MAT 271: Applied & Computational Harmonic Analysis Comments on Homework 3

Problem 2: Almost everyone got Part (a) correct. However, some people did not give enough explanations. You need to *explain* and *justify* your derivations!

For Part (b), what you really need to show is:

1. Compute the eigenvalues of the covariance matrix Γ, which is 0 and $1/n$ with geometric multiplicity 1 and $n-1$, respectively, which can be done by explicitly deriving the characteristic equation, $\det(\Gamma - \lambda I) = \lambda(1/n - \lambda)^{n-1} = 0$. Not too many people did this way, but in my opinion, this should be the easiest way to conclude this. Please review your linear algebra textbooks, in particular, how to compute the determinant of a given matrix, and how to simplify the computation!
2. Γ is a real symmetric matrix; thus it can be *unitarily* diagonalizable, i.e., there exists an orthonormal basis diagonalizing Γ.
3. The above two also means that the eigenspace corresponding to the eigenvalue 0 and the one corresponding to $1/n$ are orthogonal.

For more information and the origin of the interest of this process, please read my own paper [1], [8], [7], and references therein.

Problem 3: Almost everyone got Part (a) correct. However, several people did Part (b) unsatisfactorily, i.e., those of you simply substituted $\phi_k(t) = \sqrt{2}\sin(k\pi t)$ into the integral equation, computing the eigenvalues, and claimed they are the eigenfunctions. With this argument, you cannot be sure whether there exists other eigenfunctions. The correct argument is to derive the eigenvalue problem in the ordinary differential equation from the integral equation, that is:

$$\begin{aligned}
\lambda\phi(t) &= \int_0^1 \Gamma(t,s)\phi(s)\,\mathrm{d}s \\
&= \int_0^1 (\min(t,s) - ts)\phi(s)\,\mathrm{d}s \\
&= \int_0^t (s - ts)\phi(s)\,\mathrm{d}s + \int_t^1 (t - ts)\phi(s)\,\mathrm{d}s \\
&= \int_0^t s\phi(s)\,\mathrm{d}s - t\int_0^t s\phi(s)\,\mathrm{d}s + t\int_t^1 (1-s)\phi(s)\,\mathrm{d}s
\end{aligned}$$

Now, differentiating both sides with respect to t leads to the following ODE:

$$\phi''(t) = -\frac{1}{\lambda}\phi(t)$$

The boundary condition can be derived by setting $t = 0$ and $t = 1$ in the above integral equation. It turns out to be the *Dirichlet* boundary condition:

$$\phi(0) = \phi(1) = 0.$$

From these, we can derive the desired solution. Also, note that you need to justify that $\lambda > 0$ is of our only interest, and the $\lambda \leq 0$ case does not provide us the eigenfunctions.

For Part (c), some people used `svd` instead of `eig` and claimed that `svd` gave them better or closer eigenvectors to the analytical ones compared to `eig`. OK, why does this happen? It's a good exercise to think about it!

Also, note that the eigenvectors are not always uniquely determined for a given matrix. There is always an uncertainty about its signum. In other words, if $\boldsymbol{\phi}$ is an eigenvector, you may get $-\boldsymbol{\phi}$ depending on what software package you use.

Finally, some of you used `cov` function of MATLAB erroneously. This is very important: *MATLAB assumes the row vectors of a data matrix* `X` *are the realizations (or observations), which follows the convention in Statistics. We have been following the convention of the Signal Processing literature, i.e., the column vectors are the realizations/observations.* Hence, if you followed Problem 3's setup, you need to run `cov(X')` instead of `cov(X)`.

For more information about this stochastic process, please read the following papers [2], [3], [5, p. 19], [9].

Problem 4: Here, I would like to point out some major mistakes several people made.

- In general, you should avoid `for` loops as much as possible in MATLAB. In particular, you should not use double or triple `for` loops because they are very costly. That's the difference between the interpretive language like MATLAB and the lower-level languages such as C or Fortran. Instead, try to use the vector and array operations provided by MATLAB as much as possible. For example, in order to compute a linear combination `a(1)*W(:,1) + ... + a(n)*W(:,n)`, do simply `W * a` or `W(:,1:n)*a(1:n)` (matrix-vector multiplication) instead of using any `for` loops.
- Also, several of you used the iteration variable `i` in a `for` loop, i.e., `for i=1:n`, etc. You should not use `i` as a variable in MATLAB! It's reserved for the imaginary unit $\sqrt{-1}$. Use, `k, m, n, p, q, r` instead. Also note that as a variable, `l` (ell) is not too good to use since it is quite close to the numeral `1` (one). Hence it may create potential confusion and misunderstandings.
- Several people treated each face as a matrix, and a set of faces as 3D array, which make many procedures and computations more cumbersome than necessary. It is much easier on MATLAB to treat each face as a vector of length 128^2 and a training dataset as a matrix of size $128^2 \times 72$. Then you can always convert a resulting vector after processing (e.g., reconstruction or approximation) as a matrix by the `reshape` command, which you can use to display as a face.

- Suppose X is a training dataset (matrix) of size $128^2 \times 72$. Many people computed the mean or the average face either by `mean(X')'` or `sum(X')'/N`, but you can do this by one shot: `mean(X,2)`, which immediately gives you the average face.
- If you use the function `DCTMTX`, then specifying the lowest 72 frequency DCT coefficients are trickier than using `DCT2`. Several people used $72^2 = 5184$ coefficients. That's why the DCT reconstructions were so good for some of you. Here you need to keep the lowest 72 DCT coefficients of each and zero out the rest. How to specify the 72 lowest frequency coefficients? Let k_1 and k_2 be the indices of the output DCT coefficient matrix, i.e., k_1, k_2 both ranges from 1 to 128. Hence clearly, you should choose 72 pairs of (k_1, k_2) sorted by the value $\sqrt{k_1^2 + k_2^2}$ in a non-increasing order. There are some ties, but it is OK to pick the first 72 of the output of `sort` function. Clearly, one should not use $\{(1,1), \ldots, (72,1)\}$ or $\{(1,1), \ldots, (1,72)\}$.
- Suppose $\widetilde{X}$ is the data matrix after subtracting the mean column vector (i.e., the mean face) from each column vector. $\widetilde{X}$ can be quickly computed by the following MATLAB construct: `X-mean(X,2)*ones(1,N)` or `X-repmat(mean(X,2),1,N)`.

  Now, the sample covariance matrix is $\widehat{\Gamma} := (1/N)\widetilde{X}\widetilde{X}^{\mathsf{T}}$. (Note that several people forgot the factor $1/N = 1/72$ here.) Then, suppose the SVD of $\widehat{\Gamma}$ is $\widehat{\Gamma} = U\Sigma V^{\mathsf{T}}$. Then, as I mentioned in the class, $(1/\sqrt{N})\widetilde{X}V$ provides the first N KLB vectors of $\widehat{\Gamma}$. But you need to be careful. The column vectors of $(1/\sqrt{N})\widetilde{X}V$ are orthogonal but clearly not orthonormal! In order to make it orthonormal, you need to divide the kth column vector by σ_k or the norm of that column.

  Another simple way to compute the top N KLB is to use the MATLAB function `svds` as follows: `[U, S, V] = svds(Xtilde/sqrt(N), N);` where `Xtilde` represents $\widetilde{X}$ in MATLAB of course. Then, the column vectors of `U` is the KLB, which are orthonormal.
- It is very important to know that the MATLAB `eig` function sorts the eigenvalues and eigenvectors in the *increasing order*, i.e., from the smallest to the largest. Thus, to use the top k KLB vectors means that you need to use the last k KLB vectors in the KLB matrix if you do not reorder it immediately after getting it from `eig`. That is why the relative ℓ^2 curves for KLT were worse than those of DCT for some of you. For those of you made that mistake, I would strongly suggest that you recompute the error curve and plot against those of the DCT!
- Suppose $\widetilde{X}_{\text{recon}}$ be the reconstructed or approximated version of $\widetilde{X}$. Then, the relative ℓ^2 error of the kth face is defined as

  $$Relerr(k) = \frac{\|\widetilde{X}(:,k) - \widetilde{X}_{\text{recon}}(:,k)\|}{\|\widetilde{X}(:,k)\|}, \quad k = 1, \ldots, N,$$

  from which you can compute the average error easily by `mean(Relerr)`.
- Several of you who treated each face as a matrix of size 128×128 compute the residual errors using the `norm` function without converting them to vectors. It is very important

to realize that if you supply a matrix to `norm`, by default, it computes its L^2 *matrix norm*, which is quite different from the L^2-norm of a vector. Please review your numerical linear algebra textbooks!

- The other thing I want to point out is that *one should use the inverse transform routines to compute the basis functions*. Note that if the input signal is one of the basis functions/vectors, then the output is one of the standard basis vector. This means that if you apply the inverse transform to the identity matrix, you get all the basis functions. Thus, use `IDCT2` to compute the DCT basis vectors! That's much faster and nicer than the code segments some of you wrote.
- Finally, the most important thing I wanted to convey to you by this problem is the following: The KLB is an excellent tool for compressing the training dataset, but not necessarily for the test dataset unless the covariance matrix of the test dataset is the same as or very close to that of the training dataset. With a relatively small number of the signals in the training and test datasets, this usually won't happen. On the contrary, the DCT performs on both the training and test datasets in the same way. There should be no essential difference between its performance on the training dataset and that on the test dataset.

For more information about this dataset, please read the following papers [4], [6].

Problem 5: Several of you stated that the function $g_{x_0,\xi_0}(\cdot)$, i.e., the translated and *modulated* version of the window function $g(\cdot)$, is real-valued. It's wrong! g is real-valued but not g_{x_0,ξ_0} due to the modulation $\mathrm{e}^{2\pi \mathrm{i}\xi x}$.

References

[1] B. BÉNICHOU AND N. SAITO, *Sparsity vs. statistical independence in adaptive signal representations: A case study of the spike process*, in Beyond Wavelets, G. V. Welland, ed., vol. 10 of Studies in Computational Mathematics, Academic Press, San Diego, CA, 2003, ch. 9, pp. 225–257.

[2] J. B. BUCKHEIT AND D. L. DONOHO, *Time-frequency tilings which best expose the non-Gaussian behavior of a stochastic process*, in Proc. International Symposium on Time-Frequency and Time-Scale Analysis, IEEE, 1996, pp. 1–4. Jun. 18–21, 1996, Paris, France.

[3] D. L. DONOHO, M. VETTERLI, R. A. DEVORE, AND I. DAUBECHIES, *Data compression and harmonic analysis*, IEEE Trans. Inform. Theory, 44 (1998), pp. 2435–2476. Invited paper.

[4] M. KIRBY AND L. SIROVICH, *Application of the Karhunen-Loève procedure for the characterization of human faces*, IEEE Trans. Pattern Anal. Machine Intell., 12 (1990), pp. 103–108.

[5] Y. MEYER, *Oscillating Patterns in Image Processing and Nonlinear Evolution Equations*, vol. 22 of University Lecture Series, Amer. Math. Soc., Providence, RI, 2001.

[6] N. Saito, *Image approximation and modeling via least statistically dependent bases*, Pattern Recognition, 34 (2001), pp. 1765–1784.

[7] ——, *The generalized spike process, sparsity, and statistical independence*, in Modern Signal Processing, D. Rockmore and J. D. Healy, eds., vol. 46 of MSRI Publications, Cambridge University Press, 2004, pp. 317–340.

[8] N. Saito and B. Bénichou, *The spike process: a simple test case for independent or sparse component analysis*, in Proc. 3rd International Conference on Independent Component Analysis and Signal Separation, T.-W. Lee, T.-P. Jung, S. Makeig, and T. J. Sejnowski, eds., IEEE, 2001, pp. 698–703. Dec. 10–12, 2001, San Diego, CA.

[9] N. Saito, B. M. Larson, and B. Bénichou, *Sparsity and statistical independence from a best-basis viewpoint*, in Wavelet Applications in Signal and Image Processing VIII, A. Aldroubi, A. F. Laine, and M. A. Unser, eds., vol. Proc. SPIE 4119, 2000, pp. 474–486. Invited paper.