Singular Value Decomposition

02671 Data-Driven Methods for Computational Science and Engineering

Allan Peter Engsig-Karup
Scientific Computing Section
DTU Compute
Technical University of Denmark

DTU Compute
Department of Applied Mathematics and Computer Science

Reading

This week, you should read

- Book, Chapter 1.

Objectives of the lecture

- Understand what is Singular Value Decomposition (SVD).
- Understand how to apply SVD to large data sets.
- Understand how to use SVD in code.
- Understand how to construct low-rank approximation using truncated SVD and randomized SVD.
- Understand how to do a physics-informed least-square regression that utilise both data and mathematical models (domain knowledge).

Singular Value Decomposition (SVD)

The singular value decomposition is among the most important matrix factorizations of the computational era.

We are interested in analyzing a large data set $\mathbf{X} \in \mathbb{C}^{n \times m}$

$$\mathbf{X} = \begin{bmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_m \\ | & | & & | \end{bmatrix}$$

where $\mathbf{x}_k \in \mathbb{C}^n$, $k = 1, ..., m$ are data such as measurements from simulations or experiments.

The columns are called *snapshots* and m is the number of snapshots in $\mathbf{X}$.

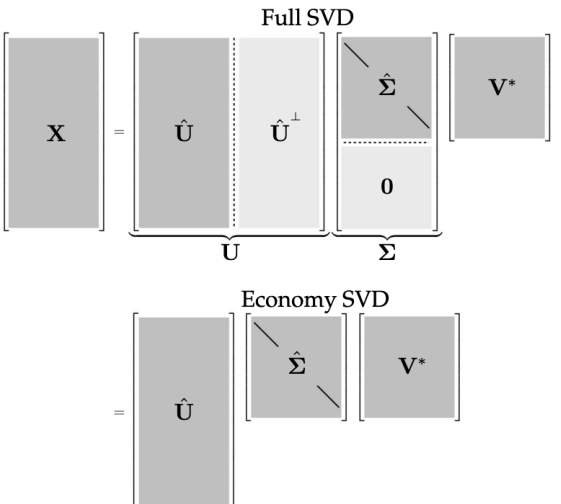
The SVD is a unique matrix decomposition that exists for every complex-valued matrix $\mathbf{X}$ and is given as

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^*$$

where $\mathbf{U} \in \mathbb{C}^{n \times n}$ (left singular vector) and $\mathbf{V} \in \mathbb{C}^{m \times m}$ (right singular vector) are unitary matrices with orthonormal columns, and $\Sigma \in \mathbb{R}^{n \times m}$ (singular values) is a matrix with real, non-negative entries on the diagonal and zeros off the diagonal.

Singular Value Decomposition (SVD)

The SVD decomposes the matrix into a factorization that may be exploited.



Computing the SVD

MATLAB:

```
>> X=randn(5,3);      % Create a 5x3 random data matrix
>> [U,S,V] = svd(X); % Singular value decomposition
>> [Uhat,Shat,V] = svd(X,'econ') % Economy sized SVD
```

Python:

```
>>> import numpy as np
>>> X = np.random.rand(5,3) # Create random data matrix
>>> U, S, VT = np.linalg.svd(X,full_matrices=True) # Full SVD
# Economy SVD
>>> Uhat, Shat, VThat = np.linalg.svd(X,full_matrices=False)
```

Remark, the SVD of an $m \times n$ matrix $\mathbf{X}$ requires $\mathcal{O}(mn^2 + m^3)$ flops (floating-point operations) and memory $\mathcal{O}(mn)$.

Matrix approximation

The SVD can be expressed as a sum of rank-one matrices

$$\mathbf{X} = \sum_{k=1}^{m} \sigma_k \mathbf{u}_k v_k^* = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^* + \sigma_2 \mathbf{u}_2 \mathbf{v}_2^* + \cdots + \sigma_m \mathbf{u}_m \mathbf{v}_m^*$$

where σ_k is the k'th diagonal entry of Σ, and $\mathbf{u}_k$ and $\mathbf{v}_k$ are the k'th columns of $\mathbf{U}$ and $\mathbf{V}$.

If the singular values are ordered in decreasing order as

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_m \geq 0$$

then each new rank-one matrix $\sigma_m \mathbf{u}_m \mathbf{v}_m^*$ is less important than the former one due to this ordering.

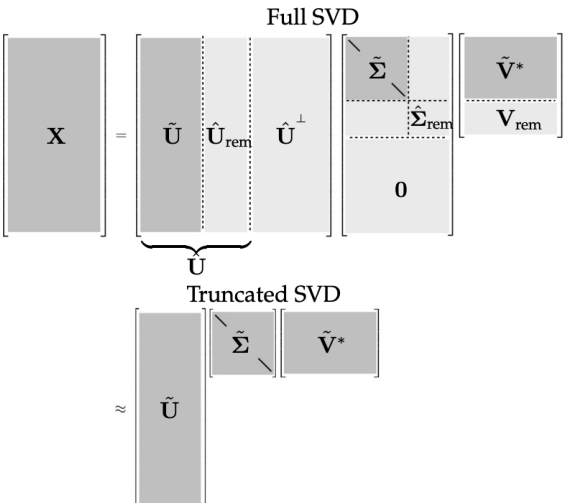
This implies that it is possible to determine an approximation of $\mathbf{X}$ by truncating at some rank r (i.e. $r < m$)

$$\mathbf{X} \approx \tilde{\mathbf{X}} = \sum_{k=1}^{r} \sigma_k \mathbf{u}_k v_k^* = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^* + \sigma_2 \mathbf{u}_2 \mathbf{v}_2^* + \cdots + \sigma_r \mathbf{u}_r \mathbf{v}_r^*$$

This property can be exploited to represent high-dimensional data in terms of a few dominant patterns given by the columns of $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{V}}$. This is a coordinate transformation from a high-dimensional space to a low-dimensional space.

Truncated Singular Value Decomposition

The truncated singular value decomposition may be exploited to represent the data in terms of a few dominant patterns.



Matrix approximation

MATLAB:

```
>> Xtilde = Uhat(:,1:r)*Shat(1:r,1:r)*Vhat(:,1:r)'; % truncated SVD
```

Python:

```
>>> Xtilde = Uhat[:,0:r]*Shat[0:r,0:r]*Vhat[:,0:r]'; % truncated SVD
```

Optimal Approximation

Theorem (Eckart-Young)

The optimal rank-r approximation to $\mathbf{X}$, in a least-squares sense, is given by the rank-r SVD truncation $\tilde{\mathbf{X}}$:

$$\underset{\tilde{\mathbf{X}}, s.t. \quad rank(\tilde{\mathbf{X}})=r}{\arg\min} ||\mathbf{X} - \tilde{\mathbf{X}}||_F = \tilde{\mathbf{U}}\tilde{\Sigma}\tilde{\mathbf{V}}^*$$

Here the Frobenius norm is defined as

$$||\mathbf{X}||_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}|\mathbf{X}_{ij}|^2}$$

In short, the Eckart-Young theorem guarantee that the truncated SVD provides the best matrix approximation of a given rank measured in the Frobenius norm.

Error bounds

It is possible to exactly quantify the *error* of the rank-r SVD approximation

$$||\mathbf{X} - \tilde{\mathbf{X}}||_F^2 = \sum_{k=r+1}^{m} \sigma_k^2$$

It is often useful to consider the relative error

$$\frac{||\mathbf{X} - \tilde{\mathbf{X}}||_F^2}{||\mathbf{X}||_F^2}$$

that may be interpreted as the total fraction of energy that is missing in the approximation to $\mathbf{X}$.

Mathematical properties

SVD is closely related to an eigenvalue problem involving the correlation matrices

$$\mathbf{X}\mathbf{X}^* = \mathbf{U}\begin{bmatrix}\tilde{\Sigma}\\ \mathbf{0}\end{bmatrix}\mathbf{V}\mathbf{V}^*\begin{bmatrix}\tilde{\Sigma} & \mathbf{0}\end{bmatrix}\mathbf{U}^* = \mathbf{U}\begin{bmatrix}\tilde{\Sigma}^2 & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\end{bmatrix}\mathbf{U}^*$$

and

$$\mathbf{X}^*\mathbf{X} = \mathbf{V}\begin{bmatrix}\tilde{\Sigma} & \mathbf{0}\end{bmatrix}\mathbf{U}\mathbf{U}^*\begin{bmatrix}\tilde{\Sigma}\\ \mathbf{0}\end{bmatrix}\mathbf{V}^* = \mathbf{V}\tilde{\Sigma}^2\mathbf{V}^*$$

Hence, $\mathbf{U}$, $\mathbf{V}$ and $\tilde{\Sigma}$ are solutions to the following eigenvalue problems

$$\mathbf{X}\mathbf{X}^*\mathbf{U} = \mathbf{U}\begin{bmatrix}\tilde{\Sigma}^2 & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\end{bmatrix}$$

and

$$\mathbf{X}^*\mathbf{X}\mathbf{V} = \mathbf{V}\tilde{\Sigma}^2$$

This allow us to interpret the columns of $\mathbf{U}$ as eigenvectors of the correlation matrix $\mathbf{X}\mathbf{X}^*$ and the columns of $\mathbf{V}$ as eigenvectors of $\mathbf{X}^*\mathbf{X}$.

Computing the eigendecomposition of correlation matrices

MATLAB:

```
>> X=randn(5,3);      % Create a 5x3 random data matrix
>> [V,S2] = eig(X'*X); % Singular value decomposition
```

Python:

```
>>> import numpy as np
>>> X = np.random.rand(5,3) # Create random data matrix
>>> S2, V = np.linalg.eig(X'*X,full_matrices=True) # Full SVD
```

Method of snapshots

The method of snapshots due to Sirovich (1987) can be used to reduce the cost of computing the first r columns of $\mathbf{U}$ of the SVD without computing the SVD. This can be done by computing the eigendecomposition of $\mathbf{X}^*\mathbf{X}$ first and then compute $\tilde{\mathbf{U}}$ from

$$\tilde{\mathbf{U}} = \mathbf{X}\tilde{\mathbf{V}}\tilde{\Sigma}^{-1}$$

MATLAB:

```
>> X=randn(5,3);     % Create a 5x3 random data matrix
>> [VT,S2] = eig(X'*X); % Singular value decomposition
     % Sort the eigenvalues and eigenvectors in descending order
>> [S2,I]= sort(diag(S2),'descend')
>> VT = VT(:,I);
>> S = diag(sqrt(S2));
     % Compute the first columns of U
>> UT=X*VT/S;
     % Select the first k columns of U
>> UT = UT(:,1:k);
```

Pseudo-Inverse

If an exact truncated SVD of $\mathbf{X} = \tilde{\mathbf{U}}\tilde{\Sigma}\tilde{\mathbf{V}}^*$ is substituted for $\mathbf{X}$ it is possible to invert each of the matrices resulting in the Moore-Penrose left pseudo-inverse

$$\mathbf{X}^\dagger = \tilde{\mathbf{V}}\tilde{\Sigma}^{-1}\tilde{\mathbf{U}}^*$$

This may be utilized to find least-square solutions to linear systems of equations

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

If we take the SVD of $\mathbf{A}$ such that $\mathbf{A} \simeq \mathbf{U}\Sigma\mathbf{V}^*$ then the least-square solution is

$$\mathbf{A}^\dagger\mathbf{A}\mathbf{x} = \mathbf{A}^\dagger\mathbf{b}$$

then we can find the solution from the expression

$$\mathbf{x} = \tilde{\mathbf{V}}\tilde{\Sigma}^{-1}\tilde{\mathbf{U}}^*\mathbf{b}$$

when all non-singular value are captured, since then $\mathbf{A}^\dagger\mathbf{A} = \mathbf{I}$ is the unitary matrix.

If the SVD is available, this is an $\mathcal{O}(n^2)$ operation to compute the solution of the system.

Ordinary Least Square Regression

Theorem (Gauss-Markov)

If $\mathbf{Ax} = \mathbf{b}$ *is an ordinary linear system, then* $\mathbf{x} = (\mathbf{A^T A})^{-1}\mathbf{A^T b}$ *is a global minimizer of* $E^2 = (\mathbf{Ax} - \mathbf{b})^{\mathbf{T}}(\mathbf{Ax} - \mathbf{b})$.

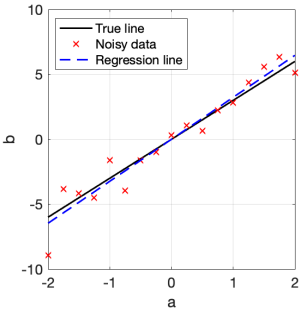
Proof: Let $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$, with $m \geq n$ and rank$(\mathbf{A}) = n$. These assumptions imply that $\mathbf{A}^*\mathbf{A}$ is invertible, cf. former slide on the Moore-Penrose left Pseudo-inverse. The squared residual error between the left and right hand side of the system $\mathbf{Ax} = \mathbf{b}$, can be written as,

$$\begin{aligned} r^2(\mathbf{x}) &= (\mathbf{Ax} - \mathbf{b})^*(\mathbf{Ax} - \mathbf{b}) \\ &= \mathbf{x}^*\mathbf{A^T A x} - \mathbf{x}^*\mathbf{A}^*\mathbf{b} - \mathbf{b}^*\mathbf{Ax} + \mathbf{b}^*\mathbf{b} = \mathbf{x}^*\mathbf{A}^*\mathbf{Ax} - \mathbf{2b}^*\mathbf{Ax} + \mathbf{b}^*\mathbf{b} \end{aligned}$$

The optimum is found by taking $\frac{\partial r^2}{\partial \mathbf{x}} = 0$, i.e.

$$\frac{\partial r^2}{\partial \mathbf{x}} = \mathbf{2A}^*\mathbf{Ax} - \mathbf{2A}^*\mathbf{b} = 0 \quad \leftrightarrow \quad \mathbf{A}^*\mathbf{Ax} = \mathbf{A}^*\mathbf{b} \quad \leftrightarrow \quad \mathbf{x} = (\mathbf{A}^*\mathbf{A})^{-1}\mathbf{A}^*\mathbf{b}$$

One-dimensional Linear regression



Example (Book, Figure 1.9, page 20): Determine the slope (x) of a linear regression line based on synthetic data we assume is related linearly as

$$\begin{bmatrix} | \\ \mathbf{b} \\ | \end{bmatrix} = \begin{bmatrix} | \\ \mathbf{a} \\ | \end{bmatrix} x = \tilde{\mathbf{U}}\tilde{\Sigma}\tilde{\mathbf{V}}^* x$$

Solve the problem through taking the SVD of the data $\mathbf{a}$ (that include noise) and compute the pseudoinverse to find an approximate solution

$$x = \tilde{\mathbf{V}}\tilde{\Sigma}^{-1}\tilde{\mathbf{U}}^*\mathbf{b}$$

Multi-Linear regression

Let's approach linear regression as a **learning problem**. We assume a linear model in the form

$$y = \sum_j w_j x_j + b$$

Remark, in the case of one-dimensional regression, this model is simply $y = wx + b$.

To make a fit to data pairs (x_i, y_i), we seek to quantify how good the fit is by introducing a loss function $L(y, t)$ that measures how far off a prediction y is from the target t, e.g., the square error

$$L(y, t) = \frac{1}{2}(y - t)^2 = \frac{1}{2}r^2,$$

that is measured in terms of the residual $r \equiv y - t$, which should be close to zero. This model together with the loss function constitute an optimization problem, where we are trying to minimize a cost function $C(w_1, ..., w_d, b)$ wrt. the model parameters (weights and bias, $\{\mathbf{w}$,b$\}$).

Multi-Linear regression

Define the cost function as the loss, averaged over all the training examples

$$C(w_1, ..., w_d, b)) = \frac{1}{2}\sum_{i=1}^{N} L(y^{(i)}, t^{(i)})$$

$$= \frac{1}{2N}\sum_{i=1}^{N}\left(y^{(i)} - t^{(i)}\right)^2 = \frac{1}{2N}\sum_{i=1}^{N}\left(\sum_j w_j x_j^{(i)} + b - t^{(i)}\right)^2$$

Remark, the loss is a function of the prediction and targets, while the cost is a function of the model parameters. The objective is to minimize the cost.

Introduce the bias b as a weight in the sum and change index ($j \mapsto j'$). The optimum is found by taking

$$\frac{\partial C}{\partial w_j} = \frac{1}{N}\sum_{i=1}^{N} x_j^{(i)}\left(\sum_{j'=1}^{D} w_{j'} x_{j'}^{(i)} - t^{(i)}\right)$$

$$= \frac{1}{N}\sum_{j'=1}^{D}\left(\sum_{i=1}^{N} x_j^{(i)}\right) w_{j'} - \frac{1}{N}\sum_{i=1}^{N} x_j^{(i)} t^{(i)} = 0$$

Multi-Linear regression

Introduce the bias b as a weight in the sum and change index ($j \mapsto j'$). The optimum is found by taking

$$\frac{\partial C}{\partial w_j} = \frac{1}{N}\sum_{i=1}^{N} x_j^{(i)} \left(\sum_{j'=1}^{D} w_{j'} x_{j'}^{(i)} - t^{(i)} \right) = \frac{1}{N}\sum_{j'=1}^{D} \left(\sum_{i=1}^{N} x_j^{(i)} \right) w_{j'} - \frac{1}{N}\sum_{i=1}^{N} x_j^{(i)} t^{(i)} = 0$$

This results in D equations in D variables

$$\sum_{j'=1}^{D} A_{jj'} w_{j'} - c_j = 0, \quad \forall j \in \{1, ..., D\},$$

where (note that $\mathbf{A}$ is defined as the correlation matrix)

$$A_{jj'} = \frac{1}{N}\sum_{j'=1}^{D} \left(\sum_{i=1}^{N} x_j^{(i)} \right), \quad c_j = \frac{1}{N}\sum_{i=1}^{N} x_j^{(i)} t^{(i)},$$

or

$$\mathbf{A} = \frac{1}{N}\mathbf{X}^*\mathbf{X}, \quad \mathbf{c} = \frac{1}{N}\mathbf{X}^*\mathbf{t}.$$

Solve this linear system to obtain the weights (and bias) that minimizes the cost

$$\mathbf{A}\mathbf{w} = \mathbf{c}, \quad \leftrightarrow \quad \mathbf{w} = \mathbf{A}^{-1}\mathbf{c} = (\mathbf{X}^*\mathbf{X})^{-1}\mathbf{X}^*\mathbf{t}.$$

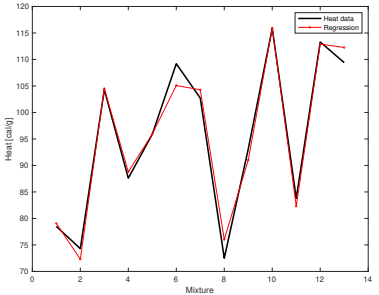
Thus, the problem can be solved using the pseudoinverse derived from the SVD.

Multi-Linear regression

Example: Cement heat data.

```
clear all, close all, clc

load hald;  % Load Portlant Cement dataset
A = ingredients;
b = heat;

[U,S,V] = svd(A,'econ');
x = V*inv(S)*U'*b;                    % Solve Ax=b using the SVD

plot(b,'k','LineWidth',2);  hold on            % Plot data
plot(A*x,'r-o','LineWidth',1.,'MarkerSize',2); % Plot regression
l1 = legend('Heat data','Regression')

%% Alternative 1  (regress)
x = regress(b,A);

%% Alternative 2  (pinv)
x = pinv(A)*b;
```

Multi-Linear regression



Example (Book, Figure 1.10, page 21): Determine the regression coefficients (x, weighting of ingredients) of a multi-linear regression based on cement mixtures containing four basic ingredients (data) we assume is related linearly as

$$\begin{bmatrix} | \\ \mathbf{b} \\ | \end{bmatrix} = \begin{bmatrix} | \\ \mathbf{a} \\ | \end{bmatrix} \mathbf{x} = \tilde{\mathbf{U}}\tilde{\Sigma}\tilde{\mathbf{V}}^*\mathbf{x}$$

Solve the problem through taking the SVD of the data $\mathbf{a}$ (that include noise) and compute the pseudoinverse to find an approximate solution

$$\mathbf{x} = \tilde{\mathbf{V}}\tilde{\Sigma}^{-1}\tilde{\mathbf{U}}^*\mathbf{b}$$

Randomized Singular Value Decomposition

Randomized linear algebra is a concept that is subject to active research that use random generators to enable new efficient types of algorithmic approaches. Following, Halko, Martinsson and Tropp (2011) we first describe the steps in a randomized singular value decomposition (rSVD) algorithm, and then we test it.

Construct a random projection matrix $\mathbf{P} \in \mathbb{R}^{m \times r}$ to sample the column space of $\mathbf{X} \in \mathbb{R}^{n \times m}$

$$\mathbf{Z} = \mathbf{X}\mathbf{P}$$

Then, it is possible to compute the low-rank $\mathbf{QR}$ decomposition of $\mathbf{Z}$ to obtain an orthonormal basis for $\mathbf{X}$

$$\mathbf{Z} = \mathbf{Q}\mathbf{R}$$

Using this low-rank basis $\mathbf{Q}$ we can project the data matrix $\mathbf{X}$ into a small space

$$\mathbf{Y} = \mathbf{Q}^*\mathbf{X}$$

from which it follows that $\mathbf{X} \approx \mathbf{Q}\mathbf{Y}$.

Randomized Singular Value Decomposition

Next, compute the SVD of $\mathbf{Y}$ that is of much smaller size than $\mathbf{X}$

$$\mathbf{Y} = \mathbf{U}_\mathbf{Y}\Sigma\mathbf{V}^*$$

As the final step, it is possible to reconstruct the high-dimensional left singular vectors $\mathbf{U}$ using $\mathbf{U}_\mathbf{Y}$ and $\mathbf{Q}$

$$\mathbf{U} = \mathbf{Q}\mathbf{U}_\mathbf{Y}$$

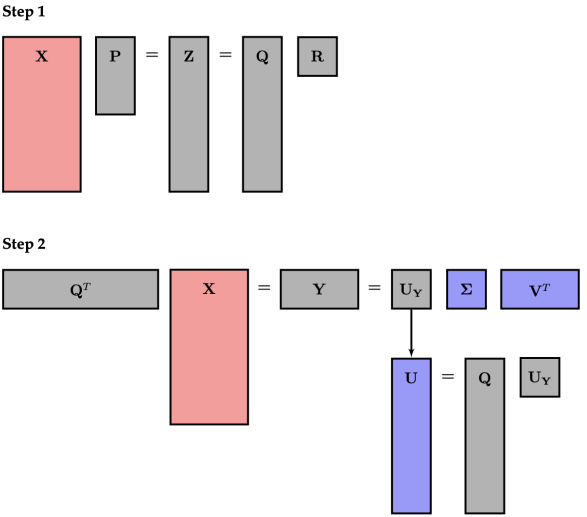
Most matrices $\mathbf{X}$ do not have an exact low-rank structure given by r modes. Instead, there are non-zero singular values σ_k for $k > r$ and the sketch $\mathbf{Z}$ will not span exactly the column space of $\mathbf{X}$. Often, increasing the number of columns in $\mathbf{P}$ from r to $r + p$ significantly improved results. This is known as *oversampling*.

A second challenge in using randomized algoritms is when the singular value spectrum decays slowly, so that remaining truncated singular values contain significant variance in the data $\mathbf{X}$. In this case, it is possible to pre-process $\mathbf{X}$ through q power iterations to create a new data matrix $\mathbf{X}^{(q)}$ with a more rapid singular value decay:

$$\mathbf{X}^{(q)} = (\mathbf{X}\mathbf{X}^*)^q\mathbf{X}$$

However, power iterations are expensive and requires more passes through the data $\mathbf{X}$.

Randomized Singular Value Decomposition

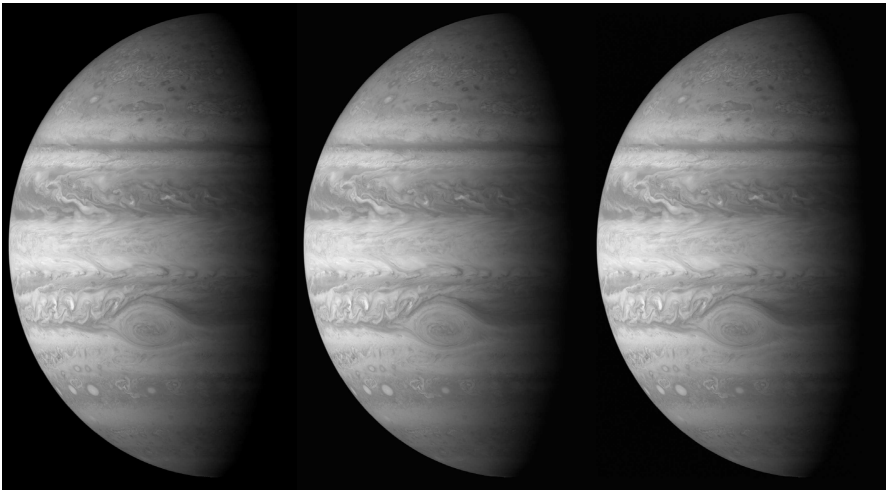


Randomized Singular Value Decomposition

MATLAB:

```
function [U,S,V] = rsvd(X,r,q,p);

% Step 1: Sample column space of X with P matrix
ny = size(X,2);
P = randn(ny,r+p); % random projection matrix
Z = X*P;
for k=1:q % power iterations
    Z = X*(X'*Z);
end
[Q,R] = qr(Z,0);

% Step 2: Compute SVD on projected Y=Q'*X;
Y = Q'*X;
[UY,S,V] = svd(Y,'econ');
U = Q*UY;
```

Randomized Singular Value Decomposition



Original high-resolution (left), rank-400 approximation from SVD (middle) and from rSVD (right).

Randomized Singular Value Decomposition

The use of SVD on the high-resolution picture is a demonstration. What did we achieve?

Method	**Storage** [bytes]	**CPU time** [s]
Image, $\mathbf{X}$	57982560	-
SVD, $\mathbf{U}$, Σ, $\mathbf{V}$	139704160	8.3
truncated SVD, $\tilde{\mathbf{U}}$, $\tilde{\Sigma}$, $\tilde{\mathbf{V}}$	17497600	-
rSVD, $\mathbf{rU}$, $r\Sigma$, $\mathbf{rV}$	19025280	1.3
truncated rSVD, $\mathbf{r}\tilde{\mathbf{U}}$, $r\tilde{\Sigma}$, $\mathbf{r}\tilde{\mathbf{V}}$	17497600	-

So, using the truncated rSVD($q = 1$,$p = 5$,$r = 400$) we can achieve an estimated $\sim 85\%$ speedup compared to the economical SVD through reduction in computing time and the SVD/rSVD makes it possible to achieve $\sim 30\%$ reduction in storage requirement without comprising the visual quality of the high-resolution picture.[1]

[1] Hardware: MacBook Pro (16-inch, 2021).

Physics-informed Least-Squares Regression

Example: Consider a simple SIR type model for epidemiological modelling

$$\begin{aligned}\frac{\partial S}{\partial t} &= -\beta \frac{SI}{N},\\ \frac{\partial I}{\partial t} &= \beta \frac{SI}{N} - \gamma I,\\ \frac{\partial R}{\partial t} &= \gamma I.\end{aligned}$$

This system contains temporal derivatives and it is nonlinear in terms of the state variables for susceptible (S), infected (I) and recovered (R), and **linear** in terms of the parameters β and γ that determines the transmission rates. The system can be expressed as

$$\begin{bmatrix} \begin{bmatrix} \frac{\partial S}{\partial t} \\ \frac{\partial I}{\partial t} \\ \frac{\partial R}{\partial t} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} -\frac{SI}{N} & 0 \\ \frac{SI}{N} & -I \\ 0 & I \end{bmatrix} \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \end{bmatrix} \quad \leftrightarrow \quad \mathbf{b} = \mathbf{A}\mathbf{x}$$

Physics-informed Least-Squares Regression

Example: SIR type model for epidemiological modelling.

To solve this system to estimate the parameters, we need to compute

$$\mathbf{A}^*\mathbf{A}\mathbf{x} = \mathbf{A}^*\mathbf{b} \quad \leftrightarrow \quad \mathbf{x} = \begin{bmatrix} \beta \\ \gamma \end{bmatrix} = (\mathbf{A}^*\mathbf{A})^{-1}\mathbf{A}^*\mathbf{b}.$$

How to determine $\mathbf{b}$ that contains temporal derivatives given only knowledge of the state variables at difference times?

Numerical Differentiation using Finite Difference Approximations

The following operators:

$$D_+u(\bar{x}) := \frac{u(\bar{x}+h)-u(\bar{x})}{h} \qquad \text{(Forward)}$$

$$D_-u(\bar{x}) := \frac{u(\bar{x})-u(\bar{x}-h)}{h} \qquad \text{(Backward)}$$

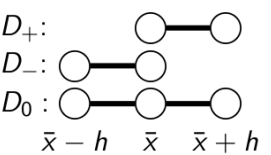
$$D_0u(\bar{x}) := \frac{u(\bar{x}+h)-u(\bar{x}-h)}{2h} \qquad \text{(Central)}$$

approximate $u'(\bar{x})$ from $u(x)$. In fact:

$$\lim_{h\to 0} D_\pm u(\bar{x}) = \lim_{h\to 0} D_0u(\bar{x}) = u'(\bar{x})$$

Finite difference approximations, with $r = \alpha + \beta + 1$ stencil, are of the form:

$$\frac{d^m u(\bar{x})}{dx^m} \approx \sum_{n=-\alpha}^{\beta} a_n u(\bar{x}+hn)$$



Physics-informed Least-Squares Regression

Example: SIR type model for epidemiological modelling. Simple forward finite difference approximations of the temporal terms can be done as

$$\frac{\partial S}{\partial t} \approx \frac{S^{n+1} - S^n}{t_{n+1} - t_n}$$

where $S(t_n) \equiv S^n$.

By measuring all states S^n, I^n, R^n, at times t_n for $n = 1, ..., N$ we can either compute the temporal derivatives assuming the right hand side (RHS) model is valid or approximate all derivatives

$$\begin{bmatrix} | \\ \begin{bmatrix} \frac{\partial S}{\partial t} \\ \frac{\partial I}{\partial t} \\ \frac{\partial R}{\partial t} \end{bmatrix} \\ | \end{bmatrix} = \begin{bmatrix} | \\ \begin{bmatrix} -\frac{SI}{N} & 0 \\ \frac{SI}{N} & -I \\ 0 & I \end{bmatrix} \\ | \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \end{bmatrix} \quad \leftrightarrow \quad \mathbf{b} = \mathbf{A}\mathbf{x}$$

... so we then need to be able evaluate (or measure) also the states with in the system.

Initial Value Problems

A general Initial Value Problem (IVP) takes the form

$$u'(t) = f(u(t), t), \quad t > t_0 \tag{1}$$

with some Initial Condition (IC) specified

$$u(t_0) = \eta$$

In the IVP $u(t)$ is an unknown function and η is given initial data.

In general, (1) may represent a scalar equation as well as a system of equations. In the latter case $u(t)$ will be a vector of the form

$$\mathbf{u}(t) = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_s \end{pmatrix}$$

in which case both $f(\boldsymbol{u}, t)$ and η are also vectors.

Solving Initial Value Problems Numerically

MATLAB:

```
% RHS function
function dydt = sir_model(t,y,beta,gamma)
    % Define the system of differential equations
    dydt = zeros(3,1);
    dydt(1) = -beta*y(1)*y(2);
    dydt(2) = beta*y(1)*y(2) - gamma*y(2);
    dydt(3) = gamma*y(2);
end

% Define the initial conditions and parameter values
% Define the initial conditions and parameter values
y0 = [1-0.001, 0.001, 0]; % [S, I, R]
beta = 0.8;
gamma = 1./5.4;
tspan = [0, 30]; % Define the time span for the simulation

% Solve the system of differential equations using ode45
[t,y] = ode45(@(t,y) sir_model(t,y,beta,gamma), tspan, y0);

% Plot the results
plot(t,y(:,1),'b',t,y(:,2),'r',t,y(:,3),'g')
legend('S','I','R')
```

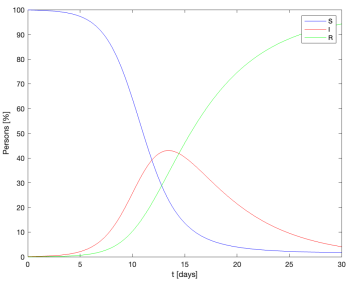
Solving Initial Value Problems Numerically

PYTHON:

```
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt

# RHS function
def sir_model(t, y, beta, gamma):
    # Define the system of differential equations
    S, I, R = y
    dSdt = -beta * S * I
    dIdt = beta * S * I - gamma * I
    dRdt = gamma * I
    return [dSdt, dIdt, dRdt]

# Define the initial conditions and parameter values
y0 = [1-0.001, 0.001, 0] # [S, I, R]
beta = 0.8
gamma = 1./5.4
tspan = (0, 30) # Define the time span for the simulation

# Solve the system of differential equations using solve_ivp
tmeasurements = np.linspace(0, 30, 100)
sol = solve_ivp(lambda t, y: sir_model(t, y, beta, gamma), tspan, y0, t_eval=tmeasurements)

# Plot the results
plt.plot(sol.t, sol.y[0], 'b', sol.t, sol.y[1], 'r', sol.t, sol.y[2], 'g')
plt.legend(['S', 'I', 'R'])
plt.show()
```

Physics-informed Least-Squares Regression

Example: SIR type model for epidemiological modelling. Results of parameter estimation using least square regression to determine the spread of disease parameters. This allow for forecasting using numerical methods when the model is known.



	Exact		Computed		Error	
Method	β	γ	$\hat{\beta}$	$\hat{\gamma}$	$\lvert\beta - \hat{\beta}\rvert$	$\lvert\gamma - \hat{\gamma}\rvert$
RHS, analytic	0.8000	0.1852	0.8000	0.1852	2.2204e-16	5.5511e-17
RHS, FDM	0.8000	0.1852	0.7974	0.1863	0.0026	0.0011

By using the numerical simulation results we have synthetic data available and hence we can assume this to be the true solution so we can compare against this and test for the influence of truncation errors by choosing the time step size Δt.

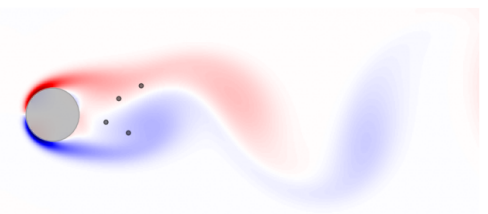
Physics-informed Least-Squares Regression

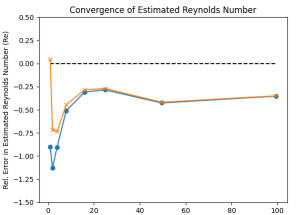
Example: Estimating the Reynolds Number from Vortex Shedding Data. Results of parameter estimation using least square regression to determine the Reynolds number that may be useful to characterise the flow regime of fluid flow about a cylinder. By combining numerical techniques with time series data it is feasible to perform such analysis.

The temporal evolution of vorticity is governed by the PDE (cf. Nielsen et al. 2025)

$$\omega_t + \mathbf{u} \cdot \nabla \omega = \frac{1}{Re} \nabla^2 \omega.$$

Hence, given time series data at (N_s) sensors it is possible to estimate the Reynolds number via physics-informed regression.

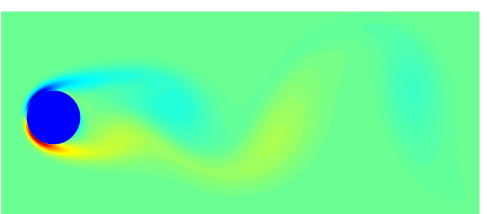




Exercises

This week, you should solve

- Exercise 1.7 : Fluid flow past a cylinder, Book, Chapter 1, page 52.
- Exercise X01.1: Setup a code that performs physics-informed least-squares regression to determine the model parameters of a spread of disease model. Reproduce first the results in the former slide example and then apply to a new spread of virus model. The data to be used for testing can be artificially constructed using a numerical solver and with known/own choice of parameters. For inspiration to select a model, cf. Shaier, Raissi & Seshaiyer (2022).
- Exercise X01.2: Setup a code that performs physics-informed least-squares regression to determine a model parameter of a PDE such as the vorticity equation. Reproduce the results of Nielsen et al. 2025 where the Reynolds number is estimated for Fluid flow past a cylinder.

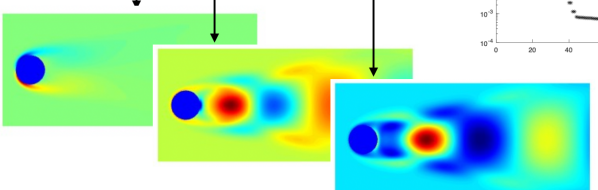


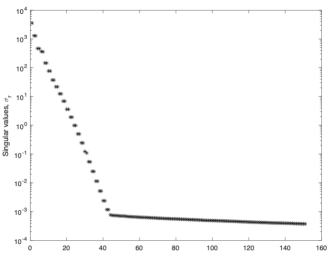
Exercises

Exercise 1.7 (a) Analysis of eigenflow fields via SVD decomposition

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^* \approx \tilde{\mathbf{U}}\tilde{\Sigma}\tilde{\mathbf{V}}^*$$

$$\tilde{\mathbf{U}} = \begin{bmatrix} | & | & & | \\ \mathbf{U}_1 & \mathbf{U}_2 & \cdots & \mathbf{U}_r \\ | & | & & | \end{bmatrix}$$





References

- J.S. Nielsen, M.G. Jacobsen, A.B. Olson, M.P. Sørensen and A.P. Engsig-Karup. Physics-Informed Regression: Parameter Estimation in Parameter-Linear Nonlinear Dynamic Models, arxiv preprint, July, 2025.
- S. Shaier, M. Raissi and P. Seshaiyer. Data-Driven Approaches for Predicting Spread of Infectious Diseases Through DINNs: Disease Informed Neural Networks. Letters in biomathematics, 2022.
- N. Halko, P. G. Martinsson and J. A. Tropp. Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions. SIAM REVIEW, Vol. 53, No. 2, pp. 217–288, 2011.