Problem Set 5 Solutions

Problem 1. *Linear algebra meets advanced differential equations*

Now that we have hashed out how to create an orthogonal set from a set of linearly independent vectors (both of which are bases for their spans), we can see where we use this idea in contexts outside of the standard inner product space $(\mathbb{R}^n, \cdot)$. Within the context of function spaces like $(C[a, b], L^2)$, we are able to generate orthogonal solutions to differential equations (whereby we can represent *all* solutions of the equation as linear combinations of this basis). These functions behave just like eigenvectors, so we call them *eigenfunctions* (of what linear transformation..?).

For example, it is quite often the case that problems in Physics and Engineering can be framed as differential equations of the form

$$\frac{d}{dx}\left[p(x)\frac{dy}{dx}\right] + q(x)y = \lambda r(x)y,$$

where $p, q, r \in C^1[a, b]$ (the vector space of continuous functions with continuous first derivative on $[a, b]$), λ is a constant called an *eigenvalue*, and $y(x)$ is a function satisfying this equation. We assign to $C^1[a, b]$ the *weighted L^2* inner product given by

$$\langle f(x), g(x)\rangle = \int_a^b r(x) \cdot f(x)g(x)\, dx,$$

and this makes $(C^1[a, b], \langle \cdot, \cdot\rangle)$ into an inner product space (provided $r(x) > 0$ on $[a, b]$).

These kinds of equations are called *Sturm-Liouville equations*, and their solution is of vital importance in many applications of Physics and Engineering. We examine two of them below and see (now that we've taken such a large tangent) how they are related to linear algebra.

(a) First verify that $(1 - x^2)y'' - 2xy' + \nu(\nu + 1)y = 0$, where $\nu \in \mathbb{R}$ is constant, is a Sturm-Liouville equation on the interval $[-1, 1]$ with

$$p(x) = (1 - x^2), \quad q(x) = 0, \quad r(x) = 1, \quad \text{and} \quad \lambda = -\nu(\nu + 1)$$

and that $xy'' + (1 - x)y' + ny = 0$, where $n \in \mathbb{R}$ is constant, is a Sturm-Liouville equation on the interval $[0, \infty)$ with

$$p(x) = xe^{-x}, \quad q(x) = 0, \quad r(x) = e^{-x}, \quad \text{and} \quad \lambda = -n.$$

These are the *Legendre* and *Laguerre equations*, respectively. (Yes, it is often the case that the q function is conspicuously absent, but we still allow for non-zero q functions.)

For the Legendre equation, we simply check that

$$
\begin{aligned}
\frac{d}{dx}\left[p(x)\frac{dy}{dx}\right] + q(x)y &= \lambda r(x)y \\
\left[(1 - x^2)y'\right]' &= -\nu(\nu + 1)(1)y \\
-2xy' + (1 - x^2)y'' &= -\nu(\nu + 1)y \\
(1 - x^2)y'' - 2xy' + \nu(\nu + 1)y &= 0,
\end{aligned}
$$

and this shows that the Legendre equation is a Sturm-Liouville equation. Similarly, for the Laguerre equation, we check that

$$\frac{d}{dx}\left[p(x)\frac{dy}{dx}\right] + q(x)y = \lambda r(x)y$$

$$\left[xe^{-x}y'\right]' = -ne^{-x}y$$

$$e^{-x}(1-x)y' + xe^{-x}y'' = -ne^{-x}y$$

$$e^{-x}[xy'' + (1-x)y' + ny] = 0$$

$$xy'' + (1-x)y' + ny = 0,$$

and this shows that the Laguerre equation is a Sturm-Liouville equation as well.

(b) Apply the Gram-Schmidt process on the linearly independent set $\beta_2 = \{1, x, x^2\}$ to construct the orthogonal sets $\mathcal{L}_1$ and $\mathcal{L}_2$ from the Legendre and Laguerre equations, respectively. Be sure to use the correct inner product induced on $C^1[-1, 1]$ and $C^1[0, \infty)$ by the Legendre and Laguerre equations. Present $\mathcal{L}_1$ and $\mathcal{L}_2$ so that the highest power term in each vector has the unit 1 coefficient (that makes them *monic*).

We first construct $\mathcal{L}_1$. Set $\vec{u}_1 = 1$, and then compute

$$\langle x, 1 \rangle = \int_{-1}^{1} r(x) \cdot x \cdot 1 \, dx = \int_{-1}^{1} x \, dx = 0 \quad \text{and} \quad \langle 1, 1 \rangle = \int_{-1}^{1} 1 \, dx = 2.$$

The first step of Gram-Schmidt then gives us

$$\vec{u}_2 = x - \frac{\langle 1, x \rangle}{\|1\|^2}\vec{u}_1 = x - 0 \cdot 1 = x.$$

We then compute

$$\langle x^2, 1 \rangle = \int_{-1}^{1} x^2 \, dx = \frac{2}{3}, \quad \langle x^2, x \rangle = \int_{-1}^{1} x^3 \, dt = 0, \quad \text{and} \quad \|x\|^2 = \int_{-1}^{1} x^2 \, dx = \frac{2}{3}.$$

The next step of Gram-Schmidt then gives us

$$\vec{u}_3 = x^2 - \frac{\langle x^2, 1 \rangle}{\langle \|1\|^2 \rangle}\vec{u}_1 - \frac{\langle x^2, x \rangle}{\langle \|x\|^2 \rangle}\vec{u}_2 = x^2 - \frac{1}{3} \cdot 1 - 0 \cdot x = x^2 - \frac{1}{3}.$$

These vectors are already monic, so we construct

$$\mathcal{L}_1 = \{\vec{u}_1, \vec{u}_2, \vec{u}_3\} = \left\{1, x, x^2 - \frac{1}{3}\right\}.$$

We now construct $\mathcal{L}_2$. Again set $\vec{u}_1 = 1$, and then compute (via integration by parts)

$$\langle x, 1 \rangle = \int_{0}^{\infty} r(x) \cdot x \cdot 1 \, dx = \int_{0}^{\infty} xe^{-x} \, dx = 1 \quad \text{and} \quad \|1\|^2 = \int_{0}^{\infty} e^{-x} \, dx = 1.$$

The first step of Gram-Schmidt then gives us

$$\vec{u}_2 = x - \frac{\langle x, 1 \rangle}{\|1\|^2}\vec{u}_1 = x - 1 \cdot 1 = x - 1.$$

We then compute

$$\langle x^2, 1 \rangle = \int_0^\infty x^2 e^{-x}\, dx = 2, \quad \langle x^2, x-1 \rangle = \int_0^\infty (x^3 - x^2)e^{-x}\, dx = 4,$$

$$\text{and} \quad \|x-1\|^2 = \int_0^\infty (x^2 - 2x + 1)e^{-x}\, dx = 1.$$

The next step of Gram-Schmidt then gives us

$$\vec{u}_3 = x^2 - \frac{\langle x^2, 1 \rangle}{\langle \|1\|^2 \rangle}\vec{u}_1 - \frac{\langle x^2, x-1 \rangle}{\langle \|x-1\|^2 \rangle}\vec{u}_2 = x^2 - \frac{4}{1}\cdot(x-1) - \frac{2}{1}\cdot(1) = x^2 - 4x + 2.$$

These vectors are again already monic, so we construct

$$\mathcal{L}_2 = \{\vec{u}_1, \vec{u}_2, \vec{u}_3\} = \{1, x-1, x^2 - 4x + 2\}$$

(c) (Optional extra credit) Show that each element of $\mathcal{L}_1$ is a solution to the Legendre equation for the appropriate choices of ν. Show that each element of $\mathcal{L}_2$ is a solution to the Laguerre equation for the appropriate choices of n.

We begin with the Legendre equation. If we take $y(x) = 1$, then choosing $\nu = 0$ gives us

$$(1 - x^2)\cdot 0 - 2x \cdot 0 + 0\cdot(0+1)\cdot 1 = 0.$$

Taking $y(x) = x$, we choose $\nu = 1$ so that

$$(1 - x^2)\cdot 0 - 2x \cdot 1 + 1\cdot(1+1)x = 0.$$

Taking $y(x) = x^2 - \frac{1}{3}$, we choose $\nu = 2$ so that

$$(1 - x^2)\cdot 2 - 2x \cdot 2x + 2\cdot(2+1)\cdot\left(x^2 - \frac{1}{3}\right) = 0.$$

We now turn to the Laguerre equation. If we take $y(x) = 1$, then choosing $n = 0$ gives us

$$x \cdot 0 + (1 - x)\cdot 0 + 0\cdot 1 = 0.$$

Taking $y(x) = t - 1$, we choose $n = 1$ so that

$$x \cdot 0 + (1 - x)\cdot 1 + 1\cdot(x - 1) = 0.$$

Taking $y(x) = x^2 - 4x + 2$, we choose $n = 2$ so that

$$x \cdot 2 + (1 - x)\cdot(2x - 4) + 2\cdot(x^2 - 4x + 2) = 0.$$

Remark - We can continue this process in part (b) for $\beta_K = \{1, t, t^2, \ldots, t^K\}$ of any size. If we do this indefinitely, we produce the sequences of polynomials called the *Legendre polynomials* and the *Laguerre polynomials*. These constitute a convenient notion of an *eigenvector basis* for $C^1[-1, 1]$ and $C^1[0, \infty)$, respectively. We can apply this more generally to many Sturm-Liouville equations, and the broader theory is called *Sturm-Liouville theory*. Take a course in PDEs if you're interested in the math behind this theory.

Problem 2. *Property passing in factorization algorithms - and a teaser*

We would like to study how efficient the Gram-Schmidt algorithm is in the computer. There is a problem with doing this with MATLAB's built-in QR factorizer `qr()` : it has been modified so much that it almost doesn't resemble Gram-Schmidt anymore! Therefore, we must first create our own QR factorizer in order to study why *MathWorks* felt the need to bastardize Gram-Schmidt.

(a) Consult page 206 of the textbook to find the *modified Gram-Schmidt* algorithm (this is the boxed algorithm from the lecture notes). Use this pseudo-code to implement a MATLAB function `myQR()` that:

- Takes in a rectangular matrix A as its only parameter.

- Stops and delivers the message "columns linearly dependent" if the columns of A are linearly dependent.

- Otherwise returns an orthogonal matrix Q and an upper triangular (square) matrix R such that $A = QR$.

Here is my `myQR()` code verbatim:

```
function [ Q,R ] = myQR( A )
%myQR() - Creates the QR-factorization
%          of the mxn matrix A if it exists.
%          Builds Q by reassigning the entries
%          of A.
%
%    Author: Kevin Lamb
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Initializations and easy checks
[m,n] = size(A);
R = zeros(n);
Q = zeros(m,n);

if m < n
    disp('Columns are linearly dependent');
    return
end

%% Begin building Q and R

for jj = 1:n

    % Get R-pivot scalar
    R(jj,jj) = sqrt(A(:,jj)'*A(:,jj));

    % Dependence check
    if R(jj,jj) == 0
        disp('Columns are linearly dependent');
        return
    else
```

```
            % Normalize column jj
            A(:,jj) = A(:,jj)/R(jj,jj);
        end % Dependence/normalization block

        for kk = (jj+1):n
            % Create jj column of R
            R(jj,kk) = A(:,jj)'*A(:,kk);

            for ii = 1:m
                A(ii,kk) = A(ii,kk) - A(ii,jj)*R(jj,kk);
            end % ii-loop
        end % kk-loop

    end % jj-loop
    Q = A;
    end
```

(b) For positive integers $m > n$, consider the family of *n-banded* matrices

$$
A_m(n) = \begin{bmatrix}
n & n-1 & n-2 & \cdots & 1 & & & & & \\
n-1 & n & n-1 & \cdots & 2 & 1 & & & & \\
n-2 & n-1 & n & \cdots & 3 & 2 & 1 & & & \\
\vdots & \vdots & \vdots & & & & \ddots & & & \\
1 & 2 & 3 & & & & & & & \\
 & 1 & 2 & & & & & & & \\
 & & 1 & & \ddots & & & \ddots & & \\
 & & & & & & & n & n-1 \\
 & & & & & & & n-1 & n
\end{bmatrix}_{m \times m}.
$$

Use your functions `myLU()` and `myQR()` to compute the LU- and QR-factorizations of $A_m(n)$. For varying values of m and n, what patterns to you notice in the matrices L, U, Q, and R? Make a conjecture about the form and/or structure of the factors in $LU = QR = A_m(n)$.

You will find that the matrices L is *lower* n-banded and lower-unitriangular, and the matrices U is *upper* n-banded and upper-triangular. The matrix R is not upper n-banded, but it does appear to be upper $(2n-1)$-banded. and upper-triangular. The matrix Q is close to being n-banded. It is *lower* n-banded, but it is still dense above its diagonal. Below are the factors for $A_7(3)$ as an example:

```
A =

     3     2     1     0     0     0     0
     2     3     2     1     0     0     0
     1     2     3     2     1     0     0
     0     1     2     3     2     1     0
     0     0     1     2     3     2     1
     0     0     0     1     2     3     2
     0     0     0     0     1     2     3
```

```
L =

    1.0000         0         0         0         0         0         0
    0.6667    1.0000         0         0         0         0         0
    0.3333    0.8000    1.0000         0         0         0         0
         0    0.6000    0.7500    1.0000         0         0         0
         0         0    0.6250    0.8333    1.0000         0         0
         0         0         0    0.6667    0.8750    1.0000         0
         0         0         0         0    0.7500    0.8571    1.0000


U =

    3.0000    2.0000    1.0000         0         0         0         0
         0    1.6667    1.3333    1.0000         0         0         0
         0         0    1.6000    1.2000    1.0000         0         0
         0         0         0    1.5000    1.2500    1.0000         0
         0         0         0         0    1.3333    1.1667    1.0000
         0         0         0         0         0    1.3125    1.1250
         0         0         0         0         0         0    1.2857


Q =

    0.8018   -0.5000    0.2113    0.0740   -0.1667    0.1422    0.0949
    0.5345    0.5000   -0.5494    0.1629    0.1667   -0.3289   -0.0000
    0.2673    0.5000    0.4648   -0.5479    0.1667    0.2311   -0.2847
         0    0.5000    0.2958    0.4591   -0.5000    0.2400    0.3797
         0         0    0.5916    0.3258    0.3333   -0.6578    0.0000
         0         0         0    0.5923    0.3333    0.4623   -0.5695
         0         0         0         0    0.6667    0.3378    0.6644


R =

    3.7417    3.7417    2.6726    1.0690    0.2673         0         0
         0    2.0000    3.0000    3.0000    1.5000    0.5000         0
         0         0    1.6903    2.4509    2.8313    1.4790    0.5916
         0         0         0    1.6882    2.5323    2.8877    1.5105
         0         0         0         0    1.5000    2.5000    3.0000
         0         0         0         0         0    0.9868    1.2801
         0         0         0         0         0         0    0.8542
```

(c) Now consider the matrices

$$
B_n = \begin{bmatrix}
10^{-1} & 1 & 1 & 1 & \cdots & 1 \\
10^{-1} & 10^{-2} & 0 & 0 & \cdots & 0 \\
10^{-1} & 10^{-2} & 10^{-3} & 0 & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
10^{-1} & 10^{-2} & 10^{-3} & 10^{-4} & \cdots & 10^{-n}
\end{bmatrix}_{n \times n}.
$$

Use your `myQR()` function compute the QR-factorizations of B_2, B_5, B_{10}, B_{20}, and B_{50}. In the cases of $n = 2, 5, 10, 20, 50$, use these factorizations to solve $B_n \vec{x} = \vec{1}_n$, where $\vec{1}_n$ is the n-dimensional vector containing all ones. Your should find $\vec{x}_n = 10\vec{e}_1$ for each case, where $\vec{e}_1$ is the appropriately sized first standard basis vector. Do you feel like you can trust your factorizer?

Here are the QR-factorizations of the matrices B_2 and B_5.

```
B_2 =

    0.1000    1.0000
    0.1000    0.0100


Q =

    0.7071    0.7071
    0.7071   -0.7071


R =

    0.1414    0.7142
         0    0.7000


x_2 =

   10.0000
    0.0000


B_5 =

    0.1000    1.0000    1.0000    1.0000    1.0000
    0.1000    0.0100         0         0         0
    0.1000    0.0100    0.0010         0         0
    0.1000    0.0100    0.0010    0.0001         0
    0.1000    0.0100    0.0010    0.0001    0.0000


Q =
```

```
     0.4472      0.8944     -0.0000     -0.0000      0.0000
     0.4472     -0.2236     -0.8660     -0.0000     -0.0000
     0.4472     -0.2236      0.2887     -0.8165     -0.0000
     0.4472     -0.2236      0.2887      0.4082     -0.7071
     0.4472     -0.2236      0.2887      0.4082      0.7071


R =

     0.2236      0.4651      0.4486      0.4473      0.4472
          0      0.8855      0.8938      0.8944      0.8944
          0           0      0.0009      0.0001      0.0000
          0           0           0      0.0001      0.0000
          0           0           0           0      0.0000


x_5 =

    10.0000
     0.0000
     0.0000
     0.0000
    -0.0000
```

For practicality, we do not include the larger B_{10} and B_{20} matrices; however, we now include $\vec{x}_1 0$ and $\vec{x}_2 0$ to see how wild they become:

```
x_10 =

   1.0e+04 *

   -0.6515
    6.5247
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0042
    0.2670
   -6.7307


Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 7.676166e-50.
> In Untitled at 24

x =
```

```
1.0e+48 *

 -0.0000
 -0.0000
 -0.0000
 -0.0000
  0.0000
 -0.0000
  0.0000
 -0.0000
  0.0000
 -0.0000
  0.0000
  0.0000
  0.0000
 -0.0000
  0.0000
  0.0000
 -0.0000
  0.0000
 -7.7947
  7.7947
```

We notice the handy little WARNING when we try to compute $\vec{x}_{20}$. We can see that the first two columns of B_n aren't multiples of one another, so they are independent of each other; and the remaining columns are orthogonal (with respect to the dot product) to each other in addition to the first two columns. This means that B_n is invertible, so we know that $\vec{x}_n = 10\vec{e}_1$ is the only solution to this system. Therefore, we know that this factorizer is inaccurate, and we can't trust its output.

(d) If this is so bloody inaccurate, why do we need to make it better? Isn't there something else that we could use? Well, sure... but in order to see why we want the QR-factorization algorithms to be better, consider the small example where

$$M_0 = \begin{bmatrix} 4 & 1 \\ 1 & 4 \end{bmatrix}.$$

Use your `myQR()` function to help you carry out the following iterative scheme

$$M_{n+1} = R_n Q_n \quad \text{for} \quad n \geq 0,$$

where we simply create the next matrix in the sequence by reversing the order of the multiplication in the QR-factorization $Q_n R_n = M_n$.

If we do this enough times, what happens to Q_n and R_n? Explain how the entries in R_n are related to M_0 if n is large enough.

Below is the result of running this iteration until the QR-factorization of M_{10}:

```
EDU>> A = [4 1;1 4]

A =

     4     1
     1     4

EDU>> [Q,R] = myQR(A)

Q =

    0.9701   -0.2425
    0.2425    0.9701


R =

    4.1231    1.9403
         0    3.6380

EDU>> [Q,R] = myQR(R*Q)

Q =

    0.9811   -0.1936
    0.1936    0.9811


R =

    4.5568    1.5491
         0    3.2918

EDU>> [Q,R] = myQR(R*Q)

Q =

    0.9912   -0.1324
    0.1324    0.9912


R =

    4.8129    1.0595
         0    3.1166

EDU>> [Q,R] = myQR(R*Q)
```

```
Q =

    0.9965   -0.0838
    0.0838    0.9965


R =

    4.9282    0.6700
         0    3.0437

EDU>>  [Q,R] = myQR(R*Q)

Q =

    0.9987   -0.0513
    0.0513    0.9987


R =

    4.9735    0.4100
         0    3.0160

EDU>>  [Q,R] = myQR(R*Q)

Q =

    0.9995   -0.0310
    0.0310    0.9995


R =

    4.9904    0.2478
         0    3.0058

EDU>>  [Q,R] = myQR(R*Q)

Q =

    0.9998   -0.0186
    0.0186    0.9998


R =

    4.9965    0.1491
```

```
         0    3.0021

EDU>> [Q,R] = myQR(R*Q)

Q =

    0.9999   -0.0112
    0.0112    0.9999


R =

    4.9987    0.0895
         0    3.0008

EDU>> [Q,R] = myQR(R*Q)

Q =

    1.0000   -0.0067
    0.0067    1.0000


R =

    4.9995    0.0537
         0    3.0003

EDU>> [Q,R] = myQR(R*Q)

Q =

    1.0000   -0.0040
    0.0040    1.0000


R =

    4.9998    0.0322
         0    3.0001
```

It would appear that Q_n is converging to the identity matrix, and R_n appears to be converging to a diagonal matrix with entries 5 and 3.
We quickly check

```
EDU>> eig(A)

ans =
```

$$\begin{matrix} 3 \\ 5 \end{matrix}$$

to verify that the diagonal entries of R_n are indeed **converging to the eigenvalues of** A. This is completely unexpected and should strike us as remarkable, if unbelievable.