Notes on resolvent modes from simulation data using Hebbian updates

A S Sharma

June 16, 2022

1 Singular value decomposition using Hebbian updates

Based on [1], the following algorithm will generate the singular vectors of a matrix or linear operator $R : \mathbf{F} \to \mathbf{U}$. The notation in that paper is a bit confusing so I will try to summarise and clarify it here.

Let $u \in \mathbf{U}$ and $f \in \mathbf{F}$ be a matching data vector pair satisfying $u = Rf$. The inner product in $\mathbf{U}$ is denoted $\langle x, y \rangle_{\mathbf{U}}$ and similarly for $\mathbf{F}$.

Let c_i^{u*} be the true ith left singular vector of R to be found and c_i^{f*} be the corresponding true right singular vector. Let c_i^u be the current approximation to c_i^{u*} and define c_i^f similarly. The notation Δc_i^u indicates an update (change) to the approximation c_i^u.

According to [1] (eqs 19-20), updates of c^u and c^f may be performed using

$$\Delta c_i^u = \left\langle c_i^f, f \right\rangle_{\mathbf{F}} (u - \sum_{j<i} \left\langle u, c_j^u \right\rangle_{\mathbf{U}} c_j^u), \tag{1}$$

$$\Delta c_i^f = \langle c_i^u, u \rangle_{\mathbf{U}} (f - \sum_{j<i} \left\langle f, c_j^f \right\rangle_{\mathbf{F}} c_j^f). \tag{2}$$

My understanding is that the approximations c_i^u and c_i^f are proven to converge to the true singular vectors of R given 'enough' iterations and data vectors (u, f) that span $(\mathbf{U}, \mathbf{F})$.

Note that we do not need direct access to R — only enough data vector pairs. This opens up the possibility of using random vectors, or even direct numerical simulation datasets, with low storage and computational requirements.

2 Pseudocode

Data: $A \in \mathbb{R}^{N\times M}$
Result: $U \in \mathbb{R}^{N\times L}$, $V \in \mathbb{R}^{M\times L}$
initialise U, V ;
initialise $\eta_0 \ll 1$, $d \ll 1$;
for $n = 1$ **to** n_{max} **do**
  Data: snapshot pair $a \in \mathbb{R}^N$, $b \in \mathbb{R}^M$
  $\eta \leftarrow \eta_0 \mathrm{e}^{-dn}$
  $y_a \leftarrow U^H a$
  $y_b \leftarrow V^H b$
  $U \leftarrow U + \eta\ (a y_b^H - U\ \mathrm{triu}(y_a y_a^H))$
  $V \leftarrow V + \eta\ (b y_a^H - V\ \mathrm{triu}(y_b y_b^H))$
end

Algorithm 1: SVD algorithm of [1], as implemented in the example code in this project. Superscipt H indicates conjugate transpose, and a and b are column vectors, so that (for example) $y_a y_a^H$ is an outer product resulting in a $L \times L$ matrix. triu(X) indicates setting to zero all but the upper triangular part of a matrix X (tril also works).

3 Application to linearised code

In the case that we have access to a linearised code we would still need to form R in order to generate test vector pairs. This is a slight improvement on the loop in the algorithm of [2].

4 Application to nonlinear turbulent simulations

What really got my attention, however, is the possibility of using this algorithm on-line with data generated from a turbulent simulation.

To see why this is possible, recall that the resolvent formulation used in [3] of the Navier-Stokes equations has to parts. The linear part is (in the frequency-domain)

$$u(\omega) = R(\omega) f(\omega) \tag{3}$$

and the nonlinear part is

$$f(t) = u(t) \cdot \nabla u(t). \tag{4}$$

Both must be true simultaneously and always.

Since the updates require only the pairs (u, f), instead of generating u from f via the resolvent using (3), we can generate f from u via their nonlinear relationship (4). The frequency-domain pairs $(u(\omega), f(\omega))$ can then be generated from snapshot matrices of $u(t)$ and $f(t)$.

To do this, first form the snapshot matrix U_1,

$$U_1 = \begin{bmatrix} u(t_1) & u(t_2) & u(t_3) & \dots & u(t_N) \end{bmatrix}.$$

Its discrete Fourier transform is the matrix of frequencies which form a set (across frequencies) of snapshots in $\mathbf{U}$.

$$F_1 = DFT(U_1) = \begin{bmatrix} u(\omega_1) & u(\omega_2) & u(\omega_3) & \dots \end{bmatrix}.$$

The frequencies that can be resolved are determined by the timestep and N. Similarly, form the corresponding snapshot matrix F_1, with $DFT(F_1)$ giving a right test vector per frequency in $\mathbf{F}$.

Inspired by Welch's method, we can generate the next snapshot matrix by shifting time by one (or more) steps,

$$U_2 = \begin{bmatrix} u(t_2) & u(t_3) & u(t_4) & \dots & u(t_{N+1}) \end{bmatrix}.$$

The next set of test vector pairs (one per frequency) is given by the DFT of U_2 and F_2, and so on.

This process gives one test vector pair per frequency, per snapshot matrix and can be fed to (1) until convergence.

5 Other approaches

I haven't read [4] but it and following work may be related.

References

[1] G. Gorrell, "Generalized Hebbian algorithm for incremental singular value decomposition in natural language processing," in *EACL 2006,*

11st Conference of the European Chapter of the Association for Computational Linguistics, Proceedings of the Conference, April 3-7, 2006, Trento, Italy (D. McCarthy and S. Wintner, eds.), The Association for Computer Linguistics, 2006.

[2] J. Houtman, S. Timme, and A. Sharma, *Resolvent Analysis of Large Aircraft Wings in Edge-of-the-Envelope Transonic Flow*.

[3] B. J. McKeon and A. S. Sharma, "A critical-layer framework for turbulent pipe flow," *Journal of Fluid Mechanics*, vol. 658, p. 336–382, July 2010.

[4] A. Towne, T. Colonius, P. Jordan, A. Cavalieri, and G. Brès, "Proceedings of 21st aiaa/ceas aeroacoustics conference," 2015.