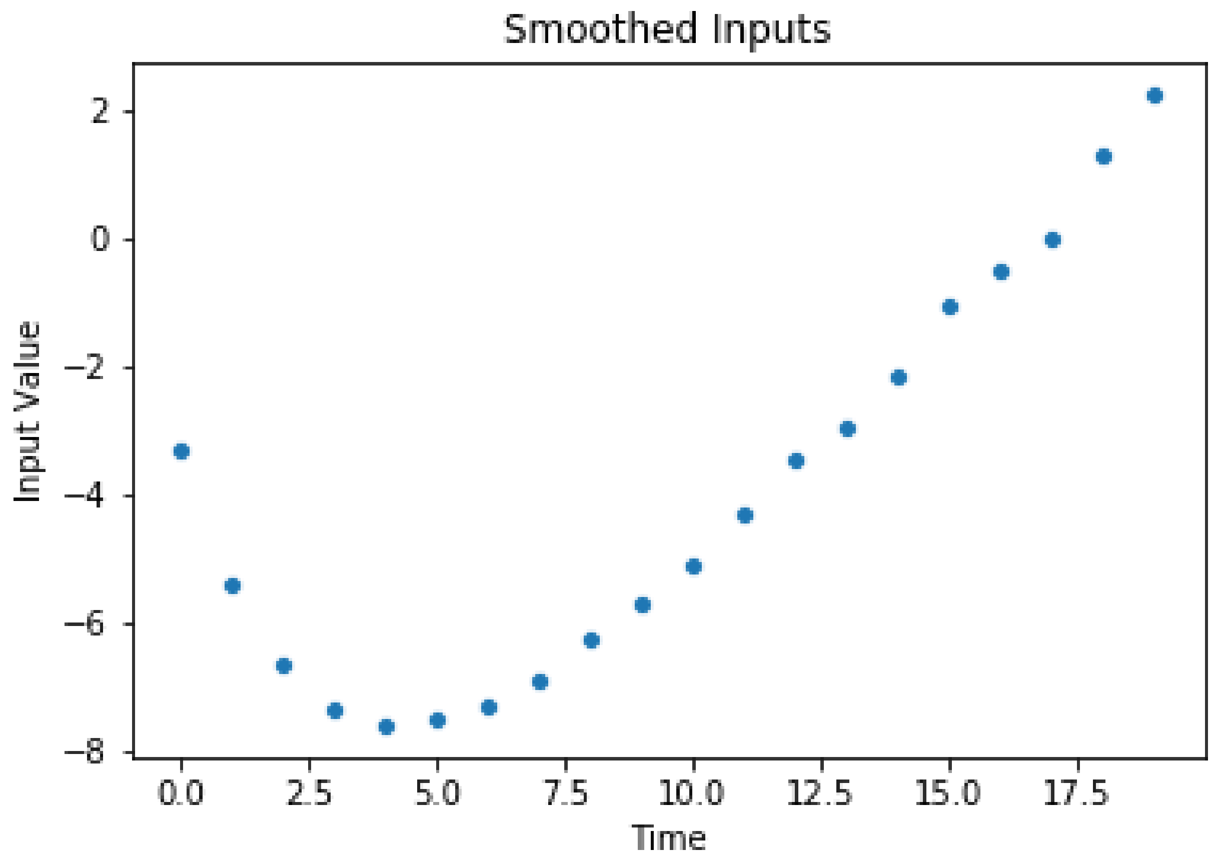


Figure 1: Smoothest inputs to achieve $x(20) = 0$ for our linear dynamical system. This achieves $J_{\text{smooth}} = 1.1246$.

Minimum fuel and minimum peak input solutions.

Solution: First, we verify the inequality $w^T v \leq ||v||_\infty ||w||_1$ for $v, w \in \mathbb{R}^p$. We have:

$$\begin{aligned}
w^T v &= \sum_{i=1}^{p} w_i v_i && \text{(Definition of dot product)} \\
&\leq \sum_{i=1}^{p} |w_i||v_i| && \text{(Sum of absolutes must be larger or equal)} \\
&\leq \max_{i=1,\cdots,p} |v_i| \sum_{i=1}^{p} |w_i| && \text{(Multiply by the max } v_i \text{ instead of each } v_i) \\
&= ||v||_\infty ||w||_1 && \text{(Definition of norms)}
\end{aligned}$$

If we let $z \in \mathbb{R}^n$ be any solution of $Az = y$, and let $\lambda \in \mathbb{R}^m$ be such that $A^T\lambda \neq 0$, then using $z = w \in \mathbb{R}^n$ and $v = A^T\lambda \in \mathbb{R}^n$, by the inequality we just proved, we must have:

$$\begin{aligned}
||z||_1 &\geq \frac{z^T(A^T\lambda)}{||A^T\lambda||_\infty} && \text{(By inequality just proved)} \\
&= \frac{\lambda^T A z}{||A^T\lambda||_\infty} && \text{(Numerator is one-dimensional, so its transpose is equal to itself)} \\
&= \frac{\lambda^T y}{||A^T\lambda||_\infty} && \text{(Using } y = Az)
\end{aligned}$$

Therefore, if we can find a vector $x_{\text{mf}} \in \mathbb{R}^n$ and $\lambda \in \mathbb{R}^m$ such that it both (1) solves $Ax_{\text{mf}} = y$ and (2) $||x_{\text{mf}}||_1 = \frac{\lambda^T y}{||A^Y\lambda||_\infty}$, we will have found the minimum fuel solution. This is because the solution solves (1), and all other solutions will have a 1-norm larger than x_{mf} by (2).

From the lecture notes, we have the linear system:

$$y = Ax$$

where

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{10} \end{bmatrix} \in \mathbb{R}^{10}$$

$$A = \begin{bmatrix} 9 + \frac{1}{2} & 8 + \frac{1}{2} & \cdots & \frac{1}{2} \\ 1 & 1 & \cdots & 1 \end{bmatrix} \in \mathbb{R}^{2\times 10}$$

$$y = \begin{bmatrix} p(10) \\ \dot{p}(10) \end{bmatrix}$$

We wish to show that:

$$x_{\text{mf}} = \begin{bmatrix} \frac{1}{9} \\ 0 \\ \vdots \\ 0 \\ -\frac{1}{9} \end{bmatrix} \in \mathbb{R}^{10}$$

Following the hint, we will use $\lambda = \begin{bmatrix} 1 \\ -5 \end{bmatrix}$ to prove that the above solution is the minimum-fuel solution. First, we verify that x_{mf} is a solution to our system:

$$\begin{aligned} Ax_{\text{mf}} &= \begin{bmatrix} 9 + \frac{1}{2} & 8 + \frac{1}{2} & \cdots & \frac{1}{2} \\ 1 & 1 & \cdots & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{9} \\ 0 \\ \vdots \\ 0 \\ -\frac{1}{9} \end{bmatrix} \\ &= \begin{bmatrix} 1 \\ 0 \end{bmatrix} = y \end{aligned}$$

The next step is to verify the norm $||x_{\text{mf}}||_1$ is equal to $\frac{\lambda^T y}{||A^T\lambda||_\infty}$.

$$\begin{aligned} \frac{\lambda^T y}{||A^T \lambda||_\infty} &= \frac{1}{|| \begin{bmatrix} -4\frac{1}{2} \\ -3\frac{1}{2} \\ -2\frac{1}{2} \\ -1\frac{1}{2} \\ -\frac{1}{2} \\ \frac{1}{2} \\ 1\frac{1}{2} \\ 2\frac{1}{2} \\ 3\frac{1}{2} \\ 4\frac{1}{2} \end{bmatrix} ||_\infty} \\ &= \frac{2}{9} \\ &= |\frac{1}{9}| + |-\frac{1}{9}| \\ &= ||x_{\text{mf}}|| \end{aligned}$$

As argued previously, since x_{mf} satisfies both properties, it must be the case that it is the minimum-fuel solution. We also verify in code that the x_{ls} norm is 0.303 and the x_{mf} is 0.2, which is lower.

As far as the minimum peak force vector, x_{mp}, a good guess would be:

$$x_{\text{mp}} = \begin{bmatrix} \frac{1}{25} \\ \frac{1}{25} \\ \frac{1}{25} \\ \frac{1}{25} \\ \frac{1}{25} \\ -\frac{1}{25} \\ -\frac{1}{25} \\ -\frac{1}{25} \\ -\frac{1}{25} \\ -\frac{1}{25} \end{bmatrix}$$

Using a similar argument as before, we can verify the above is the mimum peak force vector by verifying that (1) it is a solution such that $Ax_{\text{mp}} = y$ and that the following equality holds, for some $\lambda \in \mathbb{R}^m$ such that $A^T\lambda \neq 0$ (in our case, we use $\lambda = \begin{bmatrix} 1 \\ -5 \end{bmatrix}$):

$$||x_{\text{mp}}||_\infty = \frac{\lambda^T y}{||A^T y||_1}$$

This is because, using the inequality proved previously, we have that for any solution $Az = y$ must have the following hold true:

$$||z||_\infty \geq \frac{\lambda^T y}{||A^T \lambda||_1}$$

See the attaced code, but it is easy for use to verify that x_{mp} is a valid solution to our system. Furthermore, we can also readily verify that:

$$\begin{aligned} \frac{\lambda^T y}{||A^T \lambda||_1} &= \frac{1}{|| \begin{bmatrix} -4\frac{1}{2} \\ -3\frac{1}{2} \\ -2\frac{1}{2} \\ -1\frac{1}{2} \\ -\frac{1}{2} \\ \frac{1}{2} \\ 1\frac{1}{2} \\ 2\frac{1}{2} \\ 3\frac{1}{2} \\ 4\frac{1}{2} \end{bmatrix} ||_1} \\ &= \frac{1}{9+7+5+3+1} \\ &= \frac{1}{25} \\ &= ||x_{\text{mf}}||_\infty \end{aligned}$$

Singularity of the KKT matrix

Solution:

(a) Let us suppose that $C \in \mathbb{R}^{k\times n}$ is not full rank. Then we have the following:

$$\begin{aligned}
\dim(\mathbf{Img}(C^T)) &= \dim(\mathbf{Ker}(C))^{\perp} \\
&\qquad\qquad \text{(Property of orthogonal complement and transpose)} \\
&= n - \dim(\mathbf{Ker}(C)) && \text{(Definition of complement space)} \\
&= n - (n - \dim(\mathbf{Img}(C))) && \text{(Rank-nullity theorem)} \\
&= \dim(\mathbf{Img}(C)) \\
&< k && \text{(C is assumed to not be full-rank)}
\end{aligned}$$

The above shows that the rows of C must be linearly-dependent (there are k rows, but they don't span a k-dimensional space). Therefore, the last k columns $\begin{bmatrix} C^T \\ 0 \end{bmatrix}$ of the KKT matrix are not linearly independent, which means the KKT matrix is singular.

(b) Let us suppose that we have a $u \in \mathbb{R}^n$ such that $u \in \mathbf{Ker}(A) \cap \mathbf{Ker}(C)$. Then note the following:

$$\begin{aligned}
\begin{bmatrix} A^TA & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} u \\ 0 \end{bmatrix} &= \begin{bmatrix} A^TAu \\ Cu \end{bmatrix} \\
&= \begin{bmatrix} 0 \\ 0 \end{bmatrix}
\end{aligned}$$

As such, we have a non-zero vector in the nullspace of the KKT matrix, therefore the KKT matrix is non-sigular.

(c) For this part, let us suppose that K is singular. This means that there exists a nonzero vector $\begin{bmatrix} u \\ v \end{bmatrix} \in \mathbb{R}^{n+k}$ such that:

$$\begin{bmatrix} A^TA & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = 0$$

From the above, we have the following two equations:

$$\begin{aligned}
A^TAu + C^Tv &= 0 \\
Cu &= 0
\end{aligned}$$

We can immediately conclude that $u \in \text{null}(C)$. Taking the second equation and multiplying on the left by u^T, we have:

$$\begin{aligned}
u^T A^T A u + u^T C^T v &= (Au)^T (Au) + (Cu)^T v \\
&= (Au)^T (Au) + 0 && (Cu = 0) \\
&= ||Au||^2 \\
&= 0 \\
\implies ||Au|| &= 0 \\
\implies u &\in \text{null}(A)
\end{aligned}$$

Suppose $u \neq 0$, then by the arguments above, we have shown that $u \in \text{null}(A)$ and $u \in \text{null}(C)$, which implies $\text{null}(A) \cap \text{null}(C) \neq \{0\}$. On the other hand, if $u = 0$, then we must have $A^T A u + C^T v = 0 \implies C^T v = 0$ with $v \neq 0$. This implies C is not full rank.

Portfolio selection with sector neutrality constraints

Solution:

(a) We begin by rewriting this problem in matrix form as a minimization subject to some constraints. Our problem, stated directly from the problem statement, is to:

$$\begin{array}{ll} \text{maximize} & \mu^T x - \lambda R^{\text{id}} \\ \text{subject to} & \mathbf{1}^T x = 1, Fx = 0 \end{array}$$

If we define the matrix $\Sigma \in \mathbb{R}^{n \times n}$ where $\Sigma_{ii} = \sigma_i^2$ and $\Sigma_{ij} = 0$, and flipping the maximization to minimization, the problem becomes:

$$\begin{array}{ll} \text{minimize} & \lambda x^T \Sigma x - \mu^T x \\ \text{subject to} & \mathbf{1}^T x = 1, Fx = 0 \end{array}$$

To solve the above, we can use Lagrance multipliers. We introduce $a \in \mathbb{R}$ and $b \in \mathbb{R}^k$ $(k+1)$ multipliers, giving us the Lagragian as:

$$L(x, a, b) = \lambda x^T \Sigma x - \mu^T x + b^T F x + a(\mathbf{1}^T x - 1)$$

Taking partial derivatives of the above and setting equal to zero gives us the following set of equations:

$$\begin{aligned} \nabla_x L = 2\lambda\Sigma x - \mu + F^T b + a\mathbf{1} &= 0 \\ \nabla_a L = \mathbf{1}^T x - 1 &= 0 \\ \nabla_b L = Fx &= 0 \end{aligned}$$

The above can be written in block-matrix form as:

$$\begin{bmatrix} 2\lambda\Sigma & \mathbf{1} & F^T \\ \mathbf{1}^T & 0 & 0 \\ F & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ a \\ b \end{bmatrix} = \begin{bmatrix} \mu \\ 1 \\ 0 \end{bmatrix}$$

Finding the optimal x consists of solving this set of linear equations, using the techniques we've learned in class.

(b) Using the data in "sector_neutral_portfolio_data.m", we find the optimal portofolio x using the above technique. See the code for details. The return associated with this portfolio is

$$r = 26.706275038813430$$

with idiosyncratic risk:

$$R^{\text{id}} = 133.6297240848970$$

A simple population model

Solution: We express the given population dynamics model as an autonomous linear system with state $x(t) \in \mathbb{R}^3$ and state matrix $A \in \mathbb{R}^{3\times 3}$. We have:

$$\begin{bmatrix} x_1(t+1) \\ x_2(t+1) \\ x_3(t+1) \end{bmatrix} = \begin{bmatrix} 0 & 2 & 1 \\ 0.6 & 0 & 0 \\ 0 & 0.7 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{bmatrix}$$

Controlling a system using the initial conditions

Solution: We begin by plugging in the parameters given. This gives us the matrix:

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -2 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix}$$

The solution to this lineary dynamical system at time t is given immediately by the formula:

$$x(t) = e^{tA}x(0)$$

where $x(0) = \begin{bmatrix} 0 \\ 0 \\ \alpha_1 \\ \alpha_2 \end{bmatrix}$ We use this results in each of the following problems. As such, note that the first two entries of $x(0)$ are always 0. This means we can drop the first two columns of e^{tA} when solving the system. We call this matrix $E \in \mathbb{R}^{4\times 2}$. Then our initial state is simply $\alpha = \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} \in \mathbb{R}^2$.

(a) In this case, we want to solve the system such that $q_2(10) = 2$. We therefore have the restriction:

$$\begin{aligned} 2 &= q_2(10) \\ &= e_2^T x(10) \\ &= e_2^T E\alpha \end{aligned}$$

This is a system with two-variables and only one equation. As such, finding the least-norm solution can be done immediately. We know these variables satisfy our constraints exactly, and we use the additional variable to minimize the norm.

$$\alpha = E_2^T(E_2E_2^T)^{-1}2$$

where E_2 is the second row of our matrix E. Solving in code, we have:

$$\alpha = \begin{bmatrix} 2.46135192 \\ -9.46473528 \end{bmatrix}$$

(b) In this case, we want to solve the system such that $q_1(10) = 1$ and $q_2(10) = 2$. As such, we have the system:

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \end{bmatrix} \alpha$$

This is a system with two equations and two variables, and as such, this specification is feasible. In fact, we can check that our matrix is invertible, and as such, the one and only solutions is given by:

$$\alpha = \begin{bmatrix} E_1 \\ E_2 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

This gives us:

$$\alpha = \begin{bmatrix} -6.44083425 \\ -11.77979348 \end{bmatrix}$$

which is the unique solution.

(c) In this case, we have the system:

$$\begin{bmatrix} 1 \\ 2 \\ 0 \\ 0 \end{bmatrix} = E\alpha$$

In this case, we have four equations with just two unknowns. As such, this system is not feasible. Instead, we can find the α that comes closest to satisfying the system in the least-squares sense. This is given by:

$$\alpha = (E^T E)^{-1} E^T \begin{bmatrix} 1 \\ 2 \\ 0 \\ 0 \end{bmatrix}$$

which gives us:

$$\alpha = \begin{bmatrix} -0.13614458 \\ -0.34345832 \end{bmatrix}$$

with RMSE of 2.204944721263555.

(d) These restriction actually give rise to an interesting set of equations. We have our matrix $E \in \mathbb{R}^{4\times 2}$ as before. Additional, we compute a matrix $E' \in \mathbb{R}^{4\times 2}$ using the same process but with the new parameters of $m_1 = 1, m_2 = 1.3, k_1 = k_2 = 1$. Then our set of equations can be written as:

$$\begin{bmatrix} q_2(10) \\ q_2(10) \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix} = \begin{bmatrix} E_2 \\ E_2' \end{bmatrix} \alpha$$

This is a system of two equations with two unknowns. It turns out that the matrix above is a 2×2 invertible matrix, so there exists one unique solution to this specification given by:

$$\alpha = \left(\begin{bmatrix} E_2 \\ E_2' \end{bmatrix} \right)^{-1} \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

Computing the above, we have:

$$\alpha = \begin{bmatrix} 10.78304166 \\ -7.30063808 \end{bmatrix}$$

HW5

July 28, 2019

1 HW 5

Author: Luis A. Perez
Last Updated: July 27th, 2019

[206]:
```
import numpy as np
import seaborn as sns
from scipy import linalg
```

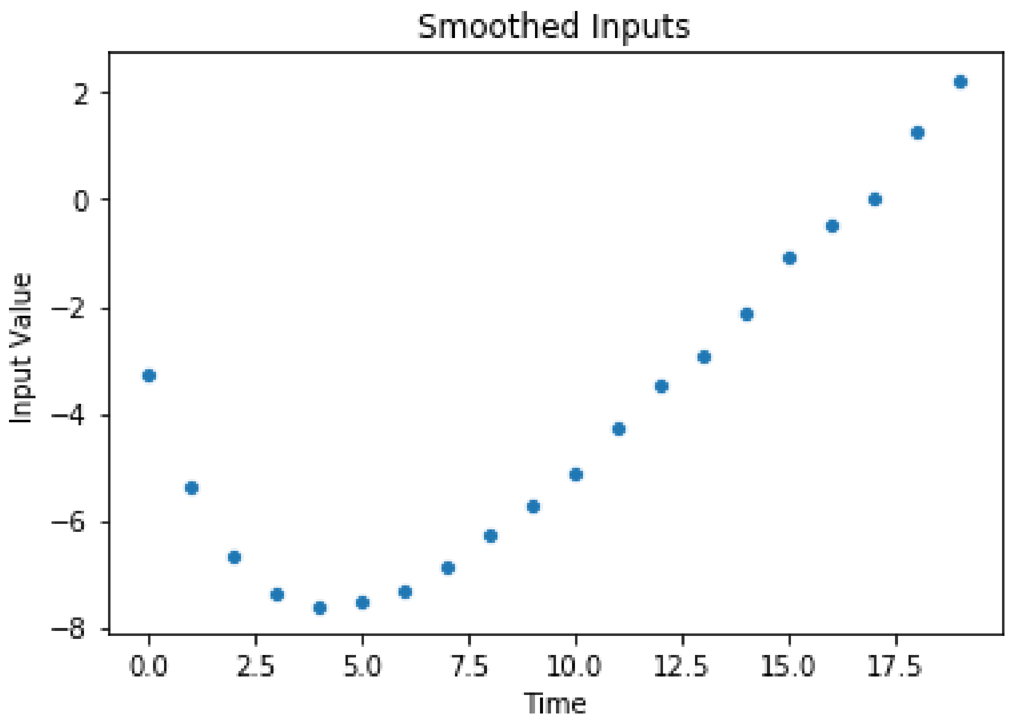
1.1 Problem 1: The smoothest input that takes the state to zero.

[12]:
```
# We follow the outline defined in the homework.
def solveProblem1(A, B, x0, n):
    """Solves to find the smoothest inputs u(t) such that the system:

        x(t + 1) = Ax(t) + Bu(t)

    will have x(n) = 0.

    B must be a vector (not matrix). A must be squared.
    """
    assert B.shape[1] == 1
    assert A.shape[0] == A.shape[1]
    CBlockColumns = []
    currA = np.identity(A.shape[0])
    for i in range(n):
        CBlockColumns.append(np.dot(currA, B))
        currA = np.dot(currA, A)
    C = np.concatenate(CBlockColumns, axis=1)
    A20 = currA
    y = -np.dot(A20, x0)
    # Upper triangular with 1s. Only applies when B is vector.
    T = np.triu(np.ones((n,n)))
    D = np.dot(C,T)
    delta = np.dot(D.T,
                   np.dot(np.linalg.inv(np.dot(D,
                                               D.T
```

```
                                                  )),
                            y
                           )
                    )
    u = np.dot(T, delta)
    RMSE = 1 / np.sqrt(n) * np.linalg.norm(delta)
    print("The RMSE of the data is %s." % (RMSE))

    # Plot the values of u.
    ax = sns.scatterplot(x=np.arange(0,20,1), y=np.flip(u.flatten()))
    ax.set_title("Smoothed Inputs")
    ax.set(ylabel='Input Value', xlabel='Time')
    ax.get_figure().savefig('smoothed_inputs')
```

[13]:
```
def getProblem1Constants():
    """Returns (A,B,x0,n) as defined for Problem 1."""
    A = np.array([
        [1.0, 0.5, 0.25],
        [0.25, 0.0, 1.0],
        [1.0, -0.5, 0.0]
    ])
    B = np.array([
        [1.0],
        [0.1],
        [0.5]
    ])
    x0 = np.array([
        [25.0],
        [0.0],
        [-25],
    ])
    return (A, B, x0, 20)
solveProblem1(*getProblem1Constants())
```

```
The RMSE of the data is 1.124615543256077.
```



[14]:
```
## Problem 2: Minimum fuel and minimum peak input solutions.
```

[132]:
```
def getProblem2Inputs():
    n = 10
    A = np.array([
        list([i + 0.5 for i in range(n-1,-1, -1)]),
        [1] * n
    ])
    y = np.array([
        [1.0],
        [0.0]
    ])
    xmf = np.array([1/n] + ([0] * (n-2)) + [-1/n])
    xmf.shape = (xmf.shape[0], 1)
    return A, y, xmf, n

def solveProblem2(A, y, xmf, n):
    xls = np.dot(A.T, np.dot(np.linalg.inv(np.dot(A,A.T)), y))
    # Verify norm lf xls is < norm of xmf.
    xmfNorm = np.linalg.norm(xmf, ord=1)
    xlsNorm = np.linalg.norm(xls, ord=1)

    # verify norms.
    print("The $x_{\\text{ls}}$ norm is $%s$ and the $x_{\\text{mf}}$ is $%s$."␣
↪% (xlsNorm, xmfNorm))
    assert xmfNorm < xlsNorm

    # compute xmp
    xmp = 1/25 * np.array([[1], [1], [1], [1], [1], [-1], [-1], [-1], [-1],␣
↪[-1]])

    # verify it is a solution to our system
    assert np.allclose(y, np.dot(A, xmp))
    print("Verified x_{mf} solves system of equations.")

    # verify its infinitynorm is equal to what we want.
    xmpNorm = np.linalg.norm(xmp, ord=np.inf)
    lam = np.array([[1],[-5]])
    xmpMin = np.dot(lam.T, y) / np.linalg.norm(np.dot(A.T, lam), ord=1)
    assert np.allclose(xmpMin, xmpNorm)
    print("Verified infinity norm of x_{mf} is minimum.")
```

[133]:
```
solveProblem2(*getProblem2Inputs())
```

```
The $x_{\text{ls}}$ norm is $0.30303030303031$ and the $x_{\text{mf}}$ is
$0.2$.
Verified x_{mf} solves system of equations.
Verified infinity norm of x_{mf} is minimum.
```

1.2 Problem 4: Singularity of the KKT matrix

[140]:
```
def getProblem4Inputs():
    # data for sector neutral portfolio selection problem

    # number of assets
    n = 40
    # number of sectors
    k = 10
    # risk aversion parameter
    lam = 1.000000e-01

    # asset mean returns
    mu = np.array([
        [-5.724206e-03],
        [-5.738256e-02],
        [-2.066057e-02],
        [3.954949e-02],
        [6.380350e-02],
        [-3.452340e-02],
        [-4.097040e-03],
        [-4.511063e-02],
        [3.559544e-02],
        [8.854415e-02],
        [-6.551461e-02],
        [-1.581188e-01],
        [-1.900198e-01],
        [-6.725867e-02],
        [-9.943040e-02],
        [1.582090e-02],
        [6.838301e-02],
        [-5.363577e-02],
        [-4.637255e-02],
        [2.441328e-01],
        [-5.528427e-02],
        [-5.774056e-02],
        [-3.624141e-02],
        [-1.385253e-01],
        [2.113887e-01],
        [4.293215e-02],
        [-7.668623e-02],
        [-1.267691e-01],
        [3.765783e-03],
        [1.465808e-01],
        [2.710706e-02],
        [6.954837e-03],
        [-1.363490e-01],
        [1.100747e-01],
        [-2.595972e-01],
        [7.837806e-02],
        [-5.963350e-02],
        [-9.465257e-03],
        [1.992940e-01],
        [1.081944e-02],
])

 # standard deviations of asset returns
 sigmas = np.array([
        [2.412026e-01],
        [9.836163e-02],
        [8.892398e-02],
        [3.176707e-01],
        [2.709629e-01],
        [4.297720e-01],
        [3.454919e-01],
        [4.021115e-01],
        [4.301839e-01],
        [2.693837e-01],
        [2.073825e-01],
        [1.649149e-01],
        [3.137776e-01],
        [3.958668e-01],
        [1.851636e-01],
        [1.934122e-01],
        [3.474158e-01],
        [4.082042e-01],
        [4.021908e-01],
        [2.399173e-01],
        [1.282238e-01],
        [3.599238e-01],
        [2.555882e-01],
        [3.498137e-01],
        [2.583165e-01],
        [2.955634e-01],
        [1.470161e-01],
        [2.495283e-01],
        [2.775099e-01],
        [3.687146e-01],
        [3.448759e-01],
        [1.207333e-01],
        [4.334230e-01],
        [2.956466e-01],
        [2.185491e-01],
        [4.060458e-01],
        [1.539531e-01],
        [3.900593e-01],
        [3.492380e-01],
        [3.201460e-01],
    ])

    # factor loading matrix
    F = np.array([
        [0,4.513325e-01,0,0,0,0,0,0,0,0,-8.628490e-01,0,0,-2.
↪130824e-01,0,0,0,-9.870088e-01,4.
↪896617e-01,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,-6.034245e-01],
        [-6.983963e-01,0,0,0,0,0,0,0,0,0,0,0,0,-2.609618e-01,-3.
↪376236e-01,0,0,0,-1.018224e+00,0,0,-4.260414e-01,0,0,0,0,0,0,0,0,1.
↪647923e+00,-1.701690e-01,1.186121e+00,0,0,0,0,-7.491083e-01,0,0],
        [0,0,0,8.362688e-01,0,0,0,-1.302310e+00,0,0,9.234870e-01,-2.
↪280104e-01,0,0,0,-1.579673e+00,0,0,0,1.350863e-01,0,0,0,-7.
↪194333e-02,0,0,0,-5.103924e-02,0,1.286124e+00,0,1.403337e+00,5.
↪818706e-01,0,0,0,0,0,0,0],
        [1.649524e+00,0,0,0,0,0,-1.628774e+00,0,0,0,0,0,0,2.098262e+00,0,0,0,-6.
↪024618e-02,0,0,-2.261465e+00,0,0,0,0,-3.065255e-01,0,0,0,0,0,0,0,-9.
↪055355e-01,0,0,0,0,0,0],
        [0,0,0,-2.626506e-01,0,0,0,0,0,0,0,0,0,8.142265e-01,0,0,0,0,0,0,0,4.
↪042442e-01,0,-1.182284e+00,0,0,0,0,-1.249714e+00,3.
↪571231e-01,0,0,0,0,0,0,0,0,0,0],
        [0,0,0,0,2.333656e+00,0,0,1.356463e+00,0,0,0,0,0,-1.472593e-01,0,0,0,-8.
↪897301e-01,1.278745e+00,0,0,0,0,0,0,0,-5.706142e-01,0,0,-1.976974e+00,1.
↪676693e+00,6.995117e-01,0,-3.134126e-02,0,1.440315e-01,0,0,0,0],
        [0,0,-1.664936e+00,0,0,0,0,0,0,0,0,0,0,0,0,1.144839e+00,3.
↪378373e-01,0,1.821861e+00,0,0,0,0,0,1.
↪048065e+00,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
        [0,0,0,0,5.284124e-02,0,5.585971e-02,0,0,0,0,1.805302e+00,0,0,0,0,1.
↪203747e+00,-2.717106e-01,0,0,0,0,0,0,-2.150712e+00,0,0,0,0,0,0,0,0,0,-3.
↪022067e-01,0,0,0,0,0],
        [0,0,0,0,0,0,0,0,0,0,0,0,0,9.337716e-01,0,0,0,0,0,0,6.
↪322444e-01,0,0,0,0,0,0,1.508306e-01,0,0,1.569423e+00,0,1.
↪411575e+00,0,0,0,0,0,0,0],
        [0,0,0,0,-4.240209e-01,0,-1.161277e+00,-9.835142e-01,0,-4.
↪623882e-01,0,0,0,0,0,0,0,0,9.714951e-01,0,0,0,0,0,0,0,0,0,0,0,0,0,0,9.
↪088481e-01,-2.979568e-01,0,0,0,0,0,3.631312e-02,0],
    ])

    return n,k,lam, mu, sigmas, F
```

[264]:
```
def solveProblem4(n, k, lam, mu, sigmas, F):
    Sigma = np.diag(sigmas.flatten() ** 2)
    One = np.ones((n,1))
    row1 = np.concatenate([2 * lam * Sigma, One.copy(), F.T.copy()], axis=1)
    row2 = np.concatenate([One.T, np.zeros((1, 1 + k))],axis=1)
    row3 = np.concatenate([F, np.zeros((k, 1 + k))], axis=1)
    A = np.concatenate([row1, row2, row3])

    y = np.concatenate([mu, [[1]], np.zeros((k,1))])
    x = np.dot(np.linalg.inv(A), y)
    optimalPortfolio = x[:n]
    print("The optimal portfolio is presented below:")
    print(optimalPortfolio.shape)
    # print(optimalPortfolio)
    print("The return associated with this portfolio is r = %s." % (
        np.dot(mu.T, optimalPortfolio)))
    print("The idiosyncratic risk is: %s." % (np.dot(np.dot(optimalPortfolio.T,␣
↪Sigma), optimalPortfolio)))

    # Verify that 1^Tx = 1
    assert np.allclose(np.dot(One.T, optimalPortfolio), 1)
    # Verify R^{fact} = Fx = 0
    assert np.allclose(np.dot(F, optimalPortfolio), 0)
```

[265]:
```
solveProblem4(*getProblem4Inputs())
```

```
The optimal portfolio is presented below:
(40, 1)
The return associated with this portfolio is r = [[26.70627504]].
The idiosyncratic risk is: [[133.62972408]].
```

1.3 Problem 6: Controlling a system using the initial conditions

[ ]:
```
def getProblem6Inputs():
    t = 10
    A = np.array([
        [0, 0, 1, 0],
        [0, 0, 0, 1],
        [-2, 1, 0, 0],
        [1, -1, 0, 0]
    ])
    expA = linalg.expm(t * A)
    E = expA[:, 2:]
    assert E.shape == (4, 2)
    return E
```

[208]:
```
E = getProblem6Inputs()
```

1.3.1 Part (a)

[236]:
```
alpha = E[1,:].T * 1 / np.dot(E[1,:], E[1,:].T) * 2
```

[237]:
```
alpha
```

[237]:
```
array([ 2.46135192, -9.46473528])
```

[238]:
```
np.dot(E[1,:],alpha)
```

[238]:
```
2.0
```

1.3.2 Part (b)

[239]:
```
alpha = np.dot(np.linalg.inv(E[:2,:]), np.array([[1],[2]]))
```

[240]:
```
alpha
```

[240]:
```
array([[ -6.44083425],
       [-11.77979348]])
```

[241]:
```
np.dot(E[:2,:], alpha)
```

[241]:
```
array([[1.],
       [2.]])
```

1.3.3 Part (c)

[242]:
```
q = np.array([[1],[2],[0],[0]])
alpha = np.dot(np.linalg.inv(np.dot(E.T, E)), np.dot(E.T, q))
```

[243]:
```
alpha
```

[243]:
```
array([[-0.13614458],
       [-0.34345832]])
```

[246]:
```
np.dot(E, alpha)
```

[246]:
```
array([[ 0.01627571],
       [ 0.06097153],
       [-0.23927003],
       [-0.27746388]])
```

[247]:
```
# Compute the RMSE
np.linalg.norm(q - np.dot(E, alpha))
```

[247]:
```
2.204944721263555
```

1.3.4 Part (d)

[248]:
```
def getProblem6Inputs2():
    t = 10
    A = np.array([
        [0, 0, 1, 0],
        [0, 0, 0, 1],
        [-2, 1, 0, 0],
        [1.0/1.3, -1.0/1.3, 0, 0]
    ])
    expA = linalg.expm(t * A)
    E = expA[:, 2:]
    assert E.shape == (4, 2)
    return E
```

[250]:
```
Ep = getProblem6Inputs2()
```

[257]:
```
A = np.stack((E[1,:], Ep[1,:]))
alpha = np.dot(np.linalg.inv(A), np.array([[2],[2]]))
```

[258]:
```
alpha
```

[258]:
```
array([[10.78304166],
       [-7.30063808]])
```

[259]:
```
np.dot(A, alpha)
```

[259]:
```
array([[2.],
       [2.]])
```