Appendix: Applying gEDMD to Financial Economics

Preston Rozwood
Department of Computer Science
Cornell University
pwr36@cornell.edu

Edward Mehrez (Not Enrolled)
Department of Economics
Cornell University
ejm322@cornell.edu

This appendix was created as an extension of the report submitted for the course project. It is meant to explain how EDMD can be formulated in RKHS with kernel functions as well as describe more details regarding our implementation of gEDMD, including any problems we ran into and possible solutions to getting a better model from our application of the Koopman generator operator in the future.

EDMD in terms of RKHS

To discuss how to frame EDMD in terms of RKHS we first introduce some definitions. As our guide, we follow Klus et al. (2020) closely.

1. RKHS Definition:
   Let $\mathbb{X} \subset \mathbb{R}^d$ be a set and $\mathbb{H}$ be the space of functions $f : \mathbb{X} \to \mathbb{R}$. Then $\mathbb{H}$ is called a reproducing kernel Hilbert space (RKHS) with the corresponding inner product $\langle \cdot, \cdot \rangle$ if there is a function $k : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$ s.t.
   - $$\langle f, k(x, \cdot) \rangle_H = f(x), \quad \forall f \in \mathbb{H}$$
   - $\mathbb{H} = \overline{span\{k(x, \cdot) | x \in \mathbb{X}\}}$

   k is called a reproducing kernel of $\mathbb{H}$. As one connection which will be useful below, we note that we can treat $k(x, \cdot)$ as a feature map $\phi(x)$ of x in $\mathbb{H}$ s.t.
   $$k(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathbb{H}} \tag{1}$$
   A function k is a reproducing kernel if and only if it is symmetric and positive definite. Note that most kernel-based learning algorithms rely on computations involving only Gram matrices evaluated on a finite number of data points.
2. Covariance operators
   Let (X, Y) be a random variable on $\mathbb{X} \times \mathbb{Y}$ with corresponding marginal distributions $\mathbb{P}(X)$ and $\mathbb{P}(Y)$, respectively, and joint distribution $\mathbb{P}(X, Y)$. Let ϕ and ψ be feature maps associated with the bounded kernels k and l, respectively. Let $\mathbb{H}$ and $\mathbb{G}$ be the RKHSs associated with the kernels k and l, respectively. THen the covariance operator $\mathcal{C}_{XX} : \mathbb{H} \to \mathbb{H}$ and the cross-covariance operator $\mathcal{C}_{YX} : \mathbb{H} \to \mathbb{G}$ are defined as
   $$\mathcal{C}_{XX} := \int \phi(X) \otimes \phi(X) \mathrm{d}\mathbb{P}(X) = \mathbb{E}_X[\phi(X) \otimes \phi(X)],$$
   $$\mathcal{C}_{YX} := \int \phi(Y) \otimes \phi(X) \mathrm{d}\mathbb{P}(Y, X) = \mathbb{E}_{YX}[\psi(Y) \otimes \phi(X)].$$

For a given set of basis functions, $\phi_1, ..., \phi_m$, we define the vector valued function $\phi = (\phi_1, ..., \phi_m) : \mathbb{R}^d \to \mathbb{R}^m$. These are the same as the feature maps mentioned above. The resulting feature matrices

$$\Phi = [\phi(x_1) \cdots \phi(x_n)]$$
$$\Psi = [\phi(y_1)\phi(y_n)]$$

where the y series can be thought of as the shifted x series. Finally, the Koopman operator as estimated with EDMD can be expressed as

$$\widehat{K} = \widehat{C}_{YX}\widehat{C}_{XX}^{-1} \tag{2}$$

where $\hat{C}_{XX} = \frac{1}{n}\Phi\Phi^T$ and $\hat{C}_{YX} = \frac{1}{n}\Psi\Phi^T$. As we mentioned in the main paper, this can be interpreted as a least-square approximation of the Koopman operator using transformed data matrices.

Finally to see the relationship with the Koopman generator note that

$$Lf := \lim_{t \downarrow 0} \frac{K_t f - f}{t} \tag{3}$$

and it can be shown that in terms of the operators

$$\mathcal{K}_t = \exp(t\mathcal{L}) \tag{4}$$

Thus, using this and the connection with kernels given in (2) we can get an approximation of the Koopman generator as

$$\widehat{L} = \frac{1}{t}\log\widehat{K} \tag{5}$$

Implementation Details

Before looking into implementation specifics, we thought it was important to note that our BSM was based on the “most recent" 10% of the dataset, just as our Koopman model from the report was.

1. **Empirical Models and their Residuals**

   We modeled the (instantaneous) return vector dr_t as the following

   $$dr_t := \frac{dP_t}{P_t} = b(r_t)dt + \sigma(r_t)dW_t \tag{6}$$

   where, as discussed in the paper, W_t is a s-dimensional Wiener process and $b(r) \in \mathbb{R}^d$ and $\sigma \in \mathbb{R}^{d\times s}$ are the state dependent drift vector and volatlity matrix of returns. In both the gEDMD and BSM implementations, we derived an expression for residuals ϵ_t by discretizing our SDE

   $$r_t := \frac{\Delta P_t}{P_t} = b(r_{t-1})\Delta t + \sigma(r_{t-1})\Delta W_t \tag{7}$$
   $$= b(r_{t-1}) + \sigma(r_{t-1})\epsilon_t \tag{8}$$

   Note that our time step Δt is 1 and we are evaluating the drift and volatility at the returns from the previous period. $\epsilon_t := W_t - W_{t-1}$ is called the white noise vector and is distributed standard multivariate normal. Solving for the white noise vector

   $$\epsilon_t = \left[\sigma^T(r_t)\sigma(r_t)\right]^{-1}\sigma^T(r_t)[r_t - b(r_{t-1})] \tag{9}$$

   and finally, the fitted model implied residuals $\widehat{\epsilon}$ are

   $$\widehat{\epsilon_t} = \left[\widehat{\sigma}^T(r_t)\widehat{\sigma}(r_t)\right]^{-1}\widehat{\sigma}^T(r_t)[r_t - \widehat{b}(r_{t-1})] \tag{10}$$

   Testing either the BSM or our model boils down to testing if the exterior (exogenous) assumption of white noise is correct. Thus, we test if the fitted model implied residuals are multivariate normal. The BSM model assumes that the drift vector μ and volatility matrix σ are constant throughout time whereas the gEDMD model allows for state dependent drift and volatility. Note that we take the state to be the full return vector in our implementation.

2. **Detailed Derivation of covariance matrix** a

   (a) Set the observable, $g(x)(\in \mathbb{R}^w) =$ second order monomials

(b) Find $B \in \mathbb{R}^{s \times w}$ s.t. $B^\top \psi(x) = g(x)$

(c)

$$\begin{aligned}\mathcal{L}g(x) &= \nabla g(x) b(x) + \text{vec}(a) \text{ where vec}(a) \text{ is a vectorized (flattened) a matrix.}\\ &= \nabla(B^\top \psi(x)) b(x) + \text{vec}(a)\\ &= B^\top \nabla \psi(x) b(x) + \text{vec}(a) (\text{we also note that the order of vec}(a) \text{ matches the order of } g(x))\end{aligned}$$

Using $\mathcal{L}g(x) \approx (\widehat{L}B)^\top \psi(x)$

$$\implies \text{vec}(a) \approx (\widehat{L}B)^\top \psi(x) - B^\top \nabla \psi(x) \widehat{b}(x)$$

(d) Given an estimate for $\widehat{b}(x)$, we can estimate $a(x)$

$$\widehat{a} = (\widehat{L}B)^\top \psi(x) - B \nabla \psi(x) \widehat{b}(x)$$

3. **Issues that arose**

A big issue that we noticed late in the development of this codebase was that Python/NumPy was giving strange behavior regarding matrix multiplications and various other operations, particularly when using vectors. Vectors have a strange property in NumPy where their shape does not allow them to use the standard transpose function to get their transpose. While the dimensions appeared to be correct, the numbers were way off as a result of this strange behavior, hence why we did not notice earlier in our development. We believe that we were able to remedy all the related problems, but are not certain.

Another issue that arose is that when attempting to use the standard Cholesky decomposition to solve for σ after obtaining a (where $a = \sigma\sigma^\top$), an error was thrown saying that the given matrix was not positive definite. We should expect to have retrieved a positive definite a, but did not. To get around this issue, we used an alternative approach using SVD (where we assumed that our a matrix was non-negative definite?). This, unfortunately, could have led to some of the computational errors we witnessed in the results.

4. **Possible fixes**

A potential fix to ensure positive semidefinite-ness could be that rather than using gEDMD, we could reframe problem in RKHS setting and then take advantage of existing kernel tools which ensure that the covariance estimates are still positive definite. With this, we could do the proper Cholesksy decomposition as previously mentioned which would hopefully result in better modelling.

5. **Computational complexity**

In the code, it was clear that the Koopman generator was not the bottleneck of our code. Specifically, it was the portion of the code dedicated to computing the ϵ values from our model. This indicates that the highly complex computations we saw were not within the model generation itself, but rather, in the testing of the model. This means that Koopman theory still has practical application in learning models, however, testing needs to be revisited and improved greatly for it to become a standard technique in machine learning.

References

Klus, S., Schuster, I., and Muandet, K. (2020). Eigendecompositions of transfer operators in reproducing kernel hilbert spaces. *Journal of Nonlinear Science*, 30(1):283–315.