7.1080. Hovercraft with limited range. We have a hovercraft moving in the plane with two thrusters, each pointing through the center of mass, exerting forces in the $x$ and $y$ directions with 100% efficiency. The hovercraft has mass 1. The discretized equations of motion for the hovercraft are

$$x(t + 1) = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} x(t) + \begin{bmatrix} \frac{1}{2} & 0 \\ 1 & 0 \\ 0 & \frac{1}{2} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix}$$

where $x_1$ and $x_2$ are the position and velocity in the $x$-direction, and $x_3$, $x_4$ are the position and velocity in the $y$-direction. Here

$$u(t) = \begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix}$$

is the force acting on the hovercraft for time in the interval $[t, t+1)$. Let the position of the vehicle at time $t$ be $q(t) \in \mathbb{R}^2$.

a) The hovercraft starts at the origin. We’d like to apply thrust to make it move through points $p_1, p_2, p_3$ at times $t_1, t_2, t_3$, where

$$p_1 = \begin{bmatrix} 1 \\ -\frac{1}{2} \end{bmatrix} \quad p_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad p_3 = \begin{bmatrix} -\frac{3}{2} \\ 0 \end{bmatrix}$$

$$t_1 = 6 \quad t_2 = 40 \quad t_3 = 50$$

We will run the hovercraft on the time interval $[0, 70]$. We’d like to apply a sequence of inputs $u(0), u(1), \ldots, u(70)$ to make the hovercraft position pass through the above sequence of points at the specified times.

We would like to find the sequence of inputs that drives the hovercraft through the desired points which has the minimum cost, given by the sum of the squares of the forces:

$$\sum_{t=0}^{70} \|u(t)\|^2$$

To do this, pick $A_{hov}$ and $y_{des}$ to set this problem up as an equivalent minimum-norm problem, where we would like to find the minimum-norm $u_{seq}$ which satisfies

$$A_{hov} u_{seq} = y_{des}$$

where $u_{seq}$ is the sequence of force inputs

$$u_{seq} = \begin{bmatrix} u(0) \\ u(1) \\ \vdots \\ u(70) \end{bmatrix}$$
Plot the trajectory of the hovercraft using this input, and the way-points $p_1, \ldots, p_3$. Also plot the optimal $u$ against time.

b) Now we would like to compute the trade-off curve between the accuracy with which the mass passes through the waypoints and the norm of the force used. Let our two objective functions be

$$J_1 = \sum_{i=1}^{3} \|q(t_i) - p_i\|^2 = \|A_{hov}u_{seq} - y_{des}\|^2$$

and

$$J_2 = \sum_{t=0}^{70} \|u(t)\|^2$$

By minimizing the weighted sum

$$J_1 + \mu J_2$$

for a range of values of $\mu$, plot the trade-off curve of $J_1$ against $J_2$ showing the achievable performance. This above trade-off curve shows how we can trade-off between how accurately the hovercraft passes through the waypoints and how much input energy is used.

c) For each of the following values of $\mu$

$$\{ \frac{10^p}{\mu} \mid p = -2, 0, 2, \ldots, 10 \}$$

plot the trajectories all on the same plot, together with the waypoints.

d) Now suppose we are controlling the hovercraft by radio control, and the maximum range possible between the transmitter and receiver is 2 (in whatever units we are using for distance.) Notice that, if we use the minimum-norm input then the hovercraft passes out of range, both when making its first turn and on the final stretch (between times 50 and 70).

We’d like to do something about this, but trading off the input norm as above doesn’t do the right thing; if $\mu$ is large then the hovercraft stays within range, but misses the waypoints entirely; if $\mu$ is small then it comes close to the waypoints, but goes out of range. Notice that this is particularly a problem on the final stretch between times 50 and 70; explain why this is.

e) One remedy for this problem is to solve a constrained multiobjective least-squares problem. We would like to impose the constraint that

$$A_{hov}u_{seq} = y_{des}$$

that is, achieve zero waypoint error $J_1 = 0$. We can attempt to keep the hovercraft in range by trading off the sum of the squares of the position

$$J_3 = \sum_{t=0}^{70} \|q(t)\|^2$$
against input cost $J_2$ subject to this constraint. To do this, we’ll solve

$$\begin{align*}
\text{minimize} & \quad J_3 + \gamma J_2 \\
\text{subject to} & \quad A_{\text{hov}} u_{\text{seq}} = y_{\text{des}}
\end{align*}$$

First, find the matrix $W$ so that the cost function is given by

$$J_3 + \gamma J_2 = \|W u_{\text{seq}}\|^2$$

f) Now we have a problem of the form

$$\begin{align*}
\text{minimize} & \quad \|W u\|^2 \\
\text{subject to} & \quad Au = y_{\text{des}}
\end{align*}$$

This is called a **weighted minimum-norm solution**; the only difference from the usual minimum-norm solution to $Au = y_{\text{des}}$ is the presence of the matrix $W$, and when $W = I$ the optimal $u$ is just given by $u_{\text{opt}} = A^\dagger y_{\text{des}}$. Show that the solution for general $W$ is

$$u_{\text{opt}} = \Sigma^{-1} A^T (A \Sigma^{-1} A^T)^{-1} y_{\text{des}}$$

where $\Sigma = W^T W$. (One way to do this is using Lagrange multipliers.) Use this to solve the remaining parts of this problem.

g) For each of the following values of $\gamma$

$$\{ 10^p \mid p = 0, 2, 4, \ldots, 20 \}$$

Plot the trajectories all on the same plot, together with the waypoints. Explain what you see.

h) By trying different values of $\gamma$, you should be able to find a trajectory which just keeps the hovercraft within range. Plot the trajectory of the hovercraft; what is the corresponding value of $\gamma$? Is this the smallest-norm input $u$ that just keeps the hovercraft within range, and drives the hovercraft through the waypoints? Explain why, or why not.

i) For a range of values of $\gamma$, plot the trade-off curve of $J_3$ against $J_2$ showing the achievable performance.

**Solution.**

a) Setting

$$C = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}$$

gives the position of the hovercraft at time $t$ as

$$y(t) = \sum_{\tau=0}^{t-1} C A^{t-1-\tau} Bu(\tau)$$
The parameters for the least-squares problem are therefore

\[
A_{\text{hov}} = \begin{bmatrix}
CA^{t_1-1}B & CA^{t_1-1} & \cdots & CB & 0 & 0 & \cdots & 0 \\
CA^{t_2-1}B & CA^{t_2-2}B & \cdots & 0 \\
CA^{t_3-1}B & CA^{t_3-2}B & \cdots & 0
\end{bmatrix}
\begin{bmatrix}
y_{\text{des}} = \\
p_1 \\
p_2 \\
p_3
\end{bmatrix}
\]

Solving this least squares problem gives optimal trajectory

![Optimal Trajectory](image1.png)

The corresponding optimal input sequence is below.

![Optimal Input Sequence](image2.png)

b) The weighted sum objective is

\[
J_1 + \mu J_2 = \left\| A_{\text{hov}} \sqrt{\mu I} u_{\text{seq}} - \begin{bmatrix} y_{\text{des}} \\ 0 \end{bmatrix} \right\|^2
\]
where

\[ u_{\text{seq}} = \begin{bmatrix} u(0) \\ \vdots \\ u(69) \end{bmatrix} \]

and so the optimal input sequence is given by

\[ u_{\text{seq}} = \begin{bmatrix} A_{\text{way}} \end{bmatrix}^{\dagger} \begin{bmatrix} \sqrt{\mu I} \\ y_{\text{des}} \\ 0 \end{bmatrix} \]

Choosing values of \( \mu \) between 1 and \( 10^7 \) using \texttt{mus=logspace(0,7,50)} , the trade-off curve is shown below.

c) All of the trajectories together are
We can see clearly that increasing $\mu$ reduces the accuracy with which the trajectory passes through the waypoints.

d) On the final stretch the input is zero, and so is unaffected by increasing $\mu$. We were attempting to use the heuristic 'keeping $u$ small keeps $x$ small' but this fails, because when $u = 0$ the hovercraft just keeps going in a straight line.

e) We would like to minimize $J_3 + \gamma J_2$ subject to the constraints that the hovercraft moves through the waypoints. Denote the sequence of positions of the hovercraft by

$$y_{\text{seq}} = \begin{bmatrix} y(0) \\ \vdots \\ y(T) \end{bmatrix}$$

where $T = 70$. Then we have

$$y_{\text{seq}} = T u_{\text{seq}}$$

where $T$ is the Toeplitz matrix

$$T = \begin{bmatrix} 0 & CB & 0 & 0 \\ CB & CAB & CB & 0 \\ \vdots & \vdots & \vdots & \vdots \\ CA^{T-1}B & CA^{T-2}B & \ldots & CB \end{bmatrix}$$

Now the cost function is

$$J_3 + \gamma J_2 = \|Tu_{\text{seq}}\|^2 + \gamma \|u_{\text{seq}}\|^2 = \|Wu_{\text{seq}}\|^2$$
where

\[ W = \begin{bmatrix} T \\ \sqrt{\gamma}I \end{bmatrix} \]

f) We’d like to solve

\[
\begin{align*}
& \text{minimize} \quad \|Wu\|^2 \\
& \text{subject to} \quad Au = y_{des}
\end{align*}
\]

One way to solve this is using Lagrange multipliers; if we augment the cost function by the Lagrange multipliers multiplied by the constraints, we have

\[ L(u, \lambda) = u^T \Sigma u + \lambda^T (Au - y_{des}) \]

and the optimality conditions are

\[
\begin{align*}
\frac{\partial L}{\partial u} &= 2u_{opt}^T \Sigma + \lambda^T A = 0 \\
\frac{\partial L}{\partial \lambda} &= u_{opt}^T A^T - y_{des} = 0
\end{align*}
\]

The first condition gives

\[ u_{opt} = -\frac{1}{2} \Sigma^{-1} A^T \lambda \]

and substituting this into the second we have

\[ -\frac{1}{2} A \Sigma^{-1} A^T \lambda = y_{des} \]

hence

\[ \lambda = -2(A \Sigma^{-1} A^T)^{-1} y_{des} \]

and

\[ u_{opt} = \Sigma^{-1} A^T (A \Sigma^{-1} A^T)^{-1} y_{des} \]

as desired.
g) The trajectory for a range of $\gamma$ values is shown below. (Actually these are clearer on separate plots)

We can see the trade-off clearly; decreasing $\gamma$ causes the hovercraft to try very hard to
stay close to the origin. Also notice the asymmetry caused by the different times at which the hovercraft must be at the waypoints.

h) A good choice of gamma is about $1.7 \times 10^4$. Here the trajectory just remains within range, as shown below.

This is not the smallest-norm $u$ that keeps the hovercraft within range and drives the hovercraft through the waypoints, because we are minimizing the sum of the squares of $||q(t)||$, rather than constraining each $||q(t)||$ independently. You can see this in the plot, since in the final stretch the hovercraft is expending extra effort to stay well within range, and this excessive input could be reduced.

In fact, one can compute the exact optimal, but this is not required and not covered in this course; (an approximation of) it is below.
i) The trade-off is below.

Notice that the vertical asymptote occurs when $J_2 \approx 0.03$; this is the minimum-norm of $u$ which drives the hovercraft through the desired trajectory, as seen in part (b).

code that solves this problem
helper functions

function y=vec(x)
% VEC produces a vector of length m*n from an m by n matrix

10
function y=vec(x)
% Given an m by n matrix x, y=vec(x) constructs a vector y
% consisting of the columns of x stacked on top of each other

[m,n] = size(x);

y = reshape(x,m*n,1);

function T=sys_toeplitz(A,B,C,D,out_times,in_times);
% SYS_TOEPLITZ computes toeplitz matrices for a discrete-time LDS
%
% T=sys_toeplitz(A,B,C,D,out_times,in_times);
%
% A,B,C,D specify a discrete-time state-space realization
%
% out_times and in_times are row vectors
%
% for example, if out_times is [1,2,4] and in_times is [0:10]
% then T is the matrix which maps
%
% [u(0); u(1); ... u(10)] to [y(1); y(2); y(4)]
%
% Notice that T is Toeplitz if out_times and in_times are
% both of the form a:b

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% setup parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% number of states
n=size(A,1);

% num inputs and outputs
ny=size(C,1);
nu=size(B,2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% construct the Toeplitz matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% this is neither efficient nor numerically reliable for big matrices
% but is simple and works well for small cases
T=[];
for r=out_times

    \% now create a row for this output time
    T_row=[];

    for s=in_times
        \% three cases; either CA^tB, D, or 0

            if s<r
                \% below the diagonal
                this_block= C*A^(r-s-1)*B;

            elseif s==r
                \% on the diagonal
                this_block=D;

            else
                \% above the diagonal
                this_block=zeros(ny,nu);

            end

    T_row=[T_row, this_block];
end
T=[T; T_row];
end

main code
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\% compute min-norm input that drives a hovercraft through a
\% given set of waypoints
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\% parameters
\% desired radius
    r_max=2;
\% desired time steps and positions
way_times=[ 6 40 50 ];

way_points=[ 1, 0, -1.5 ;
            -0.5, 1, 0 ];

% final time step
t_max=70;

% sampling time
h=1;

% discrete-time system
A=[1 h 0 0 ;
   0 1 0 0 ;
   0 0 1 h ;
   0 0 0 1 ];

B=[h^2/2 0;
   h 0;
   0 h^2/2 ;
   0 h ];

C=[ 1 0 0 0 ;
   0 0 1 0 ];

D=[ 0 0 ;
   0 0 ];

% number of states
n=size(A,1);

% num inputs and outputs
ny=size(C,1);
nu=size(B,2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% real work is here
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% toeplitz matrix mapping inputs to position at way_times
A_hov=sys_toeplitz(A,B,C,D, way_times, 0:t_max-1);

% find minimum norm input
u_tmp=pinv(A_hov)*vec(way_points);

% for convenience of simulation, reshape the inputs
so that $u_{opt(:,k+1)}$ is the input vector at time $k$

$u_{opt} = \text{reshape}(u_{tmp}, 2, t_{max})$

% simulate
$x = \text{zeros}(n, t_{max}+1)$;
for $k = 0: t_{max} - 1$
    $x(:,k+2) = A*x(:,k+1) + B*u_{opt(:,k+1)}$
end
$y = C*x$

% everything from here on is just plotting

% plot trajectory in the plane

figure(1);
clear;
hold on;
axis equal;
axis([-2.5, 2.5, -2.5, 2.5]);
grid;
box on;

% plot radio range
[xs, ys] = ellipse(r_{max}^2*eye(2), [0; 0]);
plot(xs, ys, 'r');

% plot way points
for $k = 1: \text{size}(\text{way\_times}, 2)$
    plot(way_points(1, k), way_points(2, k), 'ko');
end

% plot trajectory
plot(y(1,:), y(2,:), '.-');

% plot thruster input versus time

figure(2);
clear;
hold on;
grid;

h=plot((0:t_max-1),u_opt(1,1:t_max),’b.-’);
h=plot((0:t_max-1),u_opt(2,1:t_max),’r.-’);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% for a hovercraft
% compute the trade-off curve of input norm to distance norm
% subject to the constraint that
% the trajectory passes through the waypoints
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% parameters

% desired radius
r_max=2;

% desired time steps and positions
way_times=[ 6 40 50 ];

way_points=[ 1, 0, -1.5 ;
              -0.5, 1, 0 ];

n_way_points=size(way_points,2);

% final time step
 t_max=70;

% sampling time
 h=1;

% discrete-time system

A=[1 h 0 0 ;
    0 1 0 0 ;
    0 0 1 h ;
    0 0 0 1 ];

B=[h^2/2 0;
    h 0;
    0 h^2/2 ;
    0 h ];
C = [ 1 0 0 0 ;
     0 0 1 0 ];

D = [ 0 0 ;
     0 0 ];

% number of states
n = size(A,1);

% num inputs and outputs
ny = size(C,1);
nu = size(B,2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% compute trade-off for weighted min-norm problem
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% toeplitz matrix mapping inputs to position at way_times
A_hov = sys_toeplitz(A,B,C,D, way_times, 0:t_max-1);

% toeplitz matrix mapping inputs to sequence of positions
A_pos = sys_toeplitz(A,B,C,D, 0:t_max, 0:t_max-1);

% desired gamma values
gammas = logspace(0,5,40);

% space to save costs
J2 = [];
J3 = [];

for i=1:size(gammas,2)
    this_gamma = gammas(i);

    % weight parameter - sigma inverse
    sig_inv = inv(A_pos'*A_pos + this_gamma*eye(t_max*nu));

    % stack up the way_points
    y_des = reshape(way_points,2*n_way_points,1);

    % compute input that minimizes weighted cost
    lambda = (A_hov*sig_inv*A_hov')\y_des;
    u = sig_inv*A_hov'*lambda;

% compute corresponding trajectory in the plane
y = A_pos*u;

% keep track of achieved costs
J3(i) = norm(y)^2;
J2(i) = norm(u)^2;

% store a nicely shaped y for plotting
y_keep{i} = reshape(y, 2, t_max+1);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% everything from here is just plotting
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% first plot trade off curve

% figure setup
figure(1);
cf;
hold on;
grid;
axis([0,3,0,200]);

% plot
plot(J2, J3, '.-');
xlabel('J_2 position cost');
ylabel('J_3 input cost');
title('trade off curve');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% now plot all the different inputs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% loop over each gamma value
for i=1:size(gammas,2)

    figure(2);
cf;
hold on;
axis equal;
axis([-2.5,2.5,-2.5,2.5]);
grid;
end
Linear dynamical system with constant input. We consider the system \( \dot{x} = Ax + b \), with \( x(t) \in \mathbb{R}^n \). A vector \( x_e \) is an equilibrium point if \( 0 = Ax_e + b \). (This means that the constant trajectory \( x(t) = x_e \) is a solution of \( \dot{x} = Ax + b \).)

a) When is there an equilibrium point?

b) When are there multiple equilibrium points?

c) When is there a unique equilibrium point?

d) Now suppose that \( x_e \) is an equilibrium point. Define \( z(t) = x(t) - x_e \). Show that \( \dot{z} = Az \).

From this, give a general formula for \( x(t) \) (involving \( x_e \), \( \exp(tA) \), \( x(0) \)).

e) Show that if all eigenvalues of \( A \) have negative real part, then there is exactly one equilibrium point \( x_e \), and for any trajectory \( x(t) \), we have \( x(t) \rightarrow x_e \) as \( t \rightarrow \infty \).

Solution.

a) An equilibrium point \( x_e \) exists if and only if \( 0 = Ax_e + b \), i.e., \( -b = Ax_e \). This happens exactly when \( -b \in \text{range}(A) \).

b) If \( x_e \) is any equilibrium point, and \( z \in \text{null}(A) \), then \( x_e + z \) is also an equilibrium point. It follows that in order to have multiple equilibrium points, we need \( \text{null}(A) \neq \{0\} \), as well as \( b \in \text{range}(A) \).

c) For uniqueness, we need that \( \text{null}(A) = \{0\} \), in addition to \( b \in \text{range}(A) \). The nullspace condition implies that \( A \) is nonsingular. But this means that \( \text{range}(A) = \mathbb{R}^n \), so the
condition $b \in \text{range}(A)$ holds automatically. In this case, the unique equilibrium point is $x_e = -A^{-1}b$. In summary: there is a unique equilibrium point if and only if $A$ is nonsingular; in this case, we have $x_e = -A^{-1}b$.

d) $z(t) = \exp(tA)z(0)$, so 
$$x(t) = x_e + \exp(tA)(x(0) - x_e).$$

e) Assume that all eigenvalues of $A$ have negative real part. In particular, no eigenvalue can be zero, which means $A$ is nonsingular. Therefore the unique equilibrium point is $x_e = -A^{-1}b$. Since all eigenvalues of $A$ have negative real part, the matrix $\exp(tA)$ goes to zero as $t \to \infty$. From the formula for $x(t)$ above, we see that $x(t)$ converges to $x_e$.

10.1700. **Optimal espresso cup pre-heating.** At time $t = 0$ boiling water, at 100°C, is poured into an espresso cup; after $P$ seconds (the ‘pre-heating time’), the water is poured out, and espresso, with initial temperature 95°C, is poured in. (You can assume this operation occurs instantaneously.) The espresso is then consumed exactly 15 seconds later (yes, instantaneously). The problem is to choose the pre-heating time $P$ so as to maximize the temperature of the espresso when it is consumed.

We now give the thermal model used. We take the temperature of the liquid in the cup (water or espresso) as one state; for the cup we use an $n$-state finite element model. The vector $x(t) \in \mathbb{R}^{n+1}$ gives the temperature distribution at time $t$: $x_1(t)$ is the liquid (water or espresso) temperature at time $t$, and $x_2(t), \ldots, x_{n+1}(t)$ are the temperatures of the elements in the cup. All of these are in degrees C, with $t$ in seconds. The dynamics are

$$\frac{d}{dt}(x(t) - 20 \cdot 1) = A(x(t) - 20 \cdot 1),$$

where $A \in \mathbb{R}^{(n+1)\times(n+1)}$. (The vector $20 \cdot 1$, with all components 20, represents the ambient temperature.) The initial temperature distribution is

$$x(0) = \begin{bmatrix} 100 \\ 20 \\ \vdots \\ 20 \end{bmatrix}.$$ 

At $t = P$, the liquid temperature changes instantly from whatever value it has, to 95; the other states do not change. Note that the dynamics of the system are the same before and after pre-heating (because we assume that water and espresso behave in the same way, thermally speaking).

We have *very generously* derived the matrix $A$ for you. You will find it in `espressodata.json`. In addition to $A$, the file also defines $n$, and, respectively, the ambient, espresso and preheat water temperatures $T_a$ (which is 20), $T_e$ (95), and $T_1$ (100).

Explain your method, submit your code, and give final answers, which must include the optimal value of $P$ and the resulting optimal espresso temperature when it is consumed. Give both to an accuracy of one decimal place, as in

`'P = 23.5 s, which gives an espresso temperature at consumption of 62.3°C.'`

(This is not the correct answer, of course.)
**Solution.** After $P$ seconds of pre-heating, we will have

$$x(P) - 20 \cdot 1 = e^{PA}(x(0) - 20 \cdot 1).$$

Define a new vector $\tilde{x}(P)$ with $\tilde{x}_i(P) = x_i(P)$ for $i = 2, \ldots, n+1$, and $\tilde{x}_1(P) = 95$. (Thus, $\tilde{x}(P)$ is the state immediately after the water is replaced with espresso.) The temperature distribution at time $P + 15$ will be

$$x(P + 15) - 20 \cdot 1 = e^{15A}(\tilde{x}(P) - 20 \cdot 1).$$

We now have a method for calculating the temperature of the espresso at the instant of consumption for a given $P$:

$$T(P) - 20 = e_1^T x(P + 15) = e_1^T e^{15A}(\tilde{x}(P) - 20 \cdot 1),$$

where $e_1$ is the first unit vector. Thus, we have

$$T(P) = e_1^T e^{15A}(\tilde{x}(P) - 20 \cdot 1) + 20.$$  

To find the optimal value of $P$ we use a simple search method, by calculating $T(P)$ over a finely-sampled range of values of $P$, and selecting the maximum value.

The optimal preheating time for this example is 11.1 seconds. This will give an espresso temperature of 87.6 °C.

Julia code to calculate the answers appears below.

```julia
using LinearAlgebra
include("readclassjson.jl")
data = readclassjson("espressodata.json")
A = data["A"]
n = data["n"]
Ta = data["Ta"]
Te = data["Te"]
Tl = data["Tl"]

Ps = 0:0.01:60

# condition at instance when preheating liquid is added
# work with deviation from ambient, not absolute temperature
x0 = [Tl - Ta; zeros(n)]

T = zeros(length(Ps))
for (i, P) in enumerate(Ps)
    xph = exp(P*A) * x0  # propagate forward by P
    xph[1] = Te - Ta     # instantaneously add espresso
    xf = exp(15*A) * xph # propagate forward by 15
    T[i] = xf[1]         # record final temperature
end
```

20
x, i = findmax(y)
@show Tmax = x + Ta
@show Popt = Ps[i]

The graph below shows how preheat time affects the drinking temperature.

![Graph showing temperature of espresso after 15 seconds, with varying preheat](image1)

The next graph shows the temperature of the espresso over a five-minute period, with and without preheating.

![Graph showing the difference between a cold cup and a warm cup](image2)

11.1900. **Some basic properties of eigenvalues.** Show the following:

a) The eigenvalues of $A$ and $A^T$ are the same.

b) $A$ is invertible if and only if 0 is not an eigenvalue of $A$.

c) If $A$ is invertible, and its eigenvalues are $\lambda_1, \ldots, \lambda_n$, then the eigenvalues of $A^{-1}$ are $1/\lambda_1, \ldots, 1/\lambda_n$.

d) The eigenvalues of $A$ and $T^{-1}AT$ are the same.
e) Suppose $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times m}$. Show that if $\lambda \in \mathbb{C}$ is a nonzero eigenvalue of $AB$, then $\lambda$ is also an eigenvalue of $BA$. Conclude that the nonzero eigenvalues of $AB$ and $BA$ are the same.

You may use the following without proof: $\det A = \det(A^T)$, $\det A = \det(-A)$, $\det(AB) = \det A \det B$, and, if $A$ is invertible, $\det A^{-1} = 1/\det A$.

Solution.

a) Since the determinant is invariant under transposition, the characteristic equation of $A^T$ is

$$X_{A^T}(s) = \det(sI - A^T) = \det((sI - A)^T) = \det(sI - A) = X_A(s).$$

Thus, $A$ and $A^T$ have the same characteristic equation. Because the eigenvalues are the roots of the characteristic equation, this implies that $A$ and $A^T$ have the same eigenvalues.

b) We know that $A$ is invertible if and only if $\text{null}(A) = \{0\}$. We have that 0 is an eigenvalue of $A$ if and only if there exists a nonzero vector $x$ such that $Ax = 0x = 0$. In other words, 0 is an eigenvalue of $A$ if and only if $\text{null}(A) \neq \{0\}$. Thus, $A$ is invertible if and only if 0 is not an eigenvalue of $A$.

c) Since the determinant of a product is the product of the determinants, the determinant of an inverse is the inverse of the determinant, and the determinant is a homogeneous function of degree $n$, we have that

$$X_{A^{-1}}(s) = \det(sI - A^{-1}) = \det(-sA^{-1} \left(\frac{1}{s}I - A\right)) = \frac{1}{\det(A)}(-s)^nX_A\left(\frac{1}{s}\right).$$

We can write $X_A(s)$ as

$$X_A(s) = (s - \lambda_1) \cdots (s - \lambda_n).$$

Then, we have that

$$X_{A^{-1}}(s) = \frac{1}{\det(A)}(-s)^nX_A\left(\frac{1}{s}\right) = \frac{1}{\det(A)}(-s)^n \left(\frac{1}{s} - \lambda_1\right) \cdots \left(\frac{1}{s} - \lambda_n\right) = \frac{1}{\det(A)} \prod_{i=1}^n \frac{\lambda_i}{s - \lambda_i} \left( s - \frac{1}{\lambda_1}\right) \cdots \left( s - \frac{1}{\lambda_n}\right).$$

Finally, because the determinant is the product of the eigenvalues, we have that

$$X_{A^{-1}}(s) = \left( s - \frac{1}{\lambda_1}\right) \cdots \left( s - \frac{1}{\lambda_n}\right).$$

Since the eigenvalues of a matrix are the roots of its characteristic polynomial, this proves that the eigenvalues of $A^{-1}$ are $\frac{1}{\lambda_1}, \ldots, \frac{1}{\lambda_n}$.
d) Since the determinant of a product is the product of the determinants, the determinant of an inverse is the inverse of the determinant, the characteristic polynomial of $T^{-1}AT$ is

\[
X_{T^{-1}AT}(s) = \det(sI - T^{-1}AT) \\
= \det(T^{-1}(sI - AT)) \\
= \frac{1}{\det(T)} \det(sI - A) \det(T) \\
= \det(sI - A) \\
= X_A(s).
\]

Because the eigenvalues are the roots of the characteristic equation, this implies that $A$ and $A^T$ have the same eigenvalues.

**First approach: method one** Suppose $\lambda \in \mathbb{C}$ is a nonzero eigenvalue of $AB$; let $x$ be a corresponding eigenvector. Then, $\tilde{x} = Bx$, we have that

\[(BA)\tilde{x} = BA(Bx) = B((AB)x) = B(\lambda x) = \lambda (Bx) = \lambda \tilde{x}.
\]

Additionally, note that $\tilde{x}$ is nonzero because

\[A\tilde{x} = A(Bx) = (AB)x = \lambda x \neq 0.
\]

Thus, $\lambda$ is also an eigenvalue of $AB$, with corresponding eigenvector $\tilde{x}$. This shows that the set of nonzero eigenvalues of $AB$ is a subset of the set of nonzero eigenvalues of $BA$. By symmetry, the reverse inclusion also holds, which implies that the set of nonzero eigenvalues of $AB$ is equal to the set of nonzero eigenvalues of $BA$.

**Alternative approach: method two** Define the matrices $C_1, C_2, T \in \mathbb{R}^{(m+n) \times (m+n)}$ such that

\[
C_1 = \begin{bmatrix} AB & 0_{m \times n} \\ B & 0_{n \times n} \end{bmatrix}, \quad C_2 = \begin{bmatrix} 0_{m \times m} & 0_{m \times n} \\ B & BA \end{bmatrix} \quad \text{and} \quad T = \begin{bmatrix} I & A \\ 0 & I \end{bmatrix}.
\]

Then, we have that

\[
C_1T = \begin{bmatrix} AB & 0 \\ B & 0 \end{bmatrix} \begin{bmatrix} I & A \\ 0 & I \end{bmatrix} = \begin{bmatrix} AB & ABA \\ B & BA \end{bmatrix} = \begin{bmatrix} I & A \\ 0 & I \end{bmatrix} \begin{bmatrix} 0 & 0 \\ B & BA \end{bmatrix} = TC_2.
\]

Thus, we have that $C_2 = T^{-1}C_1T$, so that $C_1$ and $C_2$ are similar: that is, they have the same characteristic equation

\[X_{C_1}(s) = X_{C_2}(s).
\]

Using the formula for the characteristic equation of a block triangular matrix, we have that

\[
X_{C_1}(s) = X_{AB}(s)X_{0_{n \times n}}(s) = s^n X_{AB}(s), \quad \text{and} \quad X_{C_2}(s) = X_{0_{m \times m}}X_{BA}(s) = s^m X_{BA}(s).
\]

23
Combining these results, we have that
\[ s^n \mathcal{X}_{AB}(s) = s^m \mathcal{X}_{BA}(s). \]

Without loss of generality, assume that \( m \leq n \). Then, we have that
\[ \mathcal{X}_{BA}(s) = s^{n-m} \mathcal{X}_{AB}(s). \]

Recall that the eigenvalues of a matrix are the roots of its characteristic polynomial. Thus, we see that the eigenvalues of \( BA \) are exactly those of \( AB \) (including multiplicities) except that \( AB \) has \( n - m \) additional zero eigenvalues.

13.2030. A method for rapidly driving the state to zero. We consider the discrete-time linear dynamical system
\[ x(t+1) = Ax(t) + Bu(t), \]
where \( A \in \mathbb{R}^{n \times n} \) and \( B \in \mathbb{R}^{n \times k} \), \( k < n \), is full rank. The goal is to choose an input \( u \) that causes \( x(t) \) to converge to zero as \( t \to \infty \). An engineer proposes the following simple method: at time \( t \), choose \( u(t) \) that minimizes \( \|x(t+1)\| \). The engineer argues that this scheme will work well, since the norm of the state is made as small as possible at every step. In this problem you will analyze this scheme.

\[ \text{a) Find an explicit expression for the proposed input } u(t) \text{ in terms of } x(t), A, \text{ and } B. \]

\[ \text{b) Now consider the linear dynamical system } x(t+1) = Ax(t) + Bu(t) \text{ with } u(t) \text{ given by the proposed scheme (i.e., as found in (??)). Show that } x \text{ satisfies an autonomous linear dynamical system equation } x(t+1) = Fx(t). \text{ Express the matrix } F \text{ explicitly in terms of } A \text{ and } B. \]

\[ \text{c) Now consider a specific case:} \]
\[ A = \begin{bmatrix} 0 & 3 \\ 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ 1 \end{bmatrix}. \]

Compare the behavior of \( x(t+1) = Ax(t) \) (i.e., the original system with \( u(t) = 0 \)) and \( x(t+1) = Fx(t) \) (i.e., the original system with \( u(t) \) chosen by the scheme described above) for a few initial conditions. Determine whether each of these systems is stable.

Solution.

\[ \text{a) We should choose } u(t) \text{ such that } \|x(t+1)\| = \|Ax(t) + Bu(t)\| \text{ is minimized. This is simply a least-squares problem in the form } \min_x \|\tilde{y} - \tilde{A} \tilde{x}\| \text{ where } \tilde{y} := Ax(t), \tilde{A} := -B \text{ and } \tilde{x} := u(t). \text{ Therefore the minimizing } u(t) \text{ is} \]
\[ u(t) = \tilde{x}_{ls} = (\tilde{A}^\top \tilde{A})^{-1} \tilde{A}^\top \tilde{y} = -(B^\top B)^{-1} B^\top Ax(t). \]
Figure 1: state trajectory for system \( x(t+1) = Ax(t) \) for \( x(0) = [2 \ 1.5] \) and \( x(0) = [-1 \ 1] \)

b) We have

\[
x(t+1) = Ax(t) + Bu(t) \\
= Ax(t) - B(B^T B)^{-1}B^T Ax(t) \\
= (I - B(B^T B)^{-1}B^T)Ax(t),
\]

and therefore

\[
F = (I - B(B^T B)^{-1}B^T)A.
\]

c) With \( A \) and \( B \) as given

\[
F = \begin{bmatrix} 0 & 1.5 \\ 0 & -1.5 \end{bmatrix}.
\]

The eigenvalues of \( A \) are 0, 0 and as a result \( x(t+1) = Ax \) is stable. However, the eigenvalues of \( F \) are 0, -1.5 and therefore \( F \) is unstable. Thus, this method, though reasonable sounding, not only does not rapidly drive the state to zero — it can actually destabilize a stable system! The state trajectory of the original system \( x(t+1) = Ax(t) \) and for the system \( x(t+1) = Fx(t) \) are shown in Figures ?? and ?? respectively for two initial conditions. Clearly, for the system \( x(t+1) = Ax(t) \), \( x(t) \) goes to zero as \( t \) increases (actually just after two steps), while for the system \( x(t+1) = Fx(t) \), \( \|x(t)\| \) increases as \( t \) increases.
Figure 2: state trajectory for system $x(t+1) = Fx(t)$ for $x(0) = [2 - 0.5]^T$ and $x(0) = [-1 1]^T$