EE 263 Homework 6

Luis A. Perez

Optimal operation of a two-state chemical reactor

Solution:

(a) We have the dynamics equation $\dot{x}(t) = A_j x(t)$. This can be solved directly as the following:

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

If we operate the first reactor for T_0 time followed by the second reactor for $T - T_0$ time, our final state will be:

$$x(T) = e^{(T-T_0)A_2} x(T_1) = e^{(T-T_0)A_2} e^{T_0 A_1} x(0)$$

With the above, we can now write a function $C_k(T_0)$ which will give us the amount of compounds k at time T. We have:

$$C_k(T_0) = e_k^T e^{(T-T_0)A_2} e^{T_0 A_1} x(0)$$

where e_k is the k-th unit vector. Since our answer only needs to be accurate to two decimal places, computing the optimal value of T_0 that maximizes the above can be done through a simple search. We simply compute $C_k(t)$ for all $0 < t \leq T$ at intervals of of size 0.001, and report the value $t_{\max}$ which achieves the maximum C_k.

(b) We perform the method described above. See Figure 1 for plot. We conclude that:

The optimal value is $T_0 = 0.61$ with 0.37 units of compound 2 at time $T = 1$.

(c) We follow a similar set-up as before. In this case, the final state of the system will be given by:

$$x(T) = e^{(T-T_2)A_1} e^{(T_2-T_1)A_2} e^{T_1 A_1} x(0)$$

We can similarly define a function for the k-th chemical as:

$$C_k(T_1, T_2) = e_k^T e^{(T-T_2)A_1} e^{(T_2-T_1)A_2} e^{T_1 A_1} x(0)$$

Since we're once again only interested in a solution accurate up to 2 decimal places, we can just try suitable values of T_1 and T_2 in intervals of size 0.001.

(d) We perform the method described above. See Figure 2 for plot. We conclude that:

The optimal value is $T_1 = 0.49$ and $T_2 = 0.87$ with 0.38 units of compound 2 at time $T = 1$.

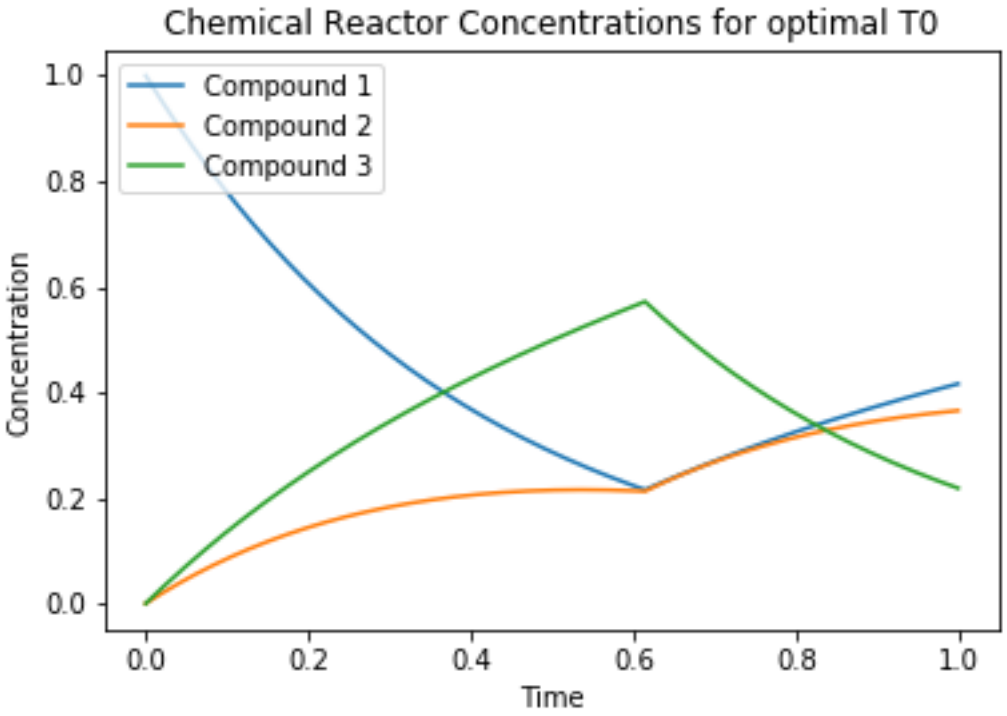


Figure 1: Chemical concentrations for $T_0 = 0.61$ (part b)

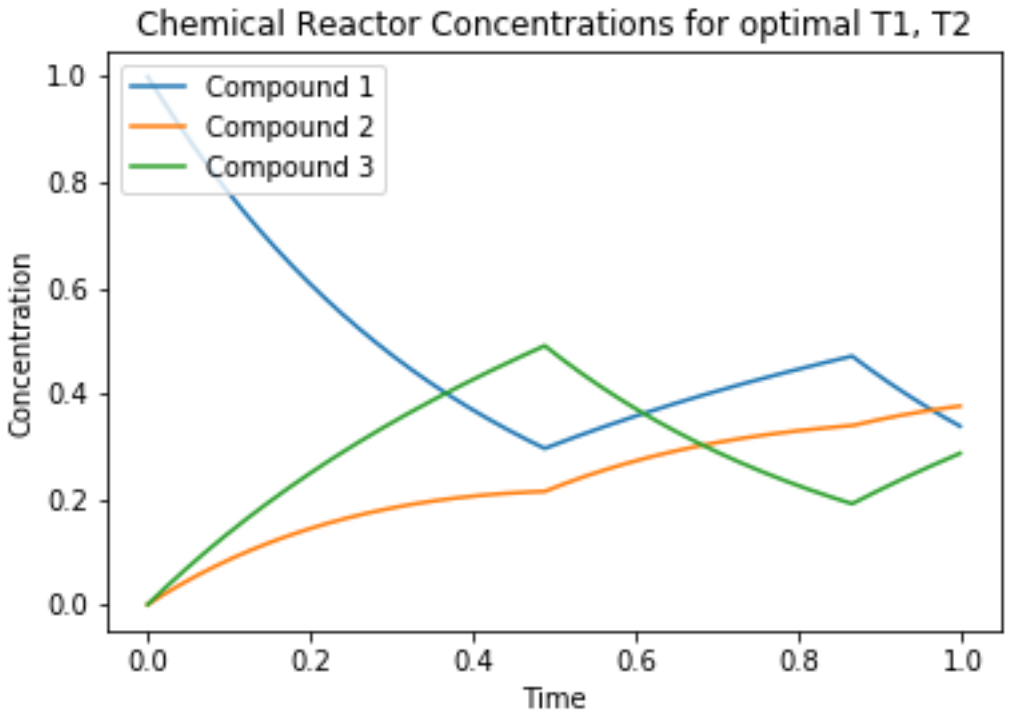


Figure 2: Chemical concentrations for $T_1 = 0.49$ and $T_2 = 0.87$ (part d)

Harmonic Oscillator

Solution:

(a) We find the eigenvalues, resolvent, and state transition matrix for the matrix:

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

The resolvent in our case is given by:

$$(sI - A)^{-1} = \begin{bmatrix} s & -w \\ w & s \end{bmatrix}^{-1}$$
$$\frac{1}{s^2 + w^2} \begin{bmatrix} s & w \\ -w & s \end{bmatrix}$$

We can also derive the eigenvalues by simply solving:

$$\det(A) = s^2 + w^2 = 0$$
$$\implies s = \pm iw \qquad \text{(Where } i = \sqrt{-1}\text{)}$$

From the above, we can compute the state transition matrix as:

$$\Phi = \begin{bmatrix} \cos \omega t & \sin \omega t \\ -\sin \omega t & \cos \omega t \end{bmatrix}$$

Finally, we can express $x(t)$ as:

$$x(t) = \Phi x(0)$$

(b) The vector field is sketched out in Figure 3.

(c) We wish to verify that $||x(t)||$ is constant. Let's do this directly as:

$$\begin{aligned} ||x(t)|| &= \sqrt{x(t)^T x(t)} \\ &= \sqrt{x(0)^T \Phi^T \Phi x(0)} \\ &= \sqrt{x(0)^T \begin{bmatrix} \cos^2 \omega t + \sin^2 \omega t & -\cos \omega t \sin \omega + \sin \omega t \cos \omega t \\ \sin \omega t \cos \omega t - \cos \omega t \sin \omega & \sin^2 \omega t + \cos^2 \omega t \end{bmatrix} x(0)} \\ &= \sqrt{x(0)^T \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} x(0)} \\ &= \sqrt{x(0)^T x(0)} \\ &= ||x(0)|| \end{aligned}$$

From the above, we can conclude that $||x(t)||$ is constant for all t.

(d) We verify directly that the velocity vector is always orthogonal to the position vector. Take $x \in \mathbb{R}^2$ to be some aribitray position. Then we have:

$$\begin{aligned} x^T \dot{x} &= x^T \begin{bmatrix} 0 & \omega \\ -\omega & 0 \end{bmatrix} x && \text{(Given dynamics)} \\ &= \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 0 & \omega \\ -\omega & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\ &= \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} wx_2 \\ -wx_1 \end{bmatrix} \\ &= wx_1x_2 - wx_1x_2 \\ &= 0 \end{aligned}$$

As such, we conclude our proof.

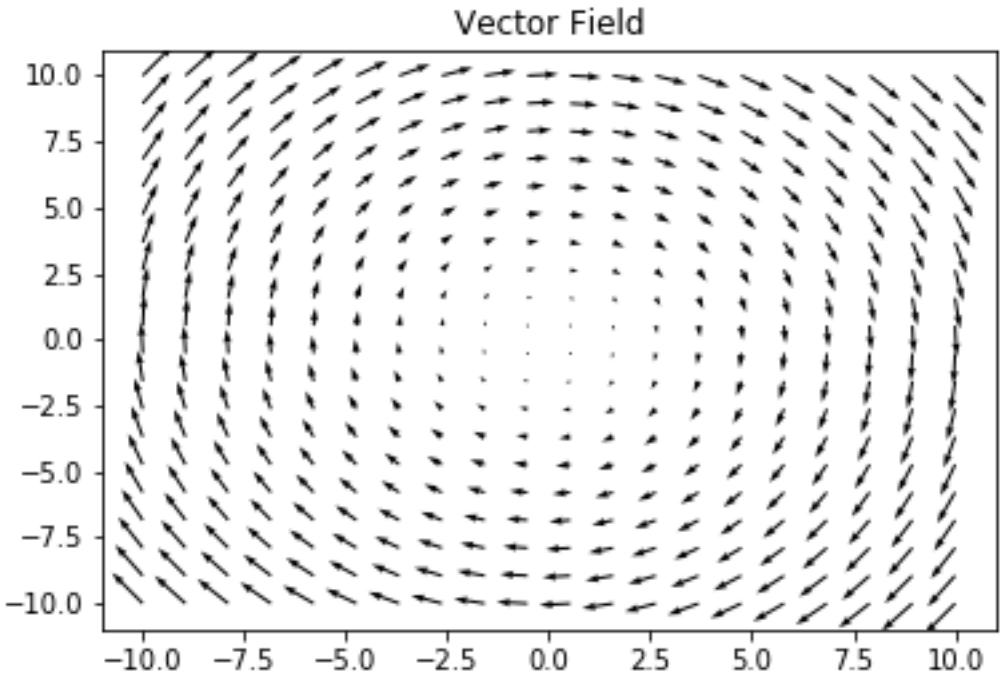


Figure 3: Vector Field Sketch for the Harmonic Oscillator

Interconnection of linear systems

Solution: Our task is to express the overall system (consisting of subsystems S and T) as a single lineary dynamical systen with input, state, and output given by:

$$\hat{u} = \begin{bmatrix} u \\ v \end{bmatrix}, \hat{x} = \begin{bmatrix} x \\ z \end{bmatrix}, \hat{y} = y$$

This is actually relatively straight-forward. First, we need to express each of the equations of each sub-system as a linear combination of only u, v, x, z and y. We begin with the dynamics equations:

$$\begin{aligned}
\dot{x} &= Ax + B_1 u + B_2 w_1 && \text{(Given dynamics of subsystem } S\text{)} \\
&= Ax + B_1 u + B_2 H_1 z && \text{(Using the fact that } w_1 = H_1 z\text{)} \\
\dot{z} &= Fz + G_1 v + G_2 w_2 && \text{(Given dynamics of subsystem } T\text{)} \\
&= Fz + G_1 v + G_2(Cx + D_1 u + D_2 w_1) && \text{(Using given formula for } w_2\text{)} \\
&= Fz + G_1 v + G_2(Cx + D_1 u + D_2 H_1 z) && \text{(Using formula for } w_1\text{)} \\
&= (F + G_2 D_2 H_1)z + G_2 C x + G_1 v + G_2 D_1 u && \text{(Grouping like-terms)}
\end{aligned}$$

Note that we can express the equations above in matrix form as:

$$\begin{bmatrix} \dot{x} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} A & B_2 H_1 \\ G_2 C & F + G_2 D_2 H_1 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} + \begin{bmatrix} B_1 & 0 \\ G_2 D_1 & G_1 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

We can immediately tell from the above our dynamics and input matrix for the single lineary dynamical system.

Next, we takle the output equation. We focus on expression y as a function of x, z, u and v.

$$\begin{aligned}
y &= H_2 z + J w_2 && \text{(Given output equation)} \\
&= H_2 z + J(Cx + D_1 u + D_2 w_1) && \text{(Equation for } w_2\text{)} \\
&= H_2 z + J(Cx + D_1 u + D_2 H_1 z) && \text{(Equation for } w_1\text{)} \\
&= (H_2 + J D_2 H_1)z + JCx + JD_1 u && \text{(Grouping like terms)}
\end{aligned}$$

We can rewrite the above in matrix form as:

$$y = \begin{bmatrix} JC & H_2 + J D_2 H_1 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} + \begin{bmatrix} JD_1 & 0 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

From the above, we can immediately read off our output and feed-through matrices. We did not need to make any assumptions.

However, to be extremely explicit, we have now written our system in the form:

$$\begin{aligned}
\dot{\hat{x}} &= \hat{A}\hat{x} + \hat{B}\hat{u} \\
\hat{y} &= \hat{C}\hat{x} + \hat{D}\hat{u}
\end{aligned}$$

where:

$$\hat{x} = \begin{bmatrix} x \\ z \end{bmatrix}$$

$$\hat{u} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$\hat{y} = y$$

$$\hat{A} = \begin{bmatrix} A & B_2 H_1 \\ G_2 C & F + G_2 D_2 H_1 \end{bmatrix}$$

$$\hat{B} = \begin{bmatrix} B_1 & 0 \\ G_2 D_1 & G_1 \end{bmatrix}$$

$$\hat{C} = \begin{bmatrix} JC & H_2 + J D_2 H_1 \end{bmatrix}$$

$$\hat{D} = \begin{bmatrix} J D_1 & 0 \end{bmatrix}$$

Analysis of investment allocation strategies

Solution:

(a) It turns out that both investment strategies can be described using essentially the same framework. However, for simplicity, we start with the $35 - 35 - 30$ strategy. We define our input state to be:

$$x(t) = \begin{bmatrix} B_1(t) \\ B_2(t) \\ B_3(t) \\ B_2(t-1) \\ B_3(t-1) \\ B_3(t-2) \end{bmatrix} \in \mathbb{R}^6$$

Note that $B_i(t) = 0$ for $t < 0$. Our initial state is therefore:

$$x(0) = \begin{bmatrix} 0.35 \\ 0.35 \\ 0.30 \\ 0 \\ 0 \\ 0 \end{bmatrix} \in \mathbb{R}^6$$

We can immediately compute our total wealth at time t as:

$$y(t) = Cx(t) = w(t) = B_1(t) + B_2(t) + B_3(t) + B_2(t-1) + B_3(t-1) + B_3(t-2)$$

where

$$C = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \in \mathbb{R}^{1 \times 6}$$

For the dynamics matrix, we have:

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

where $A \in \mathbb{R}^{6\times 6}$ is given by:

$$A = \underbrace{\begin{bmatrix}0.35\\0.35\\0.30\\0\\0\\0\end{bmatrix}\begin{bmatrix}1.05 & 0.06 & 0.07 & 1.06 & 0.07 & 1.07\end{bmatrix}}_{\text{Total payout is re-invested by purchasing new bonds}} + \underbrace{\begin{bmatrix}0&0&0&0&0&0\\0&0&0&0&0&0\\0&0&0&0&0&0\\0&1&0&0&0&0\\0&0&1&0&0&0\\0&0&0&0&1&0\end{bmatrix}}_{\text{Update state for one time-step}}$$

$$= \begin{bmatrix}0.3675 & 0.021 & 0.0245 & 0.371 & 0.0245 & 0.3735\\0.3675 & 0.021 & 0.0245 & 0.371 & 0.0245 & 0.3735\\0.315 & 0.018 & 0.021 & 0.318 & 0.021 & 0.321\\0 & 1 & 0 & 0 & 0 & 0\\0 & 0 & 1 & 0 & 0 & 0\\0 & 0 & 0 & 0 & 1 & 0\end{bmatrix}$$

The easiest way to explain is in parts. The first summand when multiplied by $x(t)$ serves to compute the total payout at time t, and then re-distributes this payout by purchasing new bonds at $t+1$, $B_1(t+1), B_2(t+1), B_3(t+1)$. The second summand simply updates our state, since $B_i(t)$ is the "the amount of i-year CDs boughts at period t", and since one-time period is passing, these take the place of $B_i(t-1)$ in our state vector.

With the above complete, we move on to the $60-20-20$ strategy. We use the same framwork as above, with just a few minor modifications. With this strategy, we have:

$$x(0) = \begin{bmatrix}0.6\\0.2\\0.2\\0\\0\\0\end{bmatrix}$$

and we have:

$$A = \underbrace{\begin{bmatrix} 0.60 \\ 0.20 \\ 0.20 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1.05 & 0.06 & 0.07 & 1.06 & 0.07 & 1.07 \end{bmatrix}}_{\text{Total payout is re-invested by purchasing new bonds}} + \underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}}_{\text{Update state for one time-step}}$$

$$= \begin{bmatrix} 0.63 & 0.036 & 0.042 & 0.636 & 0.042 & 0.642 \\ 0.21 & 0.012 & 0.014 & 0.212 & 0.014 & 0.214 \\ 0.21 & 0.012 & 0.014 & 0.212 & 0.014 & 0.214 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

The C matrix remains the same.

(b) We want to understand the limit:

$$\lim_{t \to \infty} \frac{w(t+1)}{w(t)}$$

From the above, we know that:

$$w(t) = Cx(t) = CA^t x(0)$$

As such, we need to understand the limiting behavior of $A^t x(0)$. We recall from lecture the following formulation, for diagonalizable A (we check that this is true for the 35-35-30 strategy):

$$x(t) = A^t x(0) = \sum_{i=1}^{6} \lambda_i^t (w_i^T x(0) v_i) \qquad (A \text{ is diagonalize for 35-35-30 strategy})$$

where λ_i, w_i, v_i are the eignevalues, right, and left eigenvectors. As such, to understand the limiting behavior of $x(t)$ as $t \to \infty$, we need to compute these values for A.

Doing this for the $35 - 35 - 30$ strategy, we have the absolute of the eigenvalues:

$$\begin{aligned} \lambda_1 &= 1.06265258 \\ \lambda_2 &= -0.32657629 + 0.44206583j \\ \lambda_3 &= -0.32657629 - 0.44206583j \\ \lambda_4 &= \lambda_5 = \lambda_6 = 0 \end{aligned}$$

As such, we can see that λ_1 will dominate as $t \to \infty$ (it has the largest absolute value). With this, we can simplify our ratio:

$$\begin{aligned}
\lim_{t\to\infty} \frac{w(t+1)}{w(t)} &= \lim_{t\to\infty} \frac{CA^{t+1}x(0)}{CA^t x(0)} \\
&= \lim_{t\to\infty} \frac{C\sum_{i=1}^{6} \lambda_i^{t+1}(w_i^T x(0) v_i)}{C\sum_{i=1}^{6} \lambda_i^{t}(w_i^T x(0) v_i)} && \text{(Subtituting } A^t x(0)\text{)} \\
&= \lim_{t\to\infty} \frac{\lambda_1^{t+1} C w_1^T x(0) v_1}{\lambda_1^{t} C w_1^T x(0) v_1} && \text{(As } t \to \infty\text{, only } \lambda_1 \text{ survives)} \\
&= \lim_{t\to\infty} \frac{\lambda_1^{t+1}}{\lambda_1^{t}} && \text{(Factor out and cancel values not dependent on } t\text{)} \\
&= \lambda_1
\end{aligned}$$

As such, we have that the wealth ration is simply given by the dominant eigenvalue. In the case of the $35 - 35 - 30$ we have:

$$\lim_{t\to\infty} \frac{w(t+1)}{w(t)} = \lambda_1 = 1.06265258$$

We would like to carry out a similar analysis for the 60-20-20 strategy. However, it turns out that the state matrix for this strategy is not diagonizable. As such, we must use the generilized Jordan Canonical form. In this case, we have:

$$x(t) = A^t x(0) = \sum_{i=1}^{4} T_i J_i^t (S_i^T x(0)) \qquad (A \text{ is not-diagonalize for 60-20-20 strategy})$$

By a similar analysis for the case of the $60-20-20$ strategy, we have the eigenvalues:

$$\begin{aligned}
\lambda_1 &= 1.05978649 \\
\lambda_2 &= -0.20189324 + 0.40145558 \\
\lambda_3 &= -0.20189324 - 0.40145558 \\
\lambda_4 &= \lambda_5 = \lambda_6 = 0
\end{aligned}$$

It turns out that the eigenvectors associated with the 0 eigenvalue only span a space of one-dimension, and as such, this is the reason why the matrix is not diagonizable. However, our limiting analysis doesn't actually change a whole lot, given that λ_1 will

still overpower all other terms. In fact, we have:

$$\begin{aligned}
\lim_{t\to\infty}\frac{w(t+1)}{w(t)} &= \lim_{t\to\infty}\frac{CA^{t+1}x(0)}{CA^{t}x(0)} \\
&= \lim_{t\to\infty}\frac{C(\sum_{i=1}^{3}\lambda_i^{t+1}(w_i^Tx(0)v_i) + T_4J_4^{t+1}(S_4^Tx(0)))}{C(\sum_{i=1}^{3}\lambda_i^{t}(w_i^Tx(0)v_i) + T_4J_4^{t+1}(S_4^Tx(0)))} \\
&\qquad \text{(Subtituting } A^tx(0)) \\
&= \lim_{t\to\infty}\frac{\lambda_1^{t+1}Cw_1^Tx(0)v_1}{\lambda_1^{t}Cw_1^Tx(0)v_1} \\
&\qquad \text{(As } t\to\infty\text{, only } \lambda_1 \text{ survives since } J_4^t = \lambda_4^t\hat{J}_4 = \hat{J}_4) \\
&= \lim_{t\to\infty}\frac{\lambda_1^{t+1}}{\lambda_1^{t}} \qquad \text{(Factor out and cancel values not dependent on } t) \\
&= \lambda_1
\end{aligned}$$

which means that we have:

$$\lim_{t\to\infty}\frac{w(t+1)}{w(t)} = \lambda_1 = 1.05978649$$

(c) We follow a similar strategy as above. We know that for both of our matrices, we have a dominant eignevalue λ_1 with corresponding eigenvectors w_1, v_1. Keeping this

in mind, we have:

$$\begin{aligned}
\lim_{t\to\infty} \frac{B_1(t)+B_2(t-1)+B_3(t-2)}{w(t)} &= \lim_{t\to\infty} \frac{\begin{bmatrix}1 & 0 & 0 & 1 & 0 & 1\end{bmatrix} x(t)}{Cx(t)} \\
&\qquad \text{(Expressing all values as functions of } x(t)\text{)} \\
&= \lim_{t\to\infty} \frac{\begin{bmatrix}1 & 0 & 0 & 1 & 0 & 1\end{bmatrix} A^t x(0)}{CA^t x(0)} \\
&\qquad \text{(Using the fact that } x(t) = A^t x(0)\text{)} \\
&= \lim_{t\to\infty} \frac{\begin{bmatrix}1 & 0 & 0 & 1 & 0 & 1\end{bmatrix} \sum_{i=1}^{6} \lambda_i^t (w_i^T x(0) v_i)}{C \sum_{i=1}^{6} \lambda_i^t (w_i^T x(0) v_i)} \\
&\qquad \text{(Expand into combination of modes)} \\
&= \lim_{t\to\infty} \frac{\begin{bmatrix}1 & 0 & 0 & 1 & 0 & 1\end{bmatrix} \lambda_1^t (w_1^T x(0) v_1)}{C \lambda_1^t (w_1^T x(0) v_1)} \\
&\text{(Only dominant eigenvalue remains as } t \to \infty\text{)} \\
&= \frac{\begin{bmatrix}1 & 0 & 0 & 1 & 0 & 1\end{bmatrix} (w_1^T x(0) v_1)}{C(w_1^T x(0) v_1)} \lim_{t\to\infty} \frac{\lambda^t}{\lambda^t} \\
&\qquad \text{(Factor out constats)} \\
&= \frac{\begin{bmatrix}1 & 0 & 0 & 1 & 0 & 1\end{bmatrix} (w_1^T x(0) v_1)}{C(w_1^T x(0) v_1)} \quad \text{(Limit is 1)} \\
&= \frac{\begin{bmatrix}1 & 0 & 0 & 1 & 0 & 1\end{bmatrix} v_1}{Cv_1} \\
&\qquad \text{(Get rid of scalar } w_1^T x(0)\text{)}
\end{aligned}$$

A very similar argument applies, even for the 60-20-20 matrix which is not diagonizeable since we still have λ_1 as dominant.

By very similar arguments (which we don't repeat for succinctness), we have:

$$\lim_{t\to\infty} L_2(t) = \frac{\begin{bmatrix}0 & 1 & 0 & 0 & 1 & 0\end{bmatrix} v_1}{Cv_1}$$
$$\lim_{t\to\infty} L_3(t) = \frac{\begin{bmatrix}0 & 0 & 1 & 0 & 0 & 0\end{bmatrix} v_1}{Cv_1}$$

As such, we can see that for both strategies, the liquidity ratios do converge as $t \to \infty$ (since both strategy matrices have one dominant eigenvalue λ_1 with $|\lambda_1| > 1$ and for the non-diagonizeable case, the duplicitious eigenvalue is $\lambda_4 = 0$, which means it also dies off). In fact, they converge to the formulas presented above.

Using code to compute the actual results, we have the following. The liquidity ratios for $35 - 35 - 30$ strategy are:

$$L_1 = 0.5033876854426759$$
$$L_2 = 0.33681212927259585$$
$$L_3 = 0.15980018528472825$$

The liquidity ratios for $60 - 20 - 20$ strategy are:

$$L_1 = 0.6215267348660467$$
$$L_2 = 0.24989770184766186$$
$$L_3 = 0.12857556328629144$$

(d) The *initial* investment allocation has no effect on the asymtotic growth rate (as long as there is *some* allocation). In the long-term, the 35-35-30 strategy achieves a steady state defined by by the v_1 eigenvector, regardless of what the initial investment allocation is. Similarly, the liquidity ratios are not affected, and similarly achieve the same ratios in the long-term.

As such, the only possible *initial* investment allocations that we'd considering picking would be given by:

$$x(0) = f(v_1)$$

where f sets the last 3 entries to 0 and normalize the vector to sum to 1. The only difference with this allocation is that we would expect it to achieve the steady-state more quickly, but asymptotically, there is no difference.

Some basic properties of eigenvalues

Solution:

(a) Proving the the eigenvalues of A and A^T are the same boils down to show that their characteristic polynomials, $\chi_A(s)$ and $\chi_{A^T}(s)$ are the same. We have:

$$\begin{aligned}\chi_{A^T}(s) &= \det(sI - A^T) \\ &= \det([sI - A]^T)\end{aligned}$$

($sI - A^T = (sI - A)^T$ since the transpose leaves the diagonals unchanged and A is square)

$$\begin{aligned}&= \det(sI - A) && (\det(X) = \det(X^T) \text{ by the hint provided}) \\ &= \chi_A(s)\end{aligned}$$

As such, we have the the characteristic polynomials of both A and A^T are the same. Therefore, their roots (eigenvalues) must also be the same.

(b) We must prove both directions. For the first direction, suppose A is invertible, but A has a zero eigenvalue. Then this implies that the characteristic polynomial is 0 when evaluated with $s = 0$. More conrectely, we must have:

$$\begin{aligned}\chi_A(0) &= \det(0I - A) \\ &= \det(-A) \\ &= \det(A) \\ &= 0\end{aligned}$$

which implies A is not invertible. This is a contradiction. As such, we must have that if A is invertible, does not have a zero eigenvalue.

For the other direction, suppose A has a zero eignevalue, but it is not invertible. Then we have:

$$\begin{aligned}\det(A) &= \det(-A) \\ &= \det 0I - A \\ &= \chi_A(0) = 0\end{aligned}$$

which means that 0 must be an eigenvalue of A, a contradiction. As such, we conclude that if 0 is not an eigenvalue of A, A must be invertible.

(c) Let us focus on one eigenvalue, λ_i, with corresponding eigenvector v_i. Since A is invertible, we know that $\lambda_i \neq 0$. As such, we have:

$$\begin{aligned}A^{-1}v_i &= A^{-1}\frac{\lambda_i v_i}{\lambda_i} && (\text{Multiplying by 1, we know that } \lambda_i \neq 0) \\ &= \frac{1}{\lambda_i}A^{-1}Av_i && (\text{Using the fact that } \lambda_i v_i = Av_i) \\ &= \frac{1}{\lambda_i}v_i\end{aligned}$$

As such, we have that $\frac{1}{\lambda_i}$ is an eigenvector of A^{-1}.

(d) Similarly to the first sub-problem, this boils down to showing that the characteristic polynomials of both matrices are the same.

$$\begin{aligned}
\chi_{T^{-1}AT}(s) &= \det(sI - T^{-1}AT) \\
&= \det(sT^{-1}T - T^{-1}AT) && (T^{-1}T = I) \\
&= \det(T^{-1}(sI)T - T^{-1}AT) && \text{(Insert identity and move scalar)} \\
&= \det(T^{-1}[sI - A]T) && \text{(Factoring)} \\
&= \det(T{-}1)\det(T)\det(sI - A) && \text{(Properties of determinant)} \\
&= \frac{1}{\det(T)}\det(T)\det(sI - A) && \text{(More properties of determinant)} \\
&= \det(sI - A) \\
&= \chi_A(s)
\end{aligned}$$

As such, we have the the characteristic polynomials of the matrices are the same. Therefore their eigenvalues are the same.

Optimal espresso cup pre-heating

Solution: We beging by developing a model for the temperatures at a given time. We have the matrix $A \in \mathbb{R}^{(n+1)\times(n+1)}$ and initial state $x(0) \in \mathbb{R}^{n+1}$. The dynamics of the system, at all times, are given by:

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

From the above, we immediately have that after P seconds of pre-heating, the state will be:

$$x(P) = e^{PA}(x(0) - 2 \cdot \mathbf{1}) + 20 \cdot \mathbf{20}$$

We define a modified version of $x(P)$, call it $\tilde{x}(P)$, where $x_i(P) = \tilde{x}_i(P)$ for $i \neq 1$, and $\tilde{x}_i(P) = 95$. Then the temporatue distribution at the time of drinking the expresso will be:

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

Then the temperatue of espresso consumptions is given by $T(P) = x_1(P + 15)$. We can solve this by just searching for P directly, sampling enough values of P and computing $T(P)$. With this method, we obtain the following answer:

The optimal value of P is $P = 11.11$s which gives an espresso temperature at consumption of 87.60C.

Real modal form

Solution: Constructing S is somewhat straight-forward. Let us assume that $A \in \mathbb{R}^{n\times n}$ is a diagonalizeable matrix with at least one non-real eigenvalue. We can write $A = T^{-1}\Lambda T$. We summarize a few facts covered in lecture. Let $v_j \in \mathbb{R}^n$ be an eigenvector with corresponding eigenvalue $\lambda_j = a_j + ib_j$ where $b_j \neq 0$. This means that we have:

$$Av_j = \lambda_j v_j$$

Expanding out the above into real and imaginary parts, we have:

$$\begin{aligned} Av_j &= A(\mathcal{R}(v_j) + i\mathcal{I}(v_j)) \\ &= A\mathcal{R}(v_j) + iA\mathcal{I}(v_j) \\ &= (a_j + ib_j)(\mathcal{R}(v_j) + i\mathcal{I}(v_j) \\ &= a_j\mathcal{R}(v_j) - b_j\mathcal{I}(v_j) + i(b_j\mathcal{R}(v_j) + a_j\mathcal{I}(v_j)) \\ \implies A\mathcal{R}(v_j) &= a_j\mathcal{R}(v_j) - b_j\mathcal{I}(v_j) \\ A\mathcal{I}(v_j) &= b_j\mathcal{R}(v_j) + a_j\mathcal{I}(v_j) \end{aligned}$$

The last two equations can be written in matrix form as:

$$A\begin{bmatrix}\mathcal{R}(v_j) & \mathcal{I}(v_j)\end{bmatrix} = \begin{bmatrix}\mathcal{R}(v_j) & \mathcal{I}(v_j)\end{bmatrix}\begin{bmatrix} a_j & b_j \\ -b_j & a_j\end{bmatrix}$$

If we follow a similar process for the conjugate eigenvector, $\bar{v}_j$, which has corresponding eigenvalue $\bar{\lambda}_j$, we'll arrive the at same set of equations as above. First, let us show that $\bar{v}_j$ is also an eigenvector of A. We have:

$$\begin{aligned} A\bar{v}_j &= \bar{Av}_j && (A \text{ is real}) \\ &= \bar{\lambda_j v}_j && (Av_j = \lambda_j v_j) \\ &= \bar{\lambda}_j\bar{v}_j && (\text{Shows } \bar{v}_j \text{ is also eigenvector}) \end{aligned}$$

Following a similar process to above, we have:

$$\begin{aligned} A\bar{v}_j &= A(\mathcal{R}(v_j) - i\mathcal{I}(v_j)) \\ &= A\mathcal{R}(v_j) - iA\mathcal{I}(v_j) \\ &= (a_j - ib_j)(\mathcal{R}(v_j) - i\mathcal{I}(v_j) \\ &= a_j\mathcal{R}(v_j) - b_j\mathcal{I}(v_j) - i(b_j\mathcal{R}(v_j) + a_j\mathcal{I}(v_j)) \\ \implies A\mathcal{R}(v_j) &= a_j\mathcal{R}(v_j) - b_j\mathcal{I}(v_j) \\ A\mathcal{I}(v_j) &= b_j\mathcal{R}(v_j) + a_j\mathcal{I}(v_j) \end{aligned}$$

These are the same set of equations implied by the conjugate.

A similar argument also applies to all conjugate pairs of complex eigenvalues. As such, let take $v_1, \cdots, v_r$ to be the eigenvectors with real eigenvalues. Let us then take $v_{r+1}, \cdots, v_n$ to

the the eigenvectors with complex eigenvalues, ordered such that eigenvectors are followed by their conjugate, if not already present. Then, we can construct the matrix S as follows:

$$S = \begin{bmatrix} v_1 & \cdots & v_r & v_{r+1} & v_{r+3} & \cdots & v_{n-1} \end{bmatrix} \in \mathbb{R}^{n \times n}$$

From the arguments made above, we must have:

$$AS = S\mathbf{diag}\left(\lambda_1, \cdots, \lambda_r, \begin{bmatrix} a_{r+1} & b_{r+1} \\ -b_{r+1} & a_{r+1} \end{bmatrix}, \begin{bmatrix} a_{r+3} & b_{r+3} \\ -b_{r+3} & a_{r+3} \end{bmatrix}, \cdots, \begin{bmatrix} a_{n-1} & b_{n-1} \\ -b_{n-1} & a_{n-1} \end{bmatrix}\right)$$

$$\implies S^{-1}AS = \mathbf{diag}\left(\lambda_1, \cdots, \lambda_r, \begin{bmatrix} a_{r+1} & b_{r+1} \\ -b_{r+1} & a_{r+1} \end{bmatrix}, \begin{bmatrix} a_{r+3} & b_{r+3} \\ -b_{r+3} & a_{r+3} \end{bmatrix}, \cdots, \begin{bmatrix} a_{n-1} & b_{n-1} \\ -b_{n-1} & a_{n-1} \end{bmatrix}\right)$$

We now have the real-modal form we were interested in. For verification, see the attached code.

Jordan form of a block matrix

Solution:

(a) First, we claim claim that the vectors:

$$\hat{v}_i = \begin{bmatrix} v_i \\ 0 \end{bmatrix} \in \mathbb{R}^{2n}$$

Are eigenvectors of C, with eigenvalue λ_i. We can see this directly with:

$$\begin{aligned} C\hat{v}_i &= \begin{bmatrix} A & I \\ 0 & A \end{bmatrix} \begin{bmatrix} v_i \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} Av_i + I0 \\ 0v_i + A0 \end{bmatrix} \\ &= \lambda_i \hat{v}_i \end{aligned}$$

As such, we have n eigenvectors with unique eigevalues. Since each vector spans only 1-dimension, this gives us the following Jordan form for C.

$$J = \begin{bmatrix} J_1 & & \\ & \ddots & \\ & & J_n \end{bmatrix} \in \mathbb{R}^{2n \times 2n}$$

where we have:

$$J_i = \begin{bmatrix} \lambda_i & 1 \\ 0 & \lambda_i \end{bmatrix} \in \mathbb{R}^{2 \times 2}$$

(b) Now that we've found J, we are asked to find the matrix T such that $J = T^{-1}CT$. From lecture, we know that T is given by:

$$T = \begin{bmatrix} \hat{v}_1 & \hat{v}_1' & \hat{v}_2 & \hat{v}_2' & \cdots & \hat{v}_n & \hat{v}_n' \end{bmatrix} \in \mathbb{R}^{2n \times 2n}$$

where we have that:

$$
\begin{aligned}
C\hat{v}_i' &= \hat{v}_i + \lambda_i \hat{v}_i' \\
\implies \hat{v}_i' &= (C - \lambda_i I)^{-1} \hat{v}_i \\
&= \begin{bmatrix} A - I & I \\ 0 & A - I \end{bmatrix}^{-1} \hat{v} \\
&= \begin{bmatrix} (A-I)^{-1} & -(A-I)^{-1}(A-I)^{-1} \\ 0 & (A-I)^{-1} \end{bmatrix} \begin{bmatrix} v_i \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} (A-I)^{-1} v_i \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} -(I + A + A^2 + \cdots) v_i \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} -(1 + \lambda_i + \lambda_i^2 + \cdots) v_i \\ 0 \end{bmatrix} \\
&= \frac{1}{\lambda_i - 1} \hat{v}_i
\end{aligned}
$$

Affine dynamical systems

Solution: We need to define a new lineary dynamical system in terms of $\tilde{x}, u, \tilde{y}$. We have:

$$\begin{aligned}
\dot{\tilde{x}} &= \frac{d}{dt}(x + A^{-1}f) \\
&= \dot{x} && (A \text{ and } f \text{ don't vary with time}) \\
&= Ax + Bu + f && (\text{give}) \\
&= A(\tilde{x} - A^{-1}f) + Bu + f && (\text{Definition of } \tilde{x}) \\
&= A\tilde{x} + Bu
\end{aligned}$$

The above immediately gives us the dynamics of our new system. Doing the same for the output, we have:

$$\begin{aligned}
\tilde{y} &= y - g + CA^{-1}f && (\text{Definition of } \tilde{y}) \\
&= Cx + Du + g - g + CA^{-1}f && (\text{Definition of } y) \\
&= C(x + A^{-1}f) + Du && (\text{Groping like terms}) \\
&= C\tilde{x} + Du && (\text{Definition of } \tilde{x})
\end{aligned}$$

And so, we have our output equations.

HW6

August 3, 2019

1 HW 6

Author: Luis Perez
Last Updated: Aug. 2nd, 2019

[11]:
```
import sys
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Circle
from scipy import linalg
import seaborn as sns
```

1.1 Problem 1: Optimal operation of a two-state chemical reactor

1.1.1 Part (b)

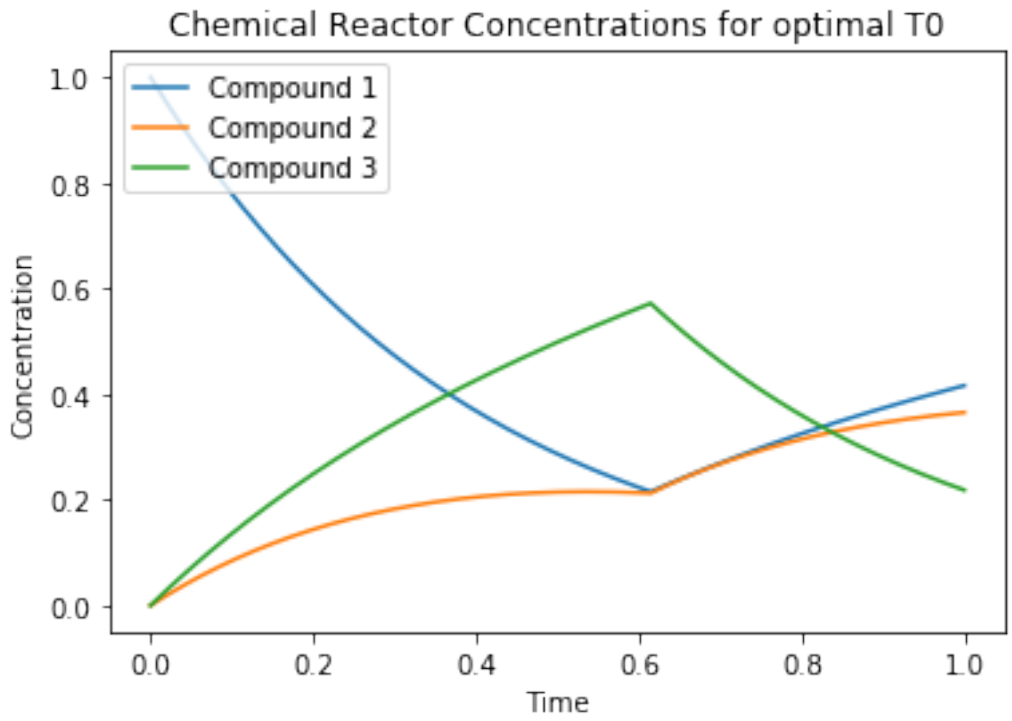
[112]:
```
def loadProblem1Data():
    n = 3
    T = 1
    k = 2
    A1 = 5 * np.array([
        [-0.5, +0.0, +0.0],
        [+0.2, +0.0, -0.1],
        [+0.3, +0.0, +0.1]
    ])
    A2 = 5 * np.array([
        [+0.0, +0.1, +0.2],
        [+0.0, -0.1, +0.3],
        [+0.0, +0.0, -0.5]
    ])
    x0 = np.array([[1], [0], [0]])

    return A1, A2, x0, n, T, k
```

[139]:
```
def solveProblem1b():
    A1, A2, x0, n, T, k = loadProblem1Data()
    kIndex = k - 1
    T0s = np.arange(0, T, step=0.001)
```

```
    Cks = []
    for T0 in T0s:
        state = np.dot(linalg.expm((T - T0) * A2), np.dot(linalg.expm(T0 * A1),␣
↪x0))
        Ck = state[kIndex]
        Cks.append(Ck)
    maxIndex = np.argmax(Cks)
    optimalT0 = T0s[maxIndex]
    maxChemical = Cks[maxIndex]
    print("The optimal value is $T_0 = %.2f$ with $%.2f$ units of compound %d␣
↪at time $T = %d$." % (optimalT0, maxChemical,k,T))

    # Use optimal T0 to to plot results.
    states = []
    for t in np.arange(0, T, step=0.001):
        state = np.dot(linalg.expm(max((t - optimalT0),0) * A2), np.dot(linalg.
↪expm(min(t,optimalT0) * A1), x0))
        states.append(state)
    series = np.concatenate(states, axis=1)
    for i in range(series.shape[0]):
        plt.plot(T0s, series[i, :], label='Compound %s' %(i + 1))
    plt.legend(loc='upper left')
    plt.title("Chemical Reactor Concentrations for optimal T0")
    plt.xlabel('Time')
    plt.ylabel('Concentration')
    plt.savefig('chemical_reactor_b')
```

[140]:
```
solveProblem1b()
```

```
The optimal value is $T_0 = 0.61$ with $0.37$ units of compound 2 at time $T =
1$.
```

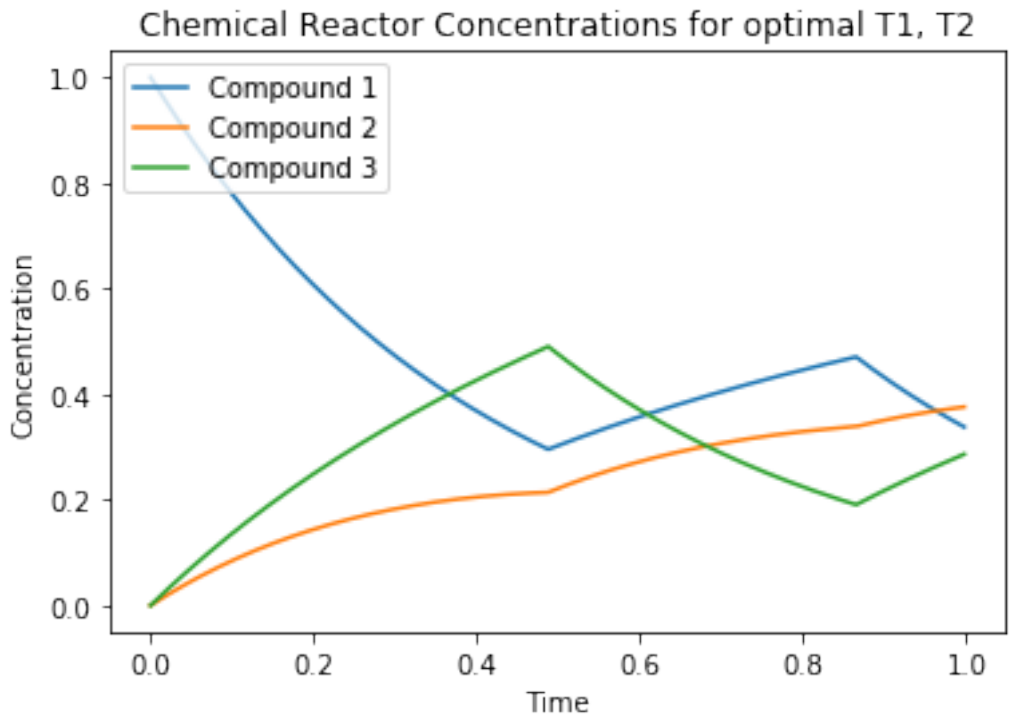


[150]:
```
def solveProblem1d():
    A1, A2, x0, n, T, k = loadProblem1Data()
    kIndex = k - 1

    T1s = np.arange(0, T, step=0.0025)
    Cks = []
    i = 0
    for T1 in T1s:
        T2s = np.arange(T1, T, step=0.0025)
        for T2 in T2s:
            state = np.dot(linalg.expm((T - T2) * A1), np.dot(linalg.
↪expm((T2-T1) * A2), np.dot(linalg.expm((T1 * A1)),x0)))
            Ck = state[kIndex]
            Cks.append((Ck, (T1, T2)))
        i += 1
        if i % 30 == 0:
            print("Finished outer loop. %s%% done" % (T1 * 100))
    maxChemical, (optT1, optT2) = max(Cks)
    print("The optimal value is $T_1 = %.2f$ and $T_2 = %.2f$ with $%.2f$ units␣
↪of compound %d at time $T = %d$." % (optT1, optT2, maxChemical,k,T))

    # Use optimal T1, T2 to to plot results.
    states = []
    for t in np.arange(0, T, step=0.0025):
```

```
        state = np.dot(linalg.expm(max((t - optT2),0) * A1), np.dot(linalg.
↪expm(min(optT2-optT1, max(t - optT1, 0)) * A2), np.dot(linalg.
↪expm((min(optT1, t) * A1)),x0)))
        states.append(state)

    # Each row is the timeseries for a chemical compounds.
    results = np.concatenate(states, axis=1)
    for i in range(results.shape[0]):
        plt.plot(T1s, results[i, :].flatten(), label='Compound %s' %(i + 1))
    plt.legend(loc='upper left')
    plt.title("Chemical Reactor Concentrations for optimal T1, T2")
    plt.xlabel('Time')
    plt.ylabel('Concentration')
    plt.savefig('chemical_reactor_d')
```

[151]:
```
solveProblem1d()
```

```
Finished outer loop. 7.249999999999999% done
Finished outer loop. 14.75% done
Finished outer loop. 22.25% done
Finished outer loop. 29.75% done
Finished outer loop. 37.25% done
Finished outer loop. 44.75% done
Finished outer loop. 52.25% done
Finished outer loop. 59.75% done
Finished outer loop. 67.25% done
Finished outer loop. 74.75% done
Finished outer loop. 82.25% done
Finished outer loop. 89.75% done
Finished outer loop. 97.25% done
The optimal value is $T_1 = 0.49$ and $T_2 = 0.87$ with $0.38$ units of compound
2 at time $T = 1$.
```



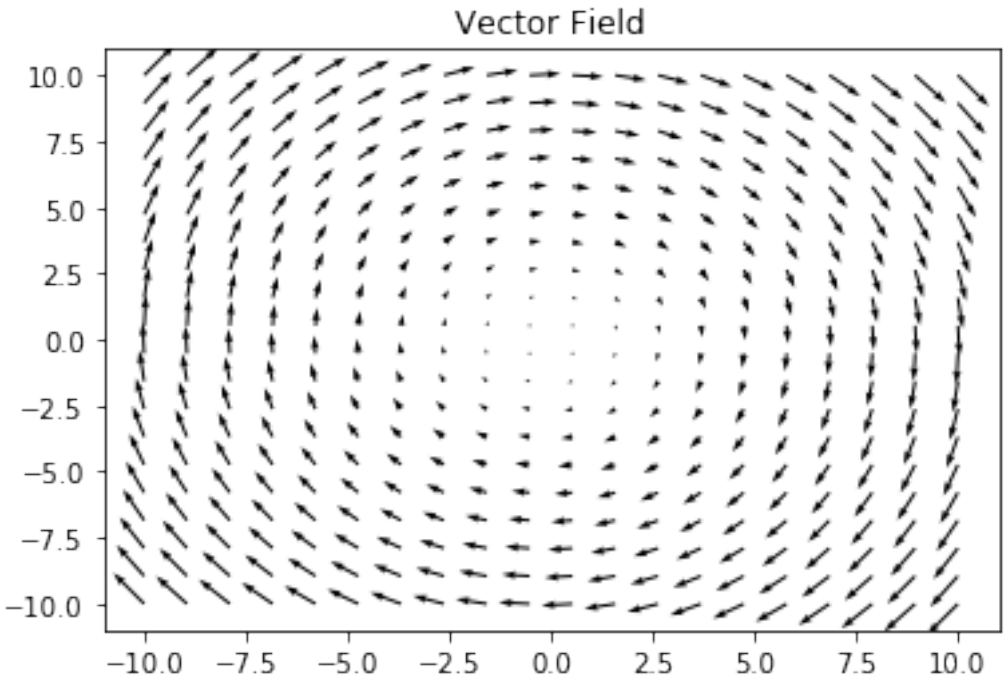
1.2 Problem 2

1.2.1 Part (b)

[25]:
```
def vectorAtPoint(X,Y, w=0.1):
    """Returns the \dot{x}"""
    A = np.array([
        [0, w],
        [-w, 0]
    ])
    dX = np.zeros(X.shape)
    dY = np.zeros(Y.shape)
    n,m = X.shape
    for i in range(n):
        for j in range(m):
            x = X[i,j]
            y = Y[i,j]
            dx, dy = np.dot(A, [x,y])
            dX[i,j] = dx
            dY[i,j] = dy
    return dX, dY

# Grid of x, y points
nx, ny = 20, 20
```

```
x = np.linspace(-10, 10, nx)
y = np.linspace(-10, 10, ny)
X, Y = np.meshgrid(x, y)
dX, dY = vectorAtPoint(X,Y)
```

[32]:
```
ax = plt.quiver(X,Y, dX, dY)
plt.title("Vector Field")
plt.savefig("vector_field_example")
```



1.3 Problem 4: Analysis of investment allocation strategies

1.3.1 Part (b)

[107]:
```
def getProblem4Matrices():
    stateChange = np.array([
        [0] * 6,
        [0] * 6,
        [0] * 6,
        [0, 1, 0, 0, 0, 0],
        [0, 0, 1, 0, 0, 0],
        [0, 0, 0, 0, 1, 0],
    ])
    payoff = np.array([
        [1.05, 0.06, 0.07, 1.06, 0.07, 1.07]
    ])
```

```
    y1 = np.array([
        [0.35],
        [0.35],
        [0.3],
        [0],
        [0],
        [0]
    ])
    y2 = np.array([
        [0.6],
        [0.2],
        [0.2],
        [0],
        [0],
        [0]
    ])
    A1 = np.dot(y1, payoff) + stateChange
    A2 = np.dot(y2, payoff) + stateChange
    return A1, A2, y1, y2
def solveProblem4():
    # A1 is 35-35-30, A2 is 60-20-20.
    A1, A2, x01, x02 =  getProblem4Matrices()
    # get eigenvalues of A1.
    vals1, left1, right1 = linalg.eig(A1, left=True, right=True)
    # Assert we're getting what we think.
    assert np.linalg.matrix_rank(left1) == 6
    vals2, left2, right2 = linalg.eig(A2, left=True, right=True)
    # Turns out that this one is not diaginazeble.
    # However, the eignevalues with multiplicity are 0, so
    # they end up dying away in the limit regardless, so it's safe
    # to ignore. See homework for more details.
    assert np.linalg.matrix_rank(left2) == 4
    with np.printoptions(formatter={'complexfloat': '{: .8f}'.format}):
        print("Eigenvalues for 35-35-30")
        print(vals1)
        print("Eignevalue for 60-20-20")
        print(vals2)

    print("The wealth ratio for 35-35-30 strategy is %0.8f" % (np.
↪real(vals1[0])))
    print("The wealth ratio for 60-20-20 strategy is %0.8f" % (np.
↪real(vals2[0])))

    # Now compute the liquidity ratios.
    prod1 = np.dot(left1[:,0].T, x01) * right1[:,0]
    prod2 = np.dot(left2[:,0].T, x02) * right2[:,0]
```

```
    for i, state in enumerate([prod1, prod2]):
        den = np.sum(state)
        L1 =  (state[0] + state[3] + state[5]) / den
        L2 = (state[1] + state[4]) / den
        L3 = state[2] / den
        print("The liquidity ratios for %s strategy are:" % (
            "35-35-30" if i % 2 == 0 else "60-20-20"))
        print("""
\\begin{align*}
    L_1 &= %s \\\
    L_2 &= %s \\\\
    L_3 &= %s
\\end{align*}
        """ % (L1, L2, L3))
    print("Better initial allocation for 35-35-30 is %s" % (
        right1[:3,0]/ np.linalg.norm(right1[:3,0])))
    print("Better initial allocation for 60-20-20 is %s" % (
        right2[:3,0]/ np.linalg.norm(right2[:3,1])))
```

[108]:
```
solveProblem4()
```

```
4
8.132241926763072
Eigenvalues for 35-35-30
[ 1.06265258+0.00000000j -0.32657629+0.44206583j -0.32657629-0.44206583j
  0.00000000+0.00000000j -0.00000000+0.00000000j -0.00000000-0.00000000j]
Eignevalue for 60-20-20
[ 1.05978649+0.00000000j -0.20189324+0.40145558j -0.20189324-0.40145558j
  0.00000000+0.00000000j  0.00000000+0.00000000j  0.00000000+0.00000000j]
The wealth ratio for 35-35-30 strategy is 1.06265258
The wealth ratio for 60-20-20 strategy is 1.05978649
The liquidity ratios for 35-35-30 strategy are:

\begin{align*}
    L_1 &= (0.5033876854426759-0j) \    L_2 &= (0.33681212927259585-0j) \\
    L_3 &= (0.15980018528472825-0j)
\end{align*}

The liquidity ratios for 60-20-20 strategy are:

\begin{align*}
    L_1 &= (0.6215267348660467+0j) \    L_2 &= (0.24989770184766186+0j) \\
    L_3 &= (0.12857556328629144+0j)
\end{align*}

Better initial allocation for 35-35-30 is [-0.6047079 +0.j -0.6047079 +0.j
```

```
-0.51832106+0.j]
Better initial allocation for 60-20-20 is [1.65481392+0.j 0.55160464+0.j
0.55160464+0.j]
```

1.4 Problem 6: Optimal espresso cup pre-heating

[65]:
```
def solveProblem6():
    # data for espresso problem.
    n = 10
    # ambient temperature.
    Ta = 20
    # temperature of preheat liquid.
    Tl = 100
    # temperature of espresso.
    Te = 95
    A = np.array([
     [-1.00,   1.00,   0.00,   0.00,   0.00,   0.00,   0.00,   0.00,   0.00,
↪0.00,   0.00],
     [33.33, -44.44,  11.11,   0.00,   0.00,   0.00,   0.00,   0.00,   0.00,
↪0.00,   0.00],
      [0.00,  11.11, -22.22,  11.11,   0.00,   0.00,   0.00,   0.00,   0.00,
↪0.00,   0.00],
      [0.00,   0.00,  11.11, -22.22,  11.11,   0.00,   0.00,   0.00,   0.00,
↪0.00,   0.00],
      [0.00,   0.00,   0.00,  11.11, -22.22,  11.11,   0.00,   0.00,   0.00,
↪0.00,   0.00],
      [0.00,   0.00,   0.00,   0.00,  11.11, -22.22,  11.11,   0.00,   0.00,  0.
↪00,  0.00],
      [0.00,   0.00,   0.00,   0.00,   0.00,  11.11, -22.22,  11.11,   0.00,
↪0.00,   0.00],
      [0.00,   0.00,   0.00,   0.00,   0.00,   0.00,  11.11, -22.22,  11.11,
↪0.00,   0.00],
      [0.00,   0.00,   0.00,   0.00,   0.00,   0.00,   0.00,  11.11, -22.22,
↪11.11,   0.00],
      [0.00,   0.00,   0.00,   0.00,   0.00,   0.00,   0.00,   0.00,  11.11,
↪-22.22,  11.11],
      [0.00,   0.00,   0.00,   0.00,   0.00,   0.00,   0.00,   0.00,   0.00,
↪11.11, -11.31]
    ])
    Ps = np.linspace(0, 60, 1000)
    expPs = [linalg.expm(p * A) for p in Ps]
    x0 = np.array([[Tl]] + [[Ta] for _ in range(n)])
    Ts = []
    for P in Ps:
        xP = np.dot(linalg.expm(P * A), x0 - Ta) + Ta
        xhatP = xP
```

```
        xhatP[0] = Te
        final = np.dot(linalg.expm(15 * A), xhatP - Ta) + Ta
        Ts.append(final[0])

    # Find the index of the maximu.
    maxIndex = np.argmax(Ts)
    optimalExpressoTemp = Ts[maxIndex]
    optimalP = Ps[maxIndex]
    print("The optimal value of $P$ is $P = %.2f$s which gives an espresso␣
↪temperature at consumption of $%.2f$C." % (optimalP, optimalExpressoTemp))
```

[151]:
```
solveProblem6()
```

```
The optimal value of $P$ is $P = 11.11$s which gives an espresso temperature at
consumption of $87.60$C.
```

1.5 Problem 7: Real modal form.

[114]:
```
def drawMatrixWithComplexEigenvalues(n=10):
    while True:
        A = np.random.normal(size=(n, n))
        Lam, T = np.linalg.eig(A)
        if np.any(np.iscomplex(T)):
            return A, Lam, T
def solveProblem7():
    n = 10
    A, Lam, T = drawMatrixWithComplexEigenvalues(n)
    print("The generated A matrix is:")
    with np.printoptions(formatter={'float': '{: 0.3f}'.format}):
        print(A)

    # Complex eigvalues always come in pairs.
    complexMask = np.iscomplex(Lam)
    complexVectors = T[:, complexMask]
    assert complexVectors.shape[1] % 2 == 0
    complexVectors = complexVectors[:, ::2]
    realVectors = np.real(T[:, ~complexMask])
    r = realVectors.shape[1]
    assert r + 2*complexVectors.shape[1] == n
    realPart = np.real(complexVectors)
    imgPart = np.imag(complexVectors)
    S = np.zeros((n, n))
    S[:, :r] = realVectors
    S[:, r::2] = realPart
    S[:, (r+1)::2] = imgPart

    print("The computed S matrix is:")
    with np.printoptions(formatter={'float': '{: 0.3f}'.format}):
```

```
        print(S)

    modal = np.dot(np.linalg.inv(S), np.dot(A, S))
    print("The modal form is:")
    with np.printoptions(formatter={'float': '{: 0.3f}'.format}):
        print(modal)
```

[115]:
```
solveProblem7()
```

```
The generated A matrix is:
[[-0.209  0.639  0.770 -0.869 -0.582  0.167  1.028  0.498 -0.218 -0.855]
 [ 0.142 -0.248  1.388 -1.709 -0.387 -0.611  0.056 -0.744 -1.097  1.216]
 [ 1.374 -1.703 -0.722 -1.689  0.050 -1.148 -0.692  0.376 -1.407  0.833]
 [-1.162  0.175  0.770  1.015 -0.059  0.310 -1.232 -1.083 -1.458  0.270]
 [ 0.357  1.636  0.781  1.166 -0.860  0.501 -0.231  1.236  0.508 -0.820]
 [-0.660  2.446  0.262  0.818  0.445 -2.100  2.153  0.185  0.460 -2.443]
 [ 0.647 -0.648  0.316 -2.137  1.702  0.906 -1.331 -0.091  0.418  1.338]
 [-0.126 -1.351  0.578 -0.579 -1.317 -0.232 -0.932 -1.458  0.016  1.801]
 [-0.739 -1.629  0.564  0.511 -0.439  0.083  0.999 -0.324 -1.941 -0.155]
 [ 0.103 -0.433  2.194  0.675  1.089 -0.012 -1.236  0.276 -0.552 -0.855]]
The computed S matrix is:
[[ 0.527  0.193  0.680 -0.579  0.075  0.187 -0.162 -0.120  0.301  0.141]
 [ 0.327  0.057  0.003 -0.168 -0.149  0.251 -0.054 -0.427  0.084 -0.012]
 [-0.008 -0.251  0.050  0.075 -0.411 -0.151  0.317 -0.081 -0.197  0.141]
 [-0.341  0.332  0.410  0.089 -0.082  0.063 -0.013  0.064  0.369  0.037]
 [ 0.031  0.135  0.084  0.028  0.461  0.000 -0.025 -0.237  0.091 -0.033]
 [ 0.369 -0.624 -0.041 -0.338  0.008  0.314 -0.341 -0.138 -0.411  0.138]
 [ 0.288  0.440  0.324 -0.104 -0.082 -0.321 -0.120 -0.258  0.366 -0.069]
 [-0.409  0.146 -0.411  0.579  0.022  0.227  0.445  0.000  0.045 -0.219]
 [-0.178 -0.021  0.104  0.169 -0.362  0.258  0.107  0.172  0.097  0.163]
 [-0.283  0.404  0.271  0.365 -0.002  0.035  0.298 -0.247  0.514  0.000]]
The modal form is:
[[ 1.532 -0.000  0.000 -0.000  0.000 -0.000 -0.000  0.000  0.000  0.000]
 [ 0.000 -2.509 -0.000 -0.000 -0.000 -0.000 -0.000  0.000 -0.000  0.000]
 [ 0.000 -0.000 -0.940  0.000 -0.000  0.000  0.000 -0.000 -0.000 -0.000]
 [-0.000 -0.000 -0.000  0.426  0.000 -0.000  0.000  0.000 -0.000 -0.000]
 [-0.000 -0.000 -0.000  0.000 -2.523  2.273 -0.000 -0.000  0.000 -0.000]
 [ 0.000 -0.000 -0.000  0.000 -2.273 -2.523  0.000  0.000 -0.000  0.000]
 [-0.000  0.000  0.000 -0.000 -0.000  0.000  0.893  1.463  0.000  0.000]
 [ 0.000  0.000 -0.000 -0.000  0.000  0.000 -1.463  0.893  0.000 -0.000]
 [-0.000  0.000  0.000  0.000  0.000  0.000 -0.000 -0.000 -1.980  0.485]
 [-0.000 -0.000  0.000  0.000  0.000 -0.000 -0.000  0.000 -0.485 -1.980]]
```