Pr. 1. (sol/hs105)
Since $\boldsymbol{P}$ is a projection matrix, $\boldsymbol{P}^k = \boldsymbol{P}$. Therefore

$$\begin{aligned} e^{\boldsymbol{P}} &= \boldsymbol{P}^0 + \frac{\boldsymbol{P}^1}{1!} + \frac{\boldsymbol{P}^2}{2!} + \ldots + \frac{\boldsymbol{P}^k}{k!} + \ldots = \boldsymbol{I} + \frac{\boldsymbol{P}}{1!} + \frac{\boldsymbol{P}}{2!} + \ldots + \frac{\boldsymbol{P}}{k!} + \ldots \\ &= \boldsymbol{I} + \boldsymbol{P}\left(\frac{1}{1!} + \frac{1}{2!} + \ldots + \frac{1}{k!} + \ldots\right) = \boldsymbol{I} + \boldsymbol{P}(e-1). \end{aligned}$$

Pr. 2. (sol/hs021)
The transition probability matrix is

$$\boldsymbol{P} = \left[\begin{array}{l|ccc} & \text{Cheese} & \text{Grapes} & \text{Lettuce} \\ \hline \text{Cheese} & 0 & 4/10 & 6/10 \\ \text{Grapes} & 1/2 & 1/10 & 4/10 \\ \text{Lettuce} & 1/2 & 5/10 & 0 \end{array}\right],$$

where P_{ij} is the probability of going from state j to state i.
Let $\boldsymbol{\pi} = \begin{bmatrix} \pi_1 \\ \pi_2 \\ \pi_3 \end{bmatrix}$ denote the equilibrium distribution of the states (Cheese, Grapes, Lettuce). At equilibirum, $\boldsymbol{P\pi} = \boldsymbol{\pi}$, so $\boldsymbol{\pi}$ is the eigenvector corresponding to the largest eigenvalue of $\boldsymbol{P}$ $(= 1)$.
Note that $\boldsymbol{P1} = \mathbf{1}$, where $\mathbf{1}$ is the all-ones column vector. Thus $\mathbf{1}$ is an eigenvector of $\boldsymbol{P}$ associated with the eigenvalue 1. The Perron-Frobenius theorem says that this will be the only eigenvalue of $\boldsymbol{P}$ equal to one, so $\mathbf{1}$ is the equilibrium eigenvector. Upon normalizing so that $\mathbf{1}'\boldsymbol{\pi} = 1$, we see that the equilibrium distribution is $\boldsymbol{\pi} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$.

Pr. 3. (sol/hs011)

(a) Let $\boldsymbol{A}$ and $\boldsymbol{B}$ denote companion matrices associated with degree n polynomial $p_1(x) = \alpha_0 + \alpha_1 x + \ldots + \alpha_{n-1} x^{n-1} + x^n$ and degree m polynomial $p_2(x) = \beta_0 + \beta_1 x + \ldots + \beta_{m-1} x^{m-1} + x^m$, respectively.

Consider the matrix:

$$\boldsymbol{C} = \boldsymbol{A} \otimes \boldsymbol{I}_m + \boldsymbol{I}_n \otimes (-\boldsymbol{B}) = \boldsymbol{A} \otimes \boldsymbol{I}_m - \boldsymbol{I}_n \otimes \boldsymbol{B}.$$

By Theorem 13.16 of Laub, the eigenvalues of $\boldsymbol{C}$ will equal $\lambda_i(\boldsymbol{A}) - \lambda_j(\boldsymbol{B})$, so that if the polynomials corresponding to $\boldsymbol{A}$ and $\boldsymbol{B}$ have common roots, then at least one of the eigenvalues of $\boldsymbol{C}$ will be identically zero. Thus if $\det(\boldsymbol{C}) = 0$, then we can declare that the polynomials associated with the companion matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ have common roots.

For randomly chosen Gaussian polynomial coefficients, there is an exceedingly small probability of having common roots.

(b) A possible `Julia` implementation is

```julia
function common_root(p1, p2)
#
# Syntax:       haveCommonRoot = common_root(p1, p2)
#
# Inputs:       p1 is a vector of length m + 1 with p1[1] != 0 that defines an
#               mth degree polynomial of the form:
#
#               P1(x) = p1[1]x^m + p1[2]x^(m - 1) + ... + p1[m]x + p1[m + 1]
#
#               p2 is a vector of length n + 1 with p2[1] != 0 that defines an
#               nth degree polynomial of the form:
#
#               P2(x) = p2[1]x^n + p2[2]x^(n - 1) + ... + p2[n]x + p2[n + 1]
#
# Outputs:      haveCommonRoot = true when P1 and P2 share a common root and
#               false otherwise
#
# Description:  Determines whether the input polynomials share a common root
#

    # Constants
    DELTA = 1e-6 # Zero-determinant tolerance

    # Construct companion matrices
    A = compan(p1)
    B = compan(p2)

    # Compute Kronecker sum of A and -B
    C = kron(A, eye(B)) - kron(eye(A), B)

    # Check for common roots
    return (abs(det(C)) < DELTA)
end

# Construct matrix with eigenvalues equal to the roots of polynomial p
function compan(p)
    n = length(p)
    A = [(-1 / p[1]) * vec(p[2:n])'
         eye(n - 2, n - 1)]
    return A
end
```

Pr. 4. (sol/hs019)
From the software experiment, we observe that `eigvals(A)` are equal to `1 ./ eigvals(B)` .

Given a vector $\boldsymbol{u} = \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix}$, the characteristic equation of the companion matrix $\boldsymbol{A}$ is

$$p_1(z) = z^n + \frac{u_2}{u_1} z^{n-1} + \frac{u_3}{u_1} z^{n-2} + \ldots + \frac{u_n}{u_1} = 0.$$

`flipdim` inverts the vector $\boldsymbol{u}$, so that the characteristic equation of the matrix $\boldsymbol{B}$ is

$$p_2(z) = z^n + \frac{u_{n-1}}{u_n} z^{n-1} + \frac{u_{n-2}}{u_n} z^{n-2} + \ldots + \frac{u_1}{u_n} = 0.$$

Using the transformation of variables $y = \dfrac{1}{z}$ in $p_2(z)$, yields the equation

$$p_2(y) = \left(\frac{1}{y}\right)^n + \frac{u_{n-1}}{u_n} \left(\frac{1}{y}\right)^{n-1} + \frac{u_{n-2}}{u_n} \left(\frac{1}{y}\right)^{n-2} + \ldots + \frac{u_1}{u_n} = 0.$$

Multiplying throughout by $\dfrac{u_n}{u_1} y^n$, we get

$$p_2(y) = \frac{u_n}{u_1} + \frac{u_{n-1}}{u_1} y + \frac{u_{n-2}}{u_1} y^2 \ldots + y^n = 0,$$

which has the same roots as $p_1(z)$. But the roots of $p_1(z)$ are the eigenvalues of $\boldsymbol{A}$, and the roots of $p_2(y)$ are the reciprocals of the eigenvalues of $\boldsymbol{B}$, due to the transformation $z = \dfrac{1}{y}$. Since the roots of the two equations are equal, $\operatorname{eig}(\boldsymbol{A}) = \dfrac{1}{\operatorname{eig}(\boldsymbol{B})}$.

Pr. 5. (sol/hs024)

(a) Let $p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0$, where $a_n \neq 0$. Then we form the companion matrix of $p(x)$ as:

$$\boldsymbol{P} = \begin{bmatrix} -\frac{a_{n-1}}{a_n} & -\frac{a_{n-2}}{a_n} & -\frac{a_{n-3}}{a_n} & \ldots & -\frac{a_0}{a_n} \\ 1 & 0 & 0 & \ldots & \vdots \\ 0 & 1 & 0 & \ldots & \vdots \\ & & \ddots & & \vdots \\ 0 & 0 & 0 & \ldots & 0 \end{bmatrix}.$$

From class, we know that the roots of $p(x)$ are equal to the eigenvalues of $\boldsymbol{P}$. Therefore, the largest root of $p(x)$ equals to the largest eigenvalue of $\boldsymbol{P}$. Since we have assumed that the largest root of $p(x)$ is non-repeating, we can find it by computing the largest eigenvalue of $\boldsymbol{P}$ using the Power Iteration. Starting with a random unit norm vector $\boldsymbol{x}_0$, run the algorithm

$$\boldsymbol{x}_k = \frac{\boldsymbol{P}\boldsymbol{x}_{k-1}}{\|\boldsymbol{P}\boldsymbol{x}_{k-1}\|_2}, \qquad k = 1, 2, \ldots,$$

until iteration $k_{\max}$, when some termination criterion is reached (e.g., $\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|_2 \leq \varepsilon$, where ε is the required tolerance). If $\boldsymbol{q}_1$ is the eigenvector of $\boldsymbol{A}$ corresponding to λ_1, its largest eigenvalue (in magnitude),

$$\boldsymbol{A}\boldsymbol{x}_{k_{\max}} \approx \boldsymbol{A}\boldsymbol{q}_1 = \lambda_1 \boldsymbol{q}_1 \implies \|\boldsymbol{A}\boldsymbol{x}_{k_{\max}}\|_2 \approx \|\lambda_1 \boldsymbol{q}_1\|_2 = \lambda_1,$$

which is the required largest root of $p(x)$. Recall from a previous problem that if $\{\lambda_i\}_{i=1}^n$ are the roots of $p(x)$, then the roots of

$$p'(x) = a_0 x^n + a_1 x^{n-1} + \ldots + a_n$$

will be $\{\frac{1}{\lambda_i}\}_{i=1}^n$. If $\lambda_n \neq 0$ is the smallest root of $p(x)$, we can apply the above algorithm to find the largest root of $p'(x)$ via its companion matrix

$$\boldsymbol{P}' = \begin{bmatrix} -\frac{a_1}{a_0} & -\frac{a_2}{a_0} & -\frac{a_3}{a_0} & \dots & -\frac{a_n}{a_0} \\ 1 & 0 & 0 & \dots & \vdots \\ 0 & 1 & 0 & \dots & \vdots \\ & & \ddots & & \vdots \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix}.$$

Let $\lambda_1' = \frac{1}{\lambda_n}$ be the largest eigenvalue of $\boldsymbol{P}'$ computed via Power Iteration. The required smallest root of $p(x)$ is then computed as $\frac{1}{\lambda_1'}$.

(b) A possible `Julia` implementation is

```
function outlying_zeros(p, v0, nIters)
#
# Syntax:       zmax, zmin = outlying_zeros(p, v0, nIters)
#
# Inputs:       p is a vector of length n + 1 defining the polynomial
#               P(x) = p[1]x^n + p[2]x^n-1 + ... + p[n]x + p[n + 1]
#
#               v0 is a vector of length n
#
#               nIters is the number of power iterations to perform
#
# Outputs:      zmax is the zero of P(x) with largest magnitude
#
#               zmin is the zero of P(x) with smallest magnitude
#
# Description:  Uses power iteration to compute the largest and smallest
#               magnitude zeros, respectively, of the polynomial defined by the
#               input coefficients
#

    # Compute largest absolute zero
    Cmax = compan(p)
    zmax = powerIteration(Cmax, v0, nIters)[1]

    # Compute smallest absolute zero
    Cmin = compan(p[end:-1:1])
    zmin = 1 / powerIteration(Cmin, v0, nIters)[1]

    return zmax, zmin
end

function powerIteration(A, v0, nIters)
#
# Syntax:       lambda1, v1 = powerIteration(A, v0, nIters)
#
# Inputs:       A is an n x n matrix
#
#               v0 is a vector of length n
#
#               nIters is the number of iterations to perform
#
# Outputs:      lambda1 is (an estimate of) the largest absolute eigenvalue of A
#
#               v1 is a vector of length n containing (an estimate of) the
#               eigenvector associated with eigenvalue lambda1
#
# Description:  Uses power iteration to approximate the largest absolute
#               eigenpair of A
#

    # Power iteration
    lambda1 = 0 # Declare
    v1 = v0 / norm(v0)
    for _ in 1:nIters
```

```
        Av1 = A * v1
        lambda1 = v1' * Av1
        v1 = Av1 / norm(Av1)
    end

    return lambda1[1], v1 # lambda1 is a 1 x 1 array
end

# Construct matrix with eigenvalues equal to the roots of polynomial p
function compan(p)
    n = length(p)
    A = [(-1 / p[1]) * vec(p[2:n])'
         eye(n - 2, n - 1)]
    return A
end
```

Pr. 6. (sol/hsj90)
Thank you for your feedback via the course evaluations.
