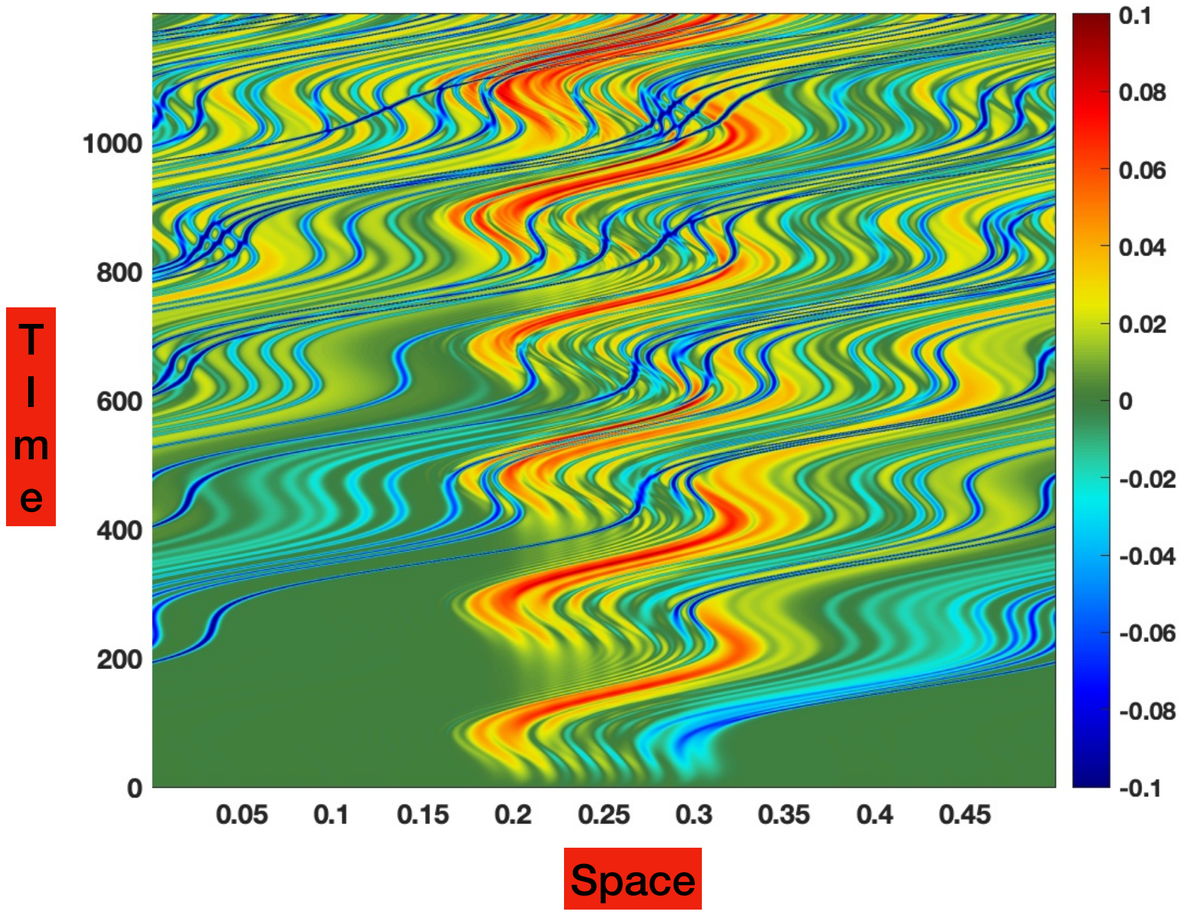
Quantitative Climate Science: Numerical Linear Algebra a Practitioners Guide

Marek Stastna Fall 2024

EOF challenges (from last slide deck)

- If we want to apply EOFs and the Error map method to our data set we have some numerical challenges to consider.
- Recall that the data set is 2048 points in space by 1200 points in time. So the resulting covariance matrix is 2048x2048 and is full.
- That is not especially large, but it will take some time to find all those eigenvectors and that seems wasteful if we are only interested in say 10-20% of them.
- So our goal for this slide deck is to explore methods that find only a subset of eigenvalues and eigenvectors.



Linear Algebra 1

- Linear algebra concerns matrix objects and operations on them.
- Classical linear algebra typically builds out from vectors, $v_i = \mathbf{v} = \vec{v}$ which are matrices with n rows, and 1 column and square matrices $A_{ij} = \mathbf{A}$ with n rows and n columns.
- Vectors have one basic operation, namely the inner product: $\vec{v} \cdot \vec{w} = \sum_{i=1}^{n} v_i w_i$, where the "size", or norm, of a vector is defined as $\|\mathbf{v}\|^2 = \vec{v} \cdot \vec{v}$. This requires n multiplications.
- Matrix multiplication, $A_{ij} v_j = \mathbf{A}\mathbf{v}$ is an operation which can be reduced to repeated dot products so that each of the n rows is dotted with the vector $\vec{v}$.
- Most of the time we simply don't think about the cost of these operations, but in some cases $\mathbf{A}$ can get huge.

Exercise: what's an efficient way to compute $\mathbf{ABx}$ where $\mathbf{A}$ and $\mathbf{B}$ are square matrices?

Linear Algebra 2

- The case of a large $\mathbf{A}$ starts to matter when we set out to do something non-trivial like solving a set of linear equations, $\mathbf{Ax} = \mathbf{b}$.
- The mathematician in you may be tempted to write $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ however in practice, finding the inverse $\mathbf{A}^{-1}$ does not have a stable algorithm (this isn't immediately obvious).
- Even if it was possible to compute the inverse, we would have to ensure that $\mathbf{A}$ is invertible, or that the determinant was non-zero, $\det \mathbf{A} \neq 0$.
- When we are taught to solve $\mathbf{Ax} = \mathbf{b}$ by hand we are typically taught to do Gaussian elimination by hand (see https://en.wikipedia.org/wiki/Gaussian_elimination for more details)
- The problem with Gaussian elimination is that it costs an astonishing O(n^3) operations, meaning that as matrices get larger it becomes out of the question.

Linear Algebra 3

- So how can we avoid O(n^3) operations? Well, we could dig into the mathematical structure of matrices and try to come up with ways to rewrite $\mathbf{Ax} = \mathbf{b}$ in terms of simpler problems.
- A famous example is called the L-U decomposition where $\mathbf{Ax} = \mathbf{LUx} = \mathbf{b}$ where $\mathbf{L}(\mathbf{U})$ is lower(upper) triangular.
- If we can rewrite $\mathbf{A}$ like the above then we just have to solve two simpler problems, $\mathbf{Ly} = \mathbf{b}$, followed by $\mathbf{Ux} = \mathbf{y}$. Because $\mathbf{L}(\mathbf{U})$ are lower(upper) triangular these problems are super easy to solve (and we expect a significant savings if we don't store all those zeros!).
- The question thus becomes whether the so-called L-U decomposition can be efficiently carried out.
- A related useful decomposition writes $\mathbf{Ax} = \mathbf{QRx}$ where $\mathbf{Q}(\mathbf{R})$ are orthogonal and upper triangular matrices.
- Most standard packages have ways of computing standard decompositions like LU and QR.

Linear Algebra 4

- An alternative to the matrix decompositions described on the last slide is to consider $\mathbf{Ax} = \mathbf{b}$ as a different problem, one that can be approached iteratively.

- To see how that may work, recall Newton's method in 1D. You want to solve $f(x) = 0$ for a differentiable function $f(x)$. You start with a guess, x_0, compute the linear approximation at that point, $\mathscr{L}_{x=x_0}(x) = f(x_0) + f'(x_0)(x - x_0)$ and set that to zero to get the next guess: $x_1 = x_0 - \dfrac{f(x_0)}{f'(x_0)}$. Then repeat the process.

- If we write it in general we get $x_{k+1} = x_k - \dfrac{f(x_k)}{f'(x_k)}$. The first term is just the previous guess, and the second piece is a correction.

- The form of the correction is useful for more than just the formula, since it tells you when you expect things to go wrong, namely when $|f'(x_k)| \ll 1$.

Linear Algebra 5

- For vector valued functions of more than one variable Newton's method generalizes but gets more complex.
- Recall that the Jacobian matrix is defined as $A_{ij} = \frac{\partial f_i}{\partial x_j}$ and the Jacobian determinant is the determinant of this matrix. The linear approximation is then wriiten as $\mathscr{L}_{x^{(i)}=x_0^{(i)}}(x^{(i)}) = f_j(x_0^{(i)}) + A_{ij}(x^{(i)} - x_0^{(i)})$
- And the process of getting the next guess, which was $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$ in 1D is now a matrix solve.
- The condition of where the approximation breaks now has to do with the properties of the matrix (and with more dimensions may be more complicated; think a steep, narrow valley that gently slopes towards a minimum).

Linear Algebra 6

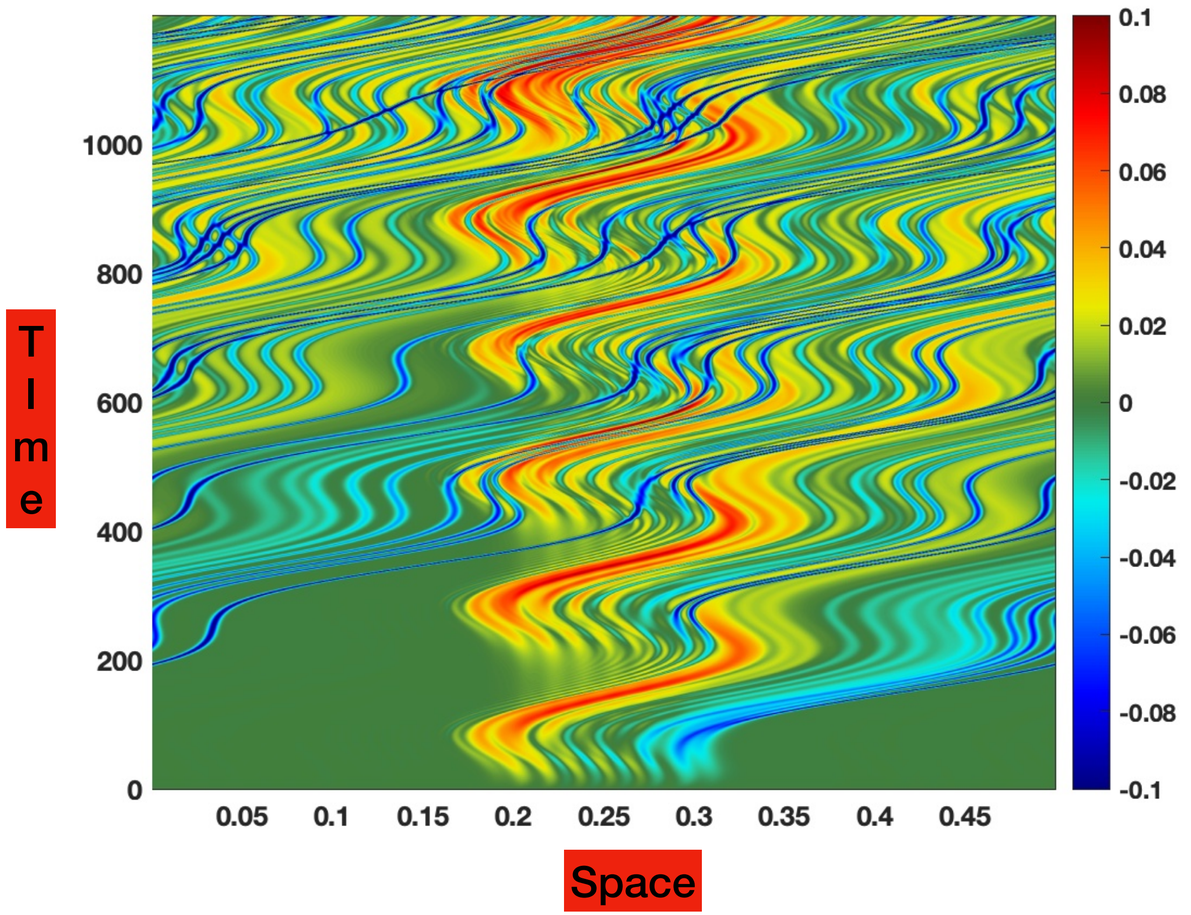
- While Newton's method certainly motivates a large number of methods in numerical linear algebra. It is actually a much more basic method, iteration, that is instructive.
- Say $f(x) = x^2$ and pretend you don't know how to factor to solve $f(x) = 0$. You could start with a guess, x_0, then define $x_1 = f(x_0)$, $x_2 = f(x_1) = f(f(x_0))$, $x_k = f^k(x_0)$ where the superscript k denotes the number of times to apply f in succession.
- For arbitrary x_0 this is a terrible idea, but when $|x_0| < 1$ it works remarkably well.
- So what about $\mathbf{Ax} = \mathbf{b}$? Well we want to write $\mathbf{A} = \mathbf{M} - \mathbf{N}$ where $\mathbf{M}$ is "easy to invert". If this is the case then write an iteration problem as: $\mathbf{Mx^{(k+1)}} = \mathbf{Nx^{(k)}} + \mathbf{b}$
- Many classical methods fit into this class, with more details here:
- https://en.wikipedia.org/wiki/Iterative_method

Linear Algebra 7

- But what we set out to talk about was the idea of computing a subset of eigenvalues and eigenvectors.
- To get at that, we build on the idea of iteration in a different way. We build a set of vectors $K_n = \left\{ \mathbf{b}\ \mathbf{Ab}\ \mathbf{A^2b} \ldots \mathbf{A^{n-1}b} \right\}$
- These vectors are independent, and we can use standard algorithms to get an orthonormal set (https://en.wikipedia.org/wiki/Arnoldi_iteration)
- We can then build a method to compute eigenvalue/eigenvector approximations using K_n.
- This is the idea behind the function “eigs: in Matlab.
- https://www.mathworks.com/help/matlab/ref/eigs.html

EOF challenges (now with perspective)

- Recall that the data set is 2048 points in space by 1200 points in time. So the resulting covariance matrix is 2048x2048 and is full.
- In practice we can use eigs to look at something like 150 eigenvalues and get an answer in a minute or less on a laptop.
- If the scree plot looks to go to zero quickly we are in business.
- If things look dodgy we grit our teeth and go up to 250 and compute again.
- The whole point of data centric methods is to be iterative in application!



Linear Algebra 8

- OK, now that we have the idea of how we might go about finding a subset of eigenvalues let's consider a different shortcut.

- Recall that we started with a data matrix $\mathbf{X}$ that was not necessarily square. So can we get the same info without building the covariance matrix?

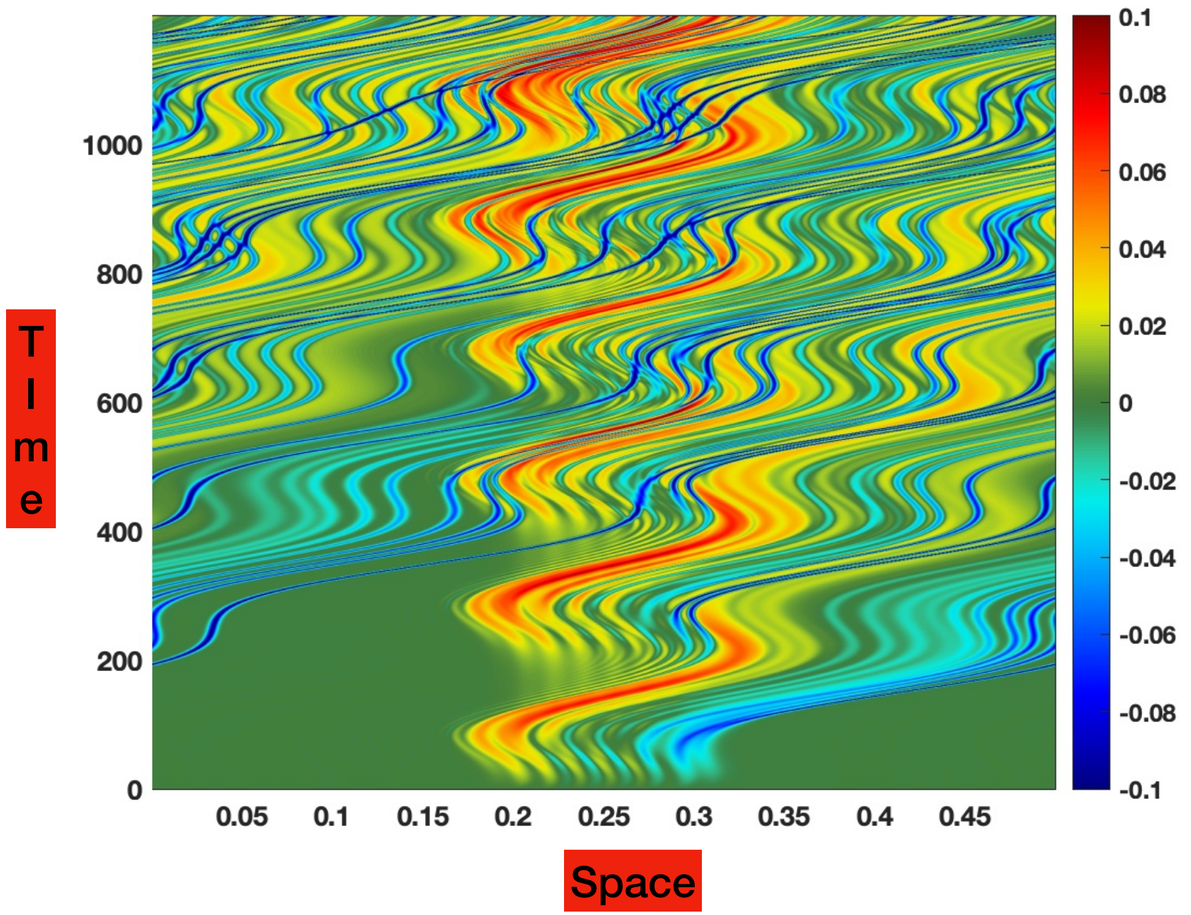
- The answer is a qualified yes. Modern linear algebra gives that a general matrix, like $\mathbf{X}$ can be decomposed via the singular value decomposition as:
  $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$ where $\mathbf{\Sigma}$ is the generalization of the matrix of eigenvalues, and $\mathbf{V}$ is the analogue of the matrix of eigenvectors. $\mathbf{U}$ and $\mathbf{V}$ are square, but $\mathbf{\Sigma}$ is not.

- https://en.wikipedia.org/wiki/Singular_value_decomposition

- The point, for practitioners is that the SVD is built into standard packages like Matlab, and for some applications finding a subset of singular values has algorithms that are more stable than the problem for eigenvalues.

EOF via SVD

- In practice we can use svds to look at something like 150 singular values and the related generalizations of eigenvectotors to get an answer in a minute or less on a laptop.
- I have not really found instances where svds works while eigs fails. Nor have I found one to be faster than the other.
- There might be a reason to prefer one over the other for problems with really large data. For example with a long time series you could do build the covariance matrix without ever bringing ALL the data into memory.



Linear Algebra 9

- Much of the numerical linear algebra from PDE applications concentrates on really big matrices, often with many entries being zero.
- The technical term is “large sparse matrices” and often one wants to find fast, iterative methods for solving $\mathbf{A}x = b$ approximately where $\mathbf{A}$ is a large, sparse matrix.
- Datacentric applications may not naturally fit into the same set of constraints and other techniques thus come to the fore.
- The SVD material a couple of slides ago is such an example.
- Randomized techniques are another example: https://medium.com/quant-guild/parallelizing-randomized-singular-value-decomposition-using-gpus-62dff9e0c945
- In practice it is a bit of a two step dance between trying something, then seeking theoretical understanding of why it does or does not work, then trying an improvement…