Problem Set 5

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.)

(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*).

(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.

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 $A = QR$.

(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 $LU = QR = A_m(n)$.

(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 $B_n\vec{\mathbf{x}} = \vec{\mathbf{1}}_n$, where $\vec{\mathbf{1}}_n$ is the n-dimensional vector containing all ones. Your should find $\vec{\mathbf{x}}_n = 10\vec{\mathbf{e}}_1$ for each case, where $\vec{\mathbf{e}}_1$ is the appropriately sized first standard basis vector. Do you feel like you can trust your factorizer?

(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.