ME 536

Week 5: SVD, ...

Matrix Notation Conventions

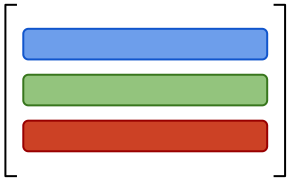
Given a matrix $\mathbf{M}$:

$\tilde{\mathbf{M}}$ is an approximation of $\mathbf{M}$,
i.e. $\tilde{\mathbf{M}} \approx \mathbf{M}$ hence, $\tilde{\mathbf{M}} - \mathbf{M} \neq \mathbf{0}$

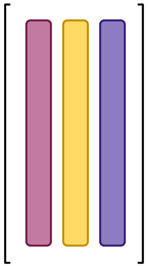
$\tilde{\mathbf{M}}_k$ is a *rank-k* approximation of $\mathbf{M}$

Matrix Notation Conventions

Given a matrix $\mathbf{M}$:



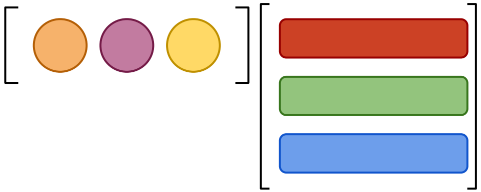
A **fat** / **wide** / **short** matrix refers to $\mathbf{M}$ that has has more columns than rows



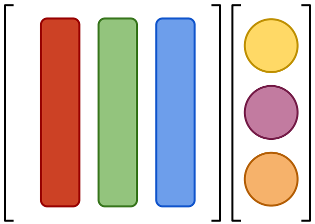
A **skinny** / **tall** matrix refers to $\mathbf{M}$ that has more rows than columns

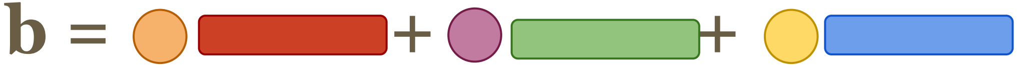
Matrix and a Vector multiplied: 2 Alternatives

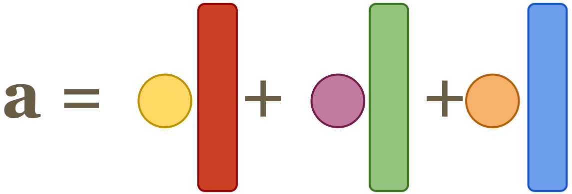
$\mathbf{rM} = \mathbf{b}$



$\mathbf{Mc} = \mathbf{a}$

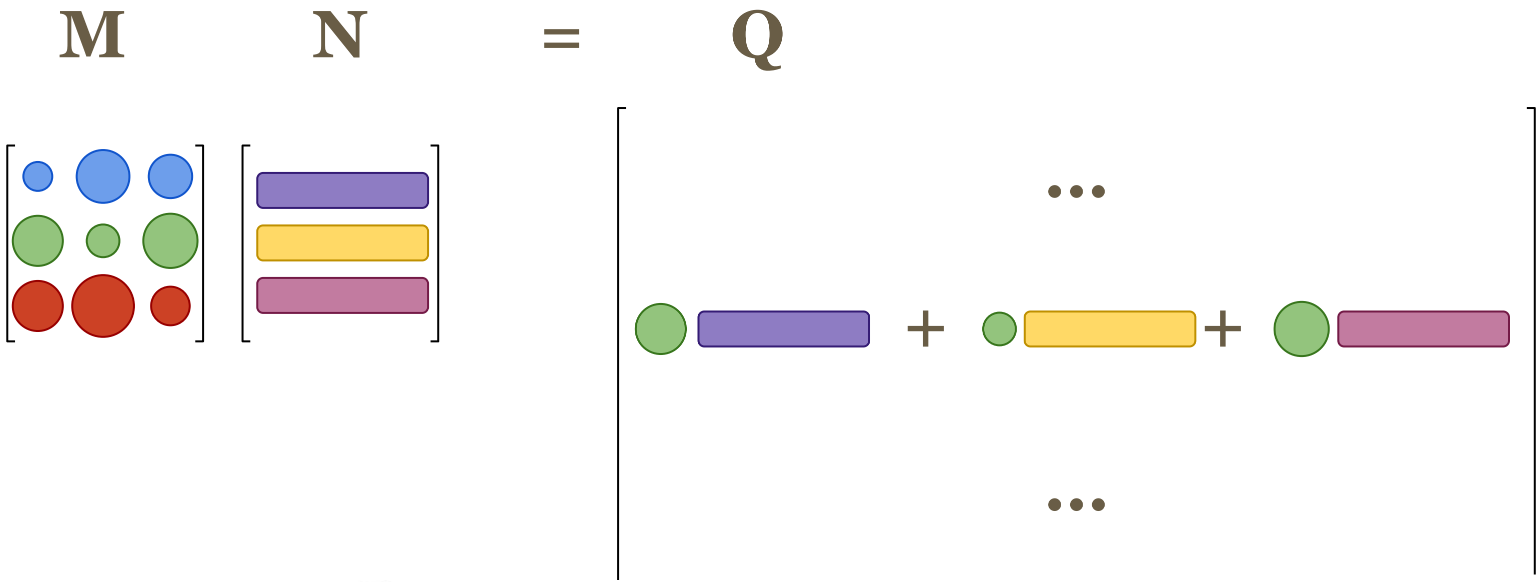






Matrix multiplied by a Matrix: $\mathbf{MN} = \mathbf{Q}$

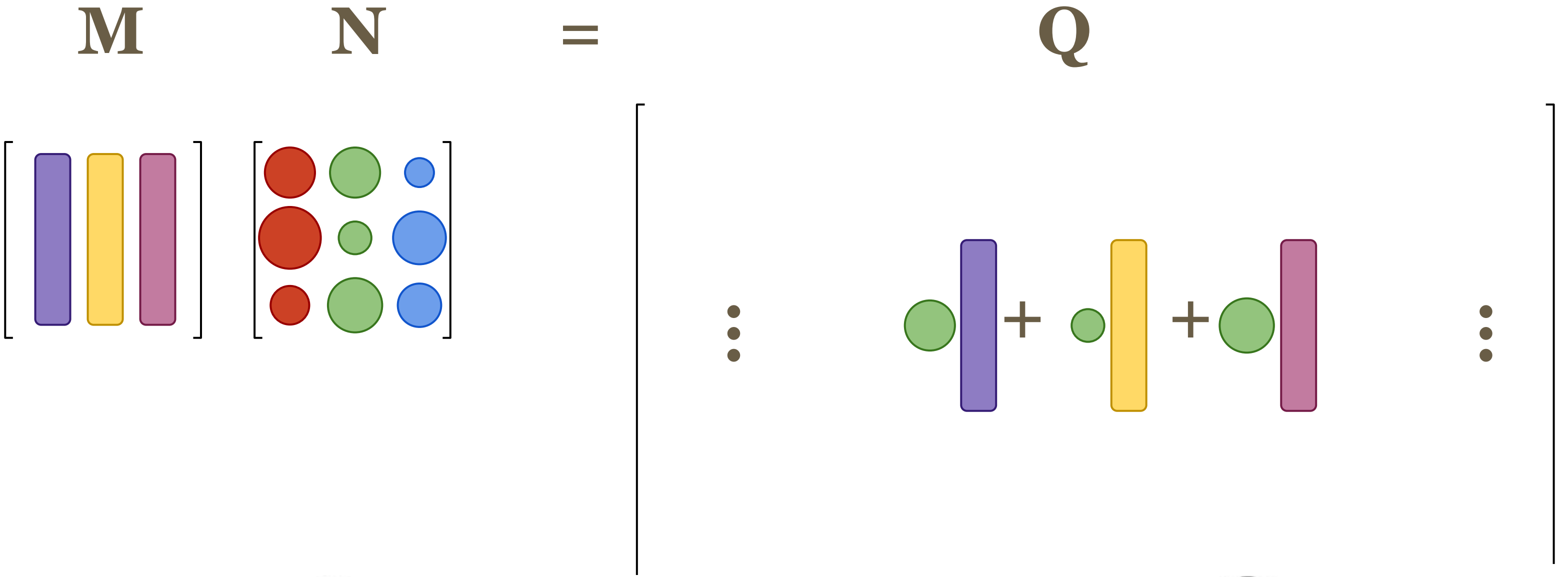
Linear combination of Rows



$$r_i^{\mathbf{Q}} = \sum_j m_{i,j}\, r_j^{\mathbf{N}} \quad \rightarrow \quad r_i^{\mathbf{Q}} \in \mathbf{R}(\mathbf{N}),\ \forall i$$

Matrix multiplied by a Matrix: $\mathbf{MN} = \mathbf{Q}$

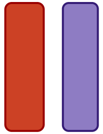
Linear combination of Columns

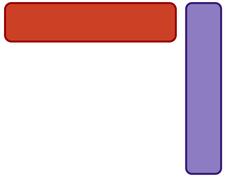


$$c_i^{\mathbf{Q}} = \sum_j n_{j,i}\, c_j^{\mathbf{M}} \quad \rightarrow \quad c_j^{\mathbf{Q}} \in \mathbf{C}(\mathbf{M}),\ \forall j$$

Recall: yet another alternative perspective

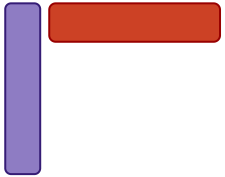
It was also possible to use ***inner-product*** perspective:

Given two column vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$ 



$$\mathbf{a}^T\mathbf{b} = s$$

How about the other way around: ***outer-product***?



$$\mathbf{a}\mathbf{b}^T = \mathbf{S}$$

Matrix multiplied by a Matrix: $\mathbf{MN} = \mathbf{Q}$

Sum of Outer Products

$\mathbf{M}$ $\mathbf{N}$ =

$$\mathbf{Q} = \sum_i \mathbf{c}_i^{\mathbf{M}} \mathbf{r}_i^{\mathbf{N}}$$

$\mathbf{c}_i^{\mathbf{M}} \mathbf{r}_i^{\mathbf{N}}$ is the outer product between the i^{th} column of $\mathbf{M}$ and the i^{th} row of $\mathbf{N}$.

Also note that: $rank(\mathbf{c}_i^{\mathbf{M}} \mathbf{r}_i^{\mathbf{N}}) = 1, \ \forall i$

EXERCISE:

Find the result as a sum of outer products

$$\begin{bmatrix} 3 & 5 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}\begin{bmatrix} 2 & 1 \end{bmatrix} + \begin{bmatrix} 5 \\ 1 \end{bmatrix}\begin{bmatrix} 1 & 3 \end{bmatrix}$$

$$= \begin{bmatrix} 6 & 3 \\ 2 & 1 \end{bmatrix} + \begin{bmatrix} 5 & 15 \\ 1 & 3 \end{bmatrix} = \begin{bmatrix} 11 & 18 \\ 3 & 4 \end{bmatrix}$$

Note that each outer product is a **Rank-1** matrix

whereas their sum is a **Rank-2** matrix

What if ***first outer-product matrix was more important than the second*** ?

Matrix multiplied by a Matrix: $\mathbf{MN} = \mathbf{Q}$

By sub-blocks

$$\mathbf{M} \qquad \mathbf{N} \qquad = \qquad \mathbf{Q}$$

$$\left[\begin{array}{c|c} \mathbf{M}_1 & \mathbf{M}_2 \\ \hline \mathbf{M}_3 & \mathbf{M}_4 \end{array}\right] \left[\begin{array}{c|c} \mathbf{N}_1 & \mathbf{N}_2 \\ \hline \mathbf{N}_3 & \mathbf{N}_4 \end{array}\right] = \left[\begin{array}{c|c} \mathbf{M}_1\mathbf{N}_1 + \mathbf{M}_2\mathbf{N}_3 & \mathbf{M}_1\mathbf{N}_2 + \mathbf{M}_2\mathbf{N}_4 \\ \hline \mathbf{M}_3\mathbf{N}_1 + \mathbf{M}_4\mathbf{N}_3 & \mathbf{M}_3\mathbf{N}_2 + \mathbf{M}_4\mathbf{N}_4 \end{array}\right]$$

If sub-matrices (blocks) in M and N are **compatible**, Q can be found by treating blocks as matrix elements.

Useful when some blocks are all-zeros, all-ones or Identity

Matrix Decomposition

Method	Desired forms, features
Eigen-	Diagonal
LU	Triangular
QR	Orthonormal, triangular
CUR	Basis from data
SVD	Orthonormal, diagonal
…	

Recall: Eigen Decomposition

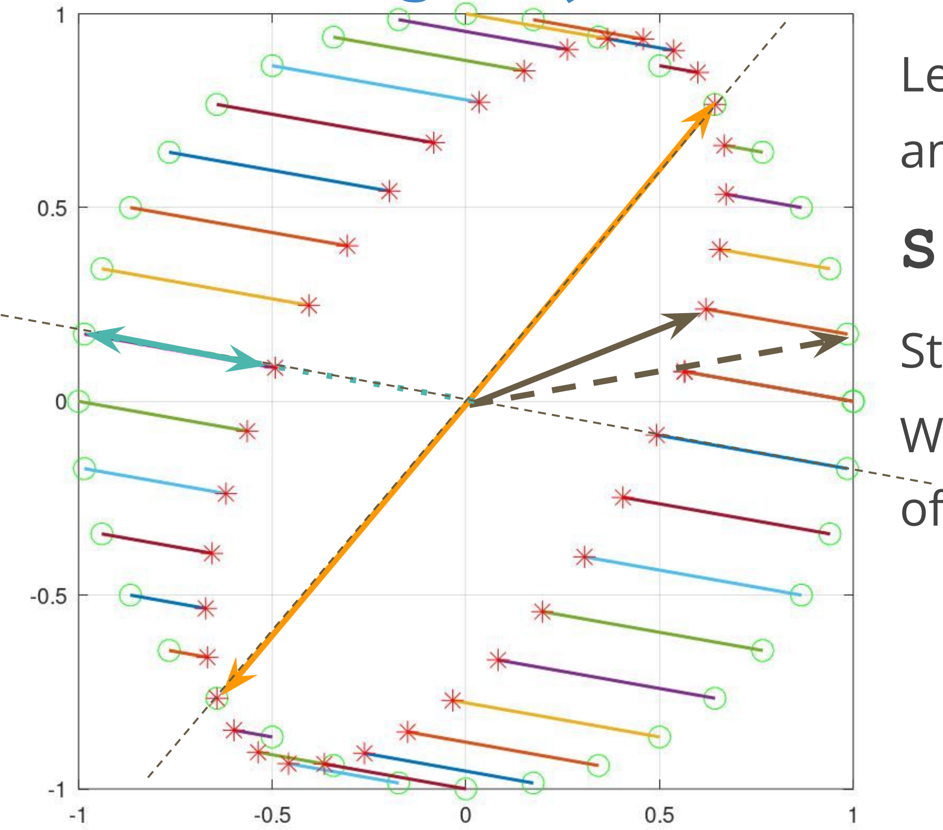
Refresh your Memory:

Eigen Decomposition starts with:

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v} \text{ where } \mathbf{A} \text{ is } n \times n$$

to yield $\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}$

Recall: Eigen-(Values, Vectors) $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$



Let columns of $\mathbf{C}$ be the green circles (○), and let there be a *2x2* matrix $\mathbf{A}$ where:

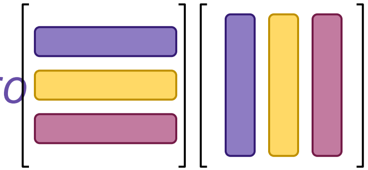
$\mathbf{S} = \mathbf{A} \; \mathbf{C}$

Stars (✶) are the columns of $\mathbf{S}$

What are the <u>eigenvalues</u> and <u>eigenvectors</u> of $\mathbf{A}$?

Orthogonal Matrices

A **matrix is $\mathbf{M}$** referred to as **orthonormal** *(orthogonal by some other)* if its *columns and ro* *are* <u>orthogonal unit vectors</u> (i.e. *orthonormal vectors*)



$$= \begin{bmatrix} 1 & \dots & 0 \\ 0 & \dots & 0 \\ 0 & \dots & 1 \end{bmatrix}$$

Therefore:

$$\mathbf{M}^T\mathbf{M} = \mathbf{M}\mathbf{M}^T = \mathbf{I}$$

$$\mathbf{M}^{-1} = \mathbf{M}^T$$

Note that: $||\mathbf{M}\mathbf{v}|| = ||\mathbf{v}||$ **but WHY ? and what does it imply ?**

Properties of Symmetric Matrices

Symmetric → Square implied

Eigen(.)s of a symmetric matrix:

- Eigenvalues are real
- Eigenvectors are **orthogonal**

Square and symmetric ??? too specific

We were getting ready to deal with some generic matrix: $\mathbf{M}_{\mathrm{dxn}}$

Properties of Orthogonal[1] Matrices

- Transpose is inverse
- They do **not scale** but **just rotate**
- Multiplication of 2 orthogonal matrices are orthogonal

1: Unitary if elements are not necessarily real

How do you get a symmetric matrix from: $\mathbf{M}_{dxn}$

$\mathbf{M}^T\mathbf{M} = \mathbf{V}_{nxn}$ → rank($\mathbf{V}$)

$\mathbf{M}\mathbf{M}^T = \mathbf{U}_{dxd}$ → rank($\mathbf{U}$)

How do you get a symmetric matrix from: $\mathbf{M}_{dxn}$

$\mathbf{M}^T\mathbf{M} = \mathbf{V}_{nxn}$ $\mathbf{M}\mathbf{M}^T = \mathbf{U}_{dxd}$

$\rightarrow$ eigen(values/vectors) of ($\mathbf{U}$)

$\rightarrow$ eigen(values/vectors) of ($\mathbf{V}$)

SVD: Singular Value Decomposition

Any matrix $\mathbf{M}_{dxn}$ can be decomposed as:

$$\mathbf{M} = \mathbf{U} \ \boldsymbol{\Sigma} \ \mathbf{V}^T$$

Left Singular Vectors

Singular Values

Diagonal

Right Singular Vectors

Orthogonal

??? dimensions ???

SVD: Singular Value Decomposition

$$\mathbf{M}_{\text{dxn}} = \mathbf{U}_{\text{dxd}}\ \mathbf{\Sigma}_{\text{dxn}}\ \mathbf{V^T}_{\text{nxn}}$$

Orthogonal

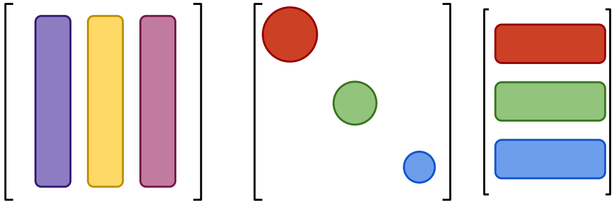
$$\mathbf{U}_{d\times d} \rightarrow \mathbf{U}\mathbf{U}^T = \mathbf{U}^T\mathbf{U} = \mathbf{I}$$

Orthogonal

$$\mathbf{V}_{n\times n} \rightarrow \mathbf{V}\mathbf{V}^T = \mathbf{V}^T\mathbf{V} = \mathbf{I}$$

SVD: Singular Value Decomposition

$$\mathbf{M}_{dxn} = \mathbf{U}_{dxd}\ \mathbf{\Sigma}_{dxn}\ \mathbf{V}^{\mathbf{T}}_{nxn} =$$



d = n

$$\Sigma_{d\times d} = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \cdots & \cdots & \ddots & \cdots \\ 0 & 0 & \cdots & \sigma_d \end{bmatrix}_{dxd}$$

$$\sigma_1 \geq \sigma_2 \geq \ldots \sigma_{d-1} \geq \sigma_d$$

SVD: Singular Value Decomposition - Tall $\mathbf{M}$

$$\mathbf{M}_{dxn} = \mathbf{U}_{dxd}\ \boldsymbol{\Sigma}_{dxn}\ \mathbf{V}^{\mathbf{T}}_{nxn}$$

d > n

$$\boldsymbol{\Sigma}_{d\times n} = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & \sigma_n \\ 0 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}_{dxn}$$

SVD: Singular Value Decomposition - Wide $\mathbf{M}$

$$\mathbf{M}_{\text{dxn}} = \mathbf{U}_{\text{dxd}}\ \mathbf{\Sigma}_{\text{dxn}}\ \mathbf{V}^{\mathbf{T}}_{\text{nxn}}$$

d < n

$$\mathbf{\Sigma}_{d\times n} = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & \sigma_d & 0 & \cdots & 0 \end{bmatrix}_{dxn}$$

Show that:

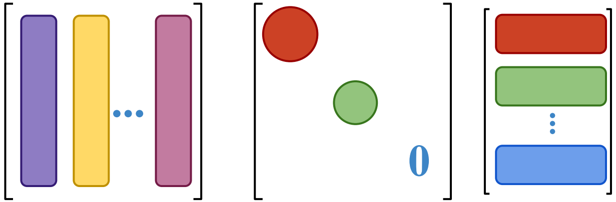
- The singular values of matrix $\mathbf{M}$ are the square root of the eigenvalues of $\mathbf{M}^T\mathbf{M}$ or $\mathbf{M}\mathbf{M}^T$

- Left Singular vectors are eigenvectors of $\mathbf{M}\mathbf{M}^T$

- Right Singular vectors are eigenvectors of $\mathbf{M}^T\mathbf{M}$

SVD: Low Rank Data matrix $\longrightarrow$ rank($\mathbf{M}$) = r < min(d,n)

Let rank($\mathbf{M}$) = r $\quad\rightarrow\quad$ $$\boldsymbol{\Sigma}_{d\times n} = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & \sigma_r & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \end{bmatrix}_{dxn}$$



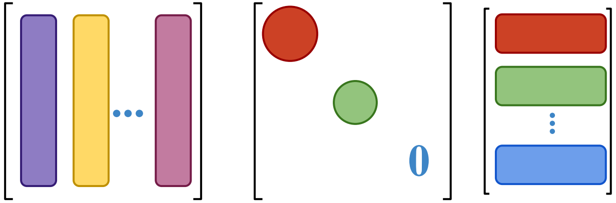
$$\mathbf{M}_{\text{dxn}} = \mathbf{U}_{\text{dx?}}\ \boldsymbol{\Sigma}_{\text{?x?}}\ \mathbf{V}^{\mathbf{T}}_{\text{?xn}}$$

SVD: Singular Value Decomposition - columns

columns are important, columns are meaningful, oooh those columns

how about the rows?

$$\mathbf{M}_{\text{dxn}} = \mathbf{U}_{\text{dxd}} \; \mathbf{\Sigma}_{\text{dxn}} \; \mathbf{V}^{\mathbf{T}}_{\text{nxn}}$$



$\mathbf{u}_1, \ldots, \mathbf{u}_r$ is an orthonormal basis for $\mathbf{C}(\mathbf{M})$

$\mathbf{u}_{r+1}, \ldots, \mathbf{u}_d$ is an orthonormal basis for $\mathbf{N}(\mathbf{M}^T)$, i.e. **left null space**

$\mathbf{v}_1, \ldots, \mathbf{v}_r$ is an orthonormal basis for $\mathbf{C}(\mathbf{M}^\mathbf{T})$

$\mathbf{v}_{r+1}, \ldots, \mathbf{v}_n$ is an orthonormal basis for $\mathbf{N}(\mathbf{M})$, i.e. **null space**

where $r = rank(\mathbf{M})$

SVD: Low Rank Data matrix $\longrightarrow$ rank($\mathbf{M}$) < n

rank($\mathbf{M}$) = r < n

$$\hat{\mathbf{M}}_{d\times n} = \mathbf{U}_{d\times r}\mathbf{\Sigma}_{r\times r}\mathbf{V}^T_{r\times n}$$

$\mathbf{M}$ can be reconstructed from $\mathbf{U}_{dxr}\mathbf{\Sigma}_{kxr}\mathbf{V}^{\mathbf{T}}_{rxn}$ without any loss

$$\hat{\mathbf{M}} = \mathbf{M}$$

Referred to with names such as: *skinny* SVD, *economy* SVD, truncated SVD

SVD: Matrix Approximation - who get to die first?

$$\mathbf{M}_{\text{dxn}} = \mathbf{U}_{\text{dxd}} \ \mathbf{\Sigma}_{\text{dxn}} \ \mathbf{V}^{\mathbf{T}}_{\text{nxn}}$$

For some $0 < \mathrm{k} < \mathrm{r} = \mathrm{rank}(\mathbf{M})$ approximation of $\mathbf{M}$:

$$\tilde{\mathbf{M}}_{d\times n} = \mathbf{U}_{d\times k}\mathbf{\Sigma}_{k\times k}\mathbf{V}^T_{k\times n}$$

$$\tilde{\mathbf{M}}_k \approx \mathbf{M}$$

$$\mathbf{M}_{error} = \tilde{\mathbf{M}}_k - \mathbf{M}$$

How good is the approximation: $\tilde{\mathbf{M}}_{d\times n} = \mathbf{U}_{d\times k}\mathbf{\Sigma}_{k\times k}\mathbf{V}^T_{k\times n}$

Best in some sense, i.e. Frobenius

Assume that $\mathrm{k} < \mathrm{rank}(\mathbf{M}) = \mathrm{r}$

$$\mathbf{\Sigma}_{d\times n} = \begin{bmatrix} \sigma_1 & \cdots & 0 & 0 & \cdots & 0 & \cdots \\ \cdots & \ddots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & \sigma_k & 0 & \cdots & 0 & \cdots \\ 0 & \cdots & 0 & \sigma_{k+1} & \cdots & 0 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \ddots & \cdots & \cdots \\ 0 & \cdots & 0 & 0 & \cdots & \sigma_r & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \ddots \end{bmatrix}_{dxn}$$

SVD: Sum of Outer Products Perspective

$$\mathbf{M}_{\text{dxn}} = \mathbf{U}_{\text{dxd}}\ \mathbf{\Sigma}_{\text{dxn}}\ \mathbf{V^T}_{\text{nxn}}$$

$$\mathbf{M} = \sum \sigma_i \mathbf{c}_i^{\mathbf{U}} \mathbf{r}_i^{\mathbf{V}^T}$$

$$\mathbf{M} = \sigma_1 \mathbf{c}_1^{\mathbf{U}} \mathbf{r}_1^{\mathbf{V}^T} + \sigma_2 \mathbf{c}_2^{\mathbf{U}} \mathbf{r}_2^{\mathbf{V}^T} + \ldots + \sigma_r \mathbf{c}_r^{\mathbf{U}} \mathbf{r}_r^{\mathbf{V}^T}$$

where $\mathbf{c}_i^{\mathbf{U}}$ is $d \times 1$ and $\mathbf{r}_i^{\mathbf{V}^T}$ is $1 \times n$

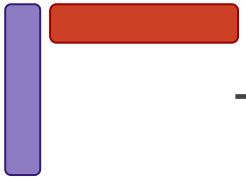
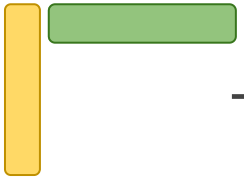
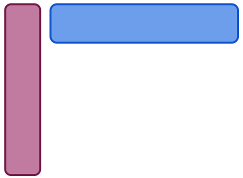
hence, $\mathbf{c}_i^{\mathbf{U}} \mathbf{r}_i^{\mathbf{V}^T}$ is $d \times n$ where the *importance* is determined by σ_i

SVD: Sum of Outer Products Perspective

$$\mathbf{M}_{\mathrm{dxn}} = \mathbf{U}_{\mathrm{dxd}}\ \mathbf{\Sigma}_{\mathrm{dxn}}\ \mathbf{V}^{\mathrm{T}}_{\mathrm{nxn}}$$

$$\mathbf{M} = \sum \sigma_i \mathbf{c}_i^{\mathbf{U}} \mathbf{r}_i^{\mathbf{V}^T}$$

$$\mathbf{M} = \sigma_1 \mathbf{c}_1^{\mathbf{U}} \mathbf{r}_1^{\mathbf{V}^T} + \sigma_2 \mathbf{c}_2^{\mathbf{U}} \mathbf{r}_2^{\mathbf{V}^T} + \ldots + \sigma_r \mathbf{c}_r^{\mathbf{U}} \mathbf{r}_r^{\mathbf{V}^T}$$

$\mathbf{M} = \sigma_1$  $+\ \sigma_2$  $+\ \ldots\ +\ \sigma_r$ 

How good is the approximation: $\tilde{\mathbf{M}}_{d\times n} = \mathbf{U}_{d\times k}\mathbf{\Sigma}_{k\times k}\mathbf{V}^T_{k\times n}$

$$\mathbf{\Sigma}_{d\times n} = \begin{bmatrix} \sigma_1 & \cdots & 0 & 0 & \cdots & 0 & \cdots \\ \cdots & \ddots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & \sigma_k & 0 & \cdots & 0 & \cdots \\ 0 & \cdots & 0 & \sigma_{k+1} & \cdots & 0 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \ddots & \cdots & \cdots \\ 0 & \cdots & 0 & 0 & \cdots & \sigma_r & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \ddots \end{bmatrix}_{d\times n}$$

Best *rank(k)* approximation in *Nuclear*, *Spectral* and *Frobenious* norm sense[1,2]

Note that: $k < \mathrm{rank}(\mathbf{M}) = r$

$$\mathbf{M} = \sigma_1 \mathbf{c}_1^{\mathbf{U}} \mathbf{r}_1^{\mathbf{V}^T} + \sigma_2 \mathbf{c}_2^{\mathbf{U}} \mathbf{r}_2^{\mathbf{V}^T} + \ldots + \sigma_r \mathbf{c}_r^{\mathbf{U}} \mathbf{r}_r^{\mathbf{V}^T}$$

$$\tilde{\mathbf{M}} = \sigma_1 \mathbf{c}_1^{\mathbf{U}} \mathbf{r}_1^{\mathbf{V}^T} + \sigma_2 \mathbf{c}_2^{\mathbf{U}} \mathbf{r}_2^{\mathbf{V}^T} + \ldots + \sigma_k \mathbf{c}_k^{\mathbf{U}} \mathbf{r}_k^{\mathbf{V}^T}$$

$$\mathbf{M} = \tilde{\mathbf{M}} + \sigma_{k+1} \mathbf{c}_{k+1}^{\mathbf{U}} \mathbf{r}_{k+1}^{\mathbf{V}^T} + \ldots + \sigma_r \mathbf{c}_r^{\mathbf{U}} \mathbf{r}_r^{\mathbf{V}^T}$$

$$\mathbf{M} - \tilde{\mathbf{M}} = \sigma_{k+1} \mathbf{c}_{k+1}^{\mathbf{U}} \mathbf{r}_{k+1}^{\mathbf{V}^T} + \ldots + \sigma_r \mathbf{c}_r^{\mathbf{U}} \mathbf{r}_r^{\mathbf{V}^T}$$

1- Check out [Eckart-Young-Mirsky theorem]

2- What sense? Do matrices have norms too, now I have seen everything

Python:

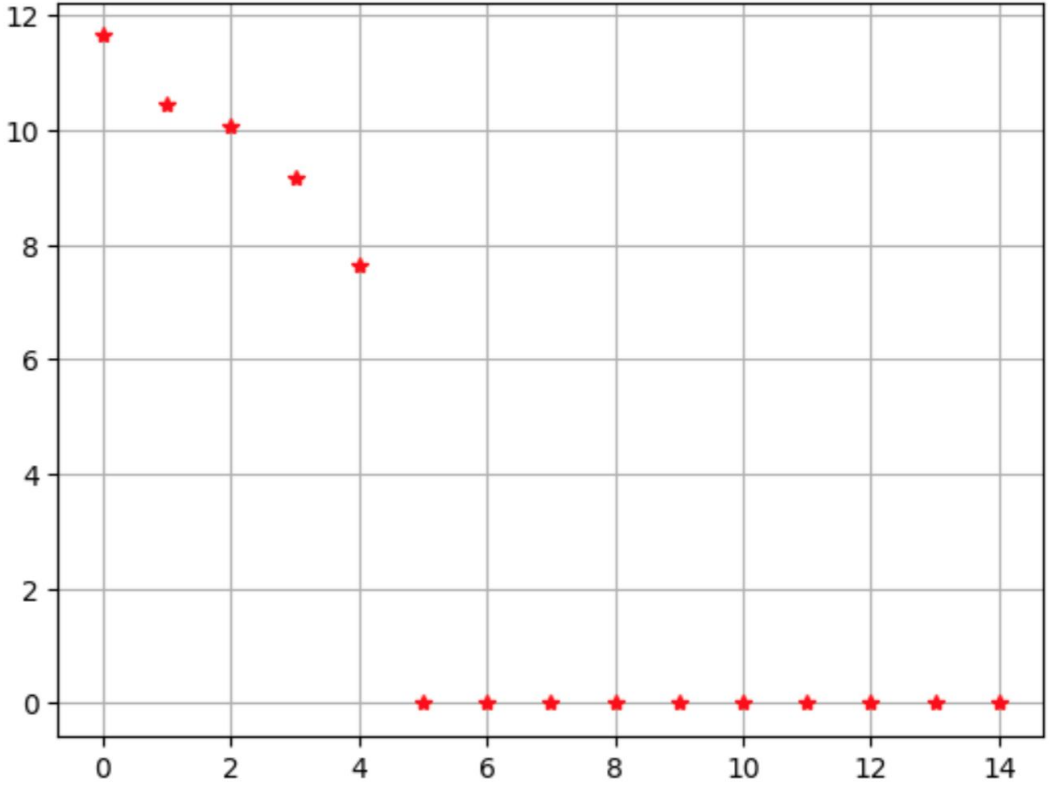
Recall my python function:

```
>> DataInSubspace()
```

```python
M = DataInSubspace(15, 100, 5)
print(f'rank(M)={np.linalg.matrix_rank(M)}')
U,S,VT = np.linalg.svd(M, full_matrices=False)

M5 = U @ (np.diag(S) @ VT)
print(f'rank(Ma)={np.linalg.matrix_rank(M5)}')
print(f'Ma - M = {abs(M5-M).sum()}')
plt.plot(S, 'r*')
plt.grid()
plt.title('Singular values')
```

```
rank(M)=5
rank(M5)=5
abs(M5 - M).sum = 6.899799741971746e-13
```

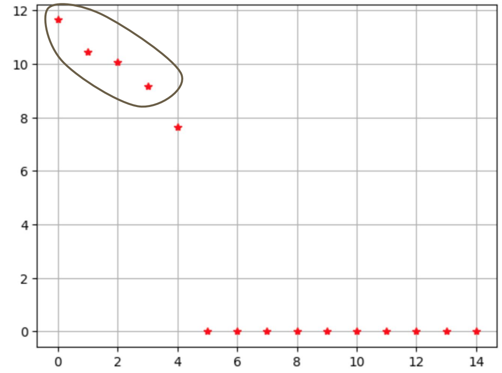


Python:

```
M = DataInSubspace(15, 100, 5)
print(f'rank(M)={np.linalg.matrix_rank(M)}')
U,S,VT = np.linalg.svd(M, full_matrices=False)

M5 = U @ (np.diag(S) @ VT)
print(f'rank(Ma)={np.linalg.matrix_rank(M5)}')
print(f'Ma - M = {abs(M5-M).sum()}')
plt.plot(S, 'r*')
plt.grid()
plt.title('Singular values')
```

```
rank(M)=5
rank(M5)=5
abs(M5 - M).sum = 6.899799741971746e-13
```



Rank-4 approximation of M

$$\mathbf{M} = \boxed{\sigma_1 \mathbf{c}_1^{\mathbf{U}} \mathbf{r}_1^{\mathbf{V}^T} + \sigma_2 \mathbf{c}_2^{\mathbf{U}} \mathbf{r}_2^{\mathbf{V}^T} + \sigma_3 \mathbf{c}_3^{\mathbf{U}} \mathbf{r}_3^{\mathbf{V}^T} + \sigma_4 \mathbf{c}_4^{\mathbf{U}} \mathbf{r}_4^{\mathbf{V}^T}} + \boxed{\sigma_5 \mathbf{c}_5^{\mathbf{U}} \mathbf{r}_5^{\mathbf{V}^T}}$$

$$\mathbf{M} - \mathbf{M}_a = \mathbf{M}_{diff} = \sigma_5 \mathbf{c}_5^{\mathbf{U}} \mathbf{r}_5^{\mathbf{V}^T}$$

```
# reconstruct a low-rank approximation of M
RankApp = 4
Ma = U[:,:RankApp] @ (np.diag(S[:RankApp]) @ VT[:RankApp,:])
print(f'abs(Ma-M).sum()={abs(Ma-M).sum()}')
```

```
abs(Ma-M).sum()=171.7903369122311
```

```

#verify result
Mdiff = U[:,RankApp:] @ (np.diag(S[RankApp:]) @ VT[RankApp:,:])
print(f'abs(Mdiff).sum()={abs(Mdiff).sum()}')
```

```
abs(Mdiff).sum()=171.7903369122311
```

Expectation

When we approximate a matrix we expect that the difference between them is small.

How do we define **small for matrices**?

Induced Matrix Norms - *induced by Vector norms*

Focus on the transformation by the matrix:

$$\mathbf{M} : \mathbb{R}^n \rightarrow \mathbb{R}^m$$

$$||\mathbf{M}||_{q,p} = \sup_{\mathbf{x} \neq \mathbf{0}} \frac{||\mathbf{M}\mathbf{x}||_p}{||\mathbf{x}||_q}$$

common usage:

$$||\mathbf{M}||_{p,p} = ||\mathbf{M}||_p = \sup_{\mathbf{x} \neq \mathbf{0}} \frac{||\mathbf{M}\mathbf{x}||_p}{||\mathbf{x}||_p}$$

Induced Matrix Norms

Commonly used induced p-norms $$\|\mathbf{M}\|_p = \sup_{\mathbf{x}\neq \mathbf{0}} \frac{\|\mathbf{M}\mathbf{x}\|_p}{\|\mathbf{x}\|_p}$$

$$\|\mathbf{M}\|_1 = \max_{j=1,\ldots,n} \left\{ \sum_{i=1}^{d} |m_{ij}| \right\} \rightarrow \text{max. column sum}$$

$$\|\mathbf{M}\|_2 = \max_{j=1,\ldots,n} \left\{ \lambda_j(\mathbf{M}^T\mathbf{M}) \right\}^{\frac{1}{2}} = \sigma_{max}(\mathbf{M}) \rightarrow \text{max. singular value}$$

a.k.a **spectral** norm

$$\|\mathbf{M}\|_\infty = \sup_{\mathbf{x}\neq \mathbf{0}} \frac{\|\mathbf{M}\mathbf{x}\|_\infty}{\|\mathbf{x}\|_\infty} = \max_{i=1,\ldots,d} \left\{ \sum_{j=1}^{n} |m_{ij}| \right\} \rightarrow \text{max. row sum}$$

Elementwise Matrix Norms

$$\|A\|_* = \operatorname{trace}\left(\sqrt{A^*A}\right) = \sum_{i=1}^{\min\{m,n\}} \sigma_i(A),$$

Elements of an $d\,x\,n$ **matrix** is treated as a **vector of length** $(d * n)$, and vector norms are used

Check out Schatten norms as well

Nuclear Norm a.k.a. Trace Norm: $\|\mathbf{M}\|_N = trace(\sqrt{\mathbf{M}^T\mathbf{M}}) = \sum_{i=1}^{rank(\mathbf{M})} \sigma_i(M)$

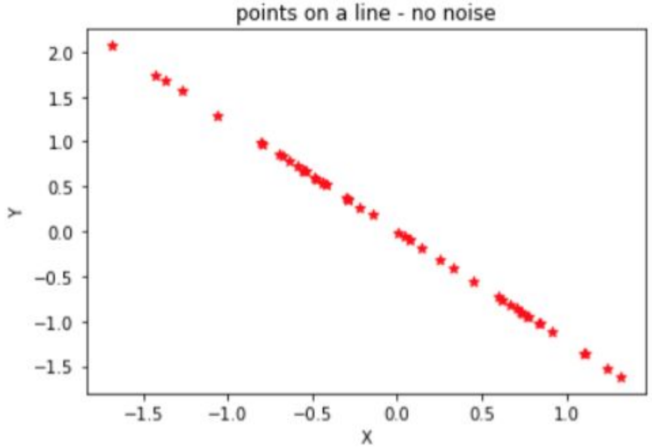
Frobenius Norm: $\|\mathbf{M}\|_F = \sqrt{\sum_{i=1}^{d}\sum_{j=1}^{n} |m_{ij}|^2} = \sqrt{trace(\mathbf{M}^T\mathbf{M})} = \sqrt{\sum_{i=1}^{rank(\mathbf{M})} \sigma_i^2(\mathbf{M})}$

Max Norm: $\|\mathbf{M}\|_{max} = \max_{ij} |m_{ij}|$

Example: Data from a line

```python
# generate data on a line
M = DataInSubspace(2, 50, 1)
print(f'rank(M) = {np.linalg.matrix_rank(M)}')
CData(M, 'points on a line - no noise')
u, s, vt = np.linalg.svd(M, full_matrices=False)
print('Singular values are:')
MatPrint(s)
```

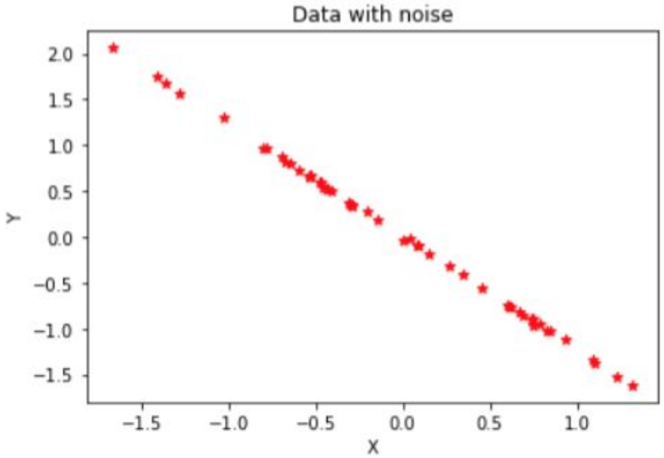
```
rank(M) = 1
```




```
Singular values are:
Matrix:
[8.32756054e+00 4.58017925e-16]
```

```python
# add a bit of noise to the data
Mn = M + 0.01 * np.random.randn(M.shape[0], M.shape[1])
print(f'rank(Mn) = {np.linalg.matrix_rank(Mn)}')
CData(Mn, 'Data with noise')
un, sn, vtn = np.linalg.svd(Mn, full_matrices=False)
print('Singular values are:')
MatPrint(sn)
```

```
rank(Mn) = 2
```

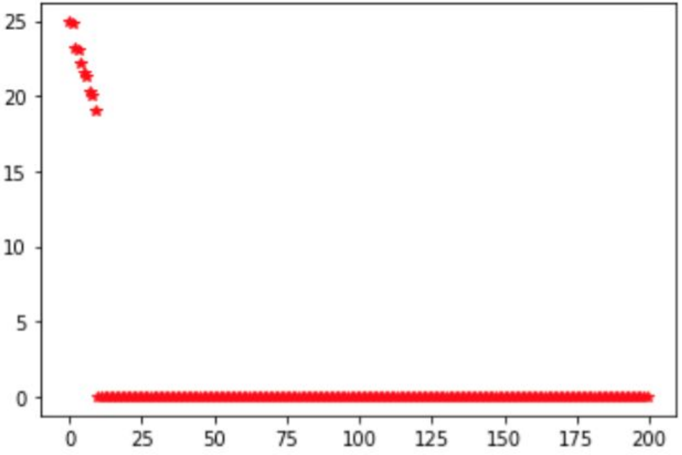



```
Singular values are:
Matrix:
[8.31458315 0.07017064]
```

Example: Data from a line

```python
# generate data on a line
M = DataInSubspace(200, 500, 10)
print(f'rank(M) = {np.linalg.matrix_rank(M)}')
u, s, vt = np.linalg.svd(M, full_matrices=False)
plt.plot(s, 'r*')
```

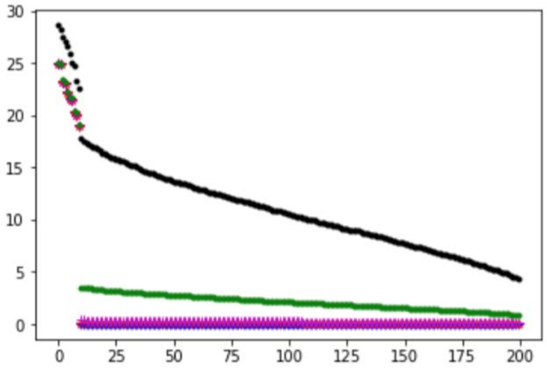
```
rank(M) = 10
[<matplotlib.lines.Line2D at 0x7ffa4fd47f28>]
```




```python
plt.plot(s, 'r*')

plotwith = ['b+', 'm+', 'g.', 'k.']
noiselevels = [0.000001, 0.01, 0.1, 0.5]
# add some tiny amount of noise
for i, nLevel in enumerate(noiselevels):
  Mn = M + nLevel * np.random.randn(M.shape[0], M.shape[1])
  print(f'rank(Mn1) = {np.linalg.matrix_rank(Mn)}')
  u, s, vt = np.linalg.svd(Mn, full_matrices=False)
  plt.plot(s, plotwith[i])
```

```
rank(Mn1) = 200
rank(Mn1) = 200
rank(Mn1) = 200
rank(Mn1) = 200
```



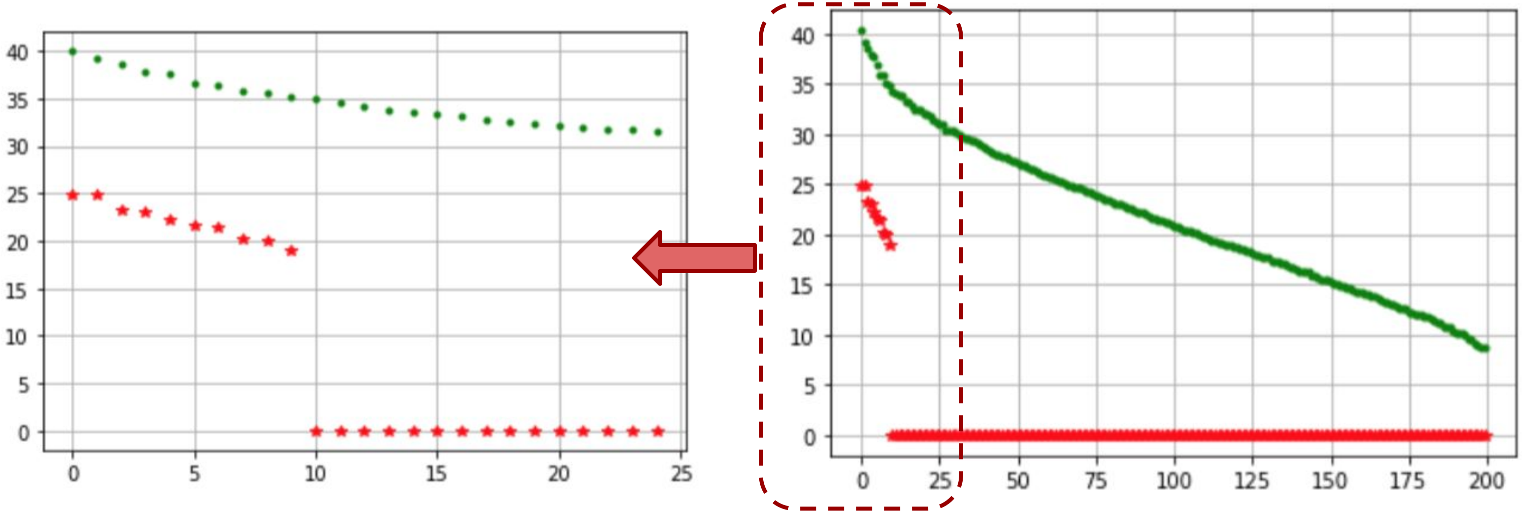
Rank Estimation

By checking the change in singular values can you estimate the rank of matrix $\mathbf{M}$?

```python
plotwith = ['r*', 'g.']
noiselevels = [0.0, 1.0]
# add some tiny amount of noise
for i, nLevel in enumerate(noiselevels):
  Mn = M + nLevel * np.random.randn(M.shape[0], M.shape[1])
  print(f'rank(Mn1) = {np.linalg.matrix_rank(Mn)}')
  u, s, vt = np.linalg.svd(Mn, full_matrices=False)
  plt.plot(s, plotwith[i])
  plt.grid(True)

```

```
rank(Mn1) = 10
rank(Mn1) = 200
```



Condition Number

Depends on the selected norm, but for convenience we will stick with the following:

Given matrix $\mathbf{M}$

$$\kappa(\mathbf{M}) = \frac{\sigma_{max}(\mathbf{M})}{\sigma_{min}(\mathbf{M})}$$

Low condition number $\rightarrow$ *well-conditioned* case .... *how low can it get?*

High condition number $\rightarrow$ *ill-conditioned* case .... *how high can it get?*

Rank Estimation - *one such* Rule of thumb

A rule of thumb is to keep 90% of the energy in $\mathbf{\Sigma}$

where total eneryg is defined as:

$$E(\mathbf{\Sigma}) = \sum_{\forall i} \sigma_i^2$$

then energy in the first k sigular values are

$$E(\mathbf{\Sigma}_k) = \sum_{i=1}^{k} \sigma_i^2$$

and

$$\frac{E(\mathbf{\Sigma}_k)}{E(\mathbf{\Sigma})} \approx 0.9$$

Update 90 to a proper value depending on the problem

Rank Estimation - *another* Approach : $\frac{4}{\sqrt{3}}$

As an exercise:

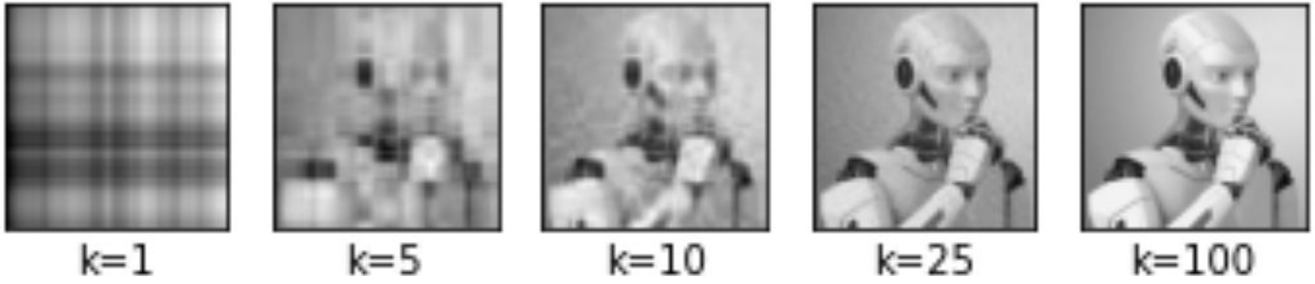
Check out this paper: https://arxiv.org/abs/1305.5870

And this video:
https://youtu.be/epoHE2rex0g?list=PLMrJAkhIeNNSVjnsviglFoY2nXildDCcv

SVD: Not a compression method but...

```python
def ImageInRankK(img, k):
  try:
    U, S, VT = np.linalg.svd(img, full_matrices=False)
    return np.matmul(U[:,:k], np.matmul(np.diag(S[:k]),VT[:k,:]))
  except:
    print('it did not work out, try again later')
    return img
```

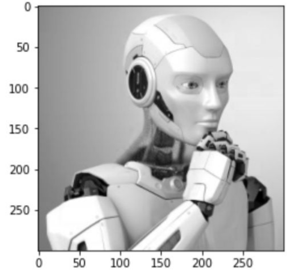
```python
rankK = [1, 5, 10, 25, 100]
nPlots = len(rankK)
for i, k in enumerate(rankK):
  plt.subplot(1, nPlots, i+1)
  #plt.axis('off')
  plt.xticks([])
  plt.yticks([])
  plt.imshow(ImageInRankK(img, k), cmap=plt.get_cmap("gray"))
  plt.xlabel(f'k={k}')
```



Play with some images

```python
# read the image file into a variable
img = io.imread('robot.jpg')
# let's play with the image's red channel only
img = img[:,:,0]
plt.imshow(img, cmap=plt.get_cmap("gray"))
```

<matplotlib.image.AxesImage at 0x7ffa4f055048>



coming up...

More matrices and matrices and matrices and ...