1 Yet another linear system

Fall 2018
Consider the system $D : \mathbb{C}^{\mathbb{Z}} \to \mathbb{C}^{\mathbb{Z}}$ defined by $(Dx)[n] := x[n] - x[n-1]$.

1. Compute the adjoint D^* (with respect to the standard inner product)
2. Let $L = DD^*$. Compute $(Lx)[n]$ explicitly
3. Determine if L is: (a) linear, (b) shift invariant, (c) causal, (d) memoryless, (e) BIBO
4. For each x_k given below, find $y_k = Lx_k$. Sketch x_k and y_k, and explain the effect of L.

$$x_1[n] = c \quad \text{for all } n \in \mathbb{Z} \qquad \text{(A constant sequence)}$$
$$x_2[n] = \delta[n] \qquad \text{(the unit impulse sequence)}$$
$$x_3[n] = u[n] = \begin{cases} 1 & n \geq 0 \\ 0 & \text{o.w.} \end{cases} \qquad \text{(the unit impulse sequence)}$$

Your Solution Here!

2 A Taste of Tensor Analysis in MIMO processing

Fall 2018
Suppose that N audio sources produce waveforms of length M, say $\{x_n\}_{n=1}^{N} \in \mathbb{C}^M$. An array of P microphones pick up a mixture (linear combination) of those pulses, each microphone getting

$$y_p = \sum_{n=1}^{N} b_{n,p} x_n, \quad p = 1, ..., P$$

here $(b_{n,p})$ account for multipath effects (reflections resulting in multiple temporally spaced arrivals of a signal) and channel loss (attenuation throughout propagation). Then, each sample y_p is processed by some post-filter $A \in \mathbb{C}^{L \times M}$ to a vector z_p of length L:

$$z_p = A y_p \in \mathbb{C}^L$$

By stacking the inputs and outputs as columns of matrices $X \in \mathbb{C}^{M \times N}$ and $Z \in \mathbb{C}^{L \times P}$, we can write the overall system as a matrix product (here $B = (b_{n,p}) \in \mathbb{C}^{N \times P}$),

$$Z = T(X) := AXB \tag{1}$$

1. Show that $T : \mathbb{C}^{M \times N} \to \mathbb{C}^{L \times P}$ is indeed linear (in X)
2. Show that the matrices $\{E_{m,n}\}$ defined below constitute a basis for $\mathbb{C}^{M \times N}$:

   $$E_{m,n} := e_m f_n^\top, \quad m = 1, ..., M, \quad n = 1, ..., N \tag{2}$$

   here $\{e_m\}_{m=1}^{M}$ and $\{f_n\}_{n=1}^{N}$ are the canonical bases for $\mathbb{C}^M$ and $\mathbb{C}^N$ respectively.
3. Prove that $\text{vec}(T(X)) = (B^\top \otimes A)\text{vec}(X)$, here $\otimes$ denotes the Kronecker Product and vec denotes the vectorization operation.

Your Solution Here!

3 Linear Estimation and Regularization

You wish to estimate some linear map $f(\mathbf{x}) = \sum_{k=1}^{N} \alpha_k x_k = \boldsymbol{\alpha}^\top \mathbf{x}$ from which you have taken a set of N measurements. Inherent to our measurement process is some form of adaptive signal amplification, and so the noise becomes somehow proportional to the observation. For this reason, you model a given measurement as

$$y_i = (1 + n_i)\mathbf{x}_i^\top \boldsymbol{\alpha} \tag{3}$$

where each $n_i \sim \mathcal{N}(0, \sigma^2)$ is distributed in an i.i.d. fashion.

Let $\mathbf{x}_i \in \mathbb{R}^{1\times N}$ be a set of measurement parameters, and let

$$X = \begin{bmatrix} \text{---}\ \mathbf{x}_1\ \text{---} \\ \text{---}\ \mathbf{x}_2\ \text{---} \\ \vdots \\ \text{---}\ \mathbf{x}_N\ \text{---} \end{bmatrix}$$

be a matrix representation of all the parameter sets tested. If we let $\mathbf{y} \in \mathbb{R}^N$ be the set of measurements, we can denote our forward model of the system to be

$$\mathbf{y} = (I + N)X\boldsymbol{\alpha},$$

where N is a diagonal matrix of our measurement noise. We have a goal of estimating $\boldsymbol{\alpha}$ given knowledge of $\mathbf{y}$, X, and σ^2.
Assume X is invertible, and define $\sigma_1 = \|X\|_2$, $\sigma_N = 1/\|X^{-1}\|_2$, and $\kappa = \sigma_1/\sigma_N$.

1. Consider constructing an estimate $\hat{\boldsymbol{\alpha}}_1 = X^{-1}\mathbf{y}$.
   You may be concerned about what amount of relative error is likely in such a case. One rough way of quantifying it is to look at the interval that contains most of the probability mass. Give an upper bound for the relative error ($\|\alpha - \hat{\alpha}\|/\|\alpha\|$) that occurs with some fairly high probability. **Hint:** If an operator norm corresponds to some maximal amplification, how do you bound $\|AB\|$?

2. Now, consider the case where σ_N is very small. In this case, we risk having very poor performance for the estimate. Consider instead using a *regularized* $\tilde{X} = X + \lambda I$, for some relatively small λ. Give upper bounds for $\|\tilde{X}\|$ and $\|\tilde{X}^{-1}\|$.
   Hint: Recall that the operator norm corresponds to the maximum amplification, so how much can $(X + \lambda I)$ amplify a vector?

3. Give a similar bound on the relative error of $\hat{\boldsymbol{\alpha}}_2 = \tilde{X}^{-1}\mathbf{y}$

4. How would you choose λ to reduce this "worst likely case" error?

Your Solution Here!

Computational Problem — Gram-Schmidt

In this problem, you will implement the Gram-Schmidt algorithm to generate an orthonormal basis. We will observe that, despite the theoretical promise, the basis generated is not actually very orthogonal. You will then implement a variant called Modified Gram-Schmidt (MGS) and observe the improved performance. Finally, the Gram-Schmidt process will be used to generate a visually interesting basis for images. Some of the following discussion comes from Trefethen and Bau [1], which is a book on numerical linear algebra. This will conclude our discussion on the topic for the semester.

The assignment is formatted in much the same way as the previous one. There is a python file `HW2.py` with function definitions, a file with some basic tests `HW2_test.py` to help with debugging, and a Jupyter notebook to explore the algorithms a little. In the previous assignment, you just implemented the functions first. In this case, it can be a little more interactive, and the Jupyter notebook has a section affiliated with each phase of the assignment.

Gram-Schmidt

For your convenience, we've copied the pseudo-code from lecture 4 here.

Algorithm 1 Gram-Schmidt Orthogonalization

```
1: procedure GRAM-SCHMIDT($\mathbf{a}_1, ...\mathbf{a}_N$)        ▷ Orthogonal Basis for span($\mathbf{a}_1, ..., \mathbf{a}_N$)
2:     $\mathbf{r}_1 \leftarrow \mathbf{a}_1/\|\mathbf{a}_1\|$
3:     for $1 < i \leq N$ do
4:         $\mathbf{r}_i \leftarrow P_i^{\perp}\mathbf{a}_i$        ▷ Project onto Orthogonal Subspace
5:         $\mathbf{r}_i \leftarrow \mathbf{r}_i/\|\mathbf{r}_i\|$        ▷ Normalize
6:     end for
7:     return ($\mathbf{r}_1, ..., \mathbf{r}_N$)
8: end procedure
```

In this case, each vector is a row of a Numpy matrix. Remember that the projection is orthogonal to all previously orthonormalized vectors.

Once you have implemented the algorithm, you may want to explore the first section of the Jupyter notebook, though nothing is required to be submitted at this point. The first phase just motivates the usage of a modified algorithm.

Modified Gram-Schmidt

Hopefully you managed to observe the loss of orthogonality, which leads to the question of how do we actually fix this problem? There are better ways to generate an orthogonal basis than the technique presented here, but this technique is important for its connections to other, similar techniques for different problems. It is also a much more straightforward method to understand than the alternatives.

The modification is simply to change the order of the projections. Rather than construct a new vector that is orthogonal to all previous vectors, orthogonalize all future vectors to the current one. Please give an explanation for why this process is equivalent to the original Gram-Schmidt when ignoring floating-point arithmetic. It does not need to be mathematically precise.

Your Solution Here!

We've included pseudo-code below, where $P_i^{\perp}$ now represents an orthogonal projection onto the subspace orthogonal to q_i

Algorithm 2 MGS Orthogonalization

```
1: procedure MODIFIED GRAM-SCHMIDT($\mathbf{q}_1, ...\mathbf{q}_N$)    ▷ Orthogonal Basis for span($\mathbf{q}_1, ..., \mathbf{q}_N$)
2:     for $1 \leq i \leq N$ do
3:         $\mathbf{q}_i \leftarrow \mathbf{q}_i / \|\mathbf{q}_i\|$    ▷ Normalize
4:         for $i+1 \leq j \leq N$ do
5:             $\mathbf{q}_j \leftarrow P_i^{\perp} \mathbf{q}_j$    ▷ Project onto Orthogonal Subspace
6:         end for
7:     end for
8:     return ($\mathbf{q}_1, ..., \mathbf{q}_N$)
9: end procedure
```

Why does the modification improve stability?

Think for a minute about the difference in the projections, and pretend that the numerical errors from each projection introduce a small, but nonzero error oriented randomly in $\mathbb{R}^N$. A loose explanation is on the next page once you've thought about it for a couple of minutes.

In the classical algorithm, the orthogonal projection can be thought of as projecting the original vector onto each of the previous basis vectors, then subtracting each of those projections together. Each of these projections onto a 1D orthogonal subspace potentially introduces a small error.

In contrast, the modified algorithm does the projection one at a time, as each new orthogonal basis vector is available. Thus, the portion of the previous floating-point error oriented in the direction of the new basis vector gets removed, resulting in a significantly slower growth of error.

As an aside, if you're familiar with QR factorization of a matrix, you can interpret Gram-Schmidt in that way. The more numerically stable algorithms come more from that perspective, such as Householder Reflections.

In the Jupyter notebook, we generate random matrices with different condition numbers and plot a measure of orthogonality as a function of condition number. Please include the plot in the write-up.

Images

This section will be entirely within the Jupyter notebook. It is primarily a (hopefully) interesting demo of orthonormal bases.

References

[1] Lloyd N. Trefethen and David Bau, *Numerical Linear Algebra*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.