Matrix methods – Singular value decomposition

MATH 2740 – Mathematics of Data Science – Lecture 09

Julien Arino
julien.arino@umanitoba.ca

Department of Mathematics @ University of Manitoba

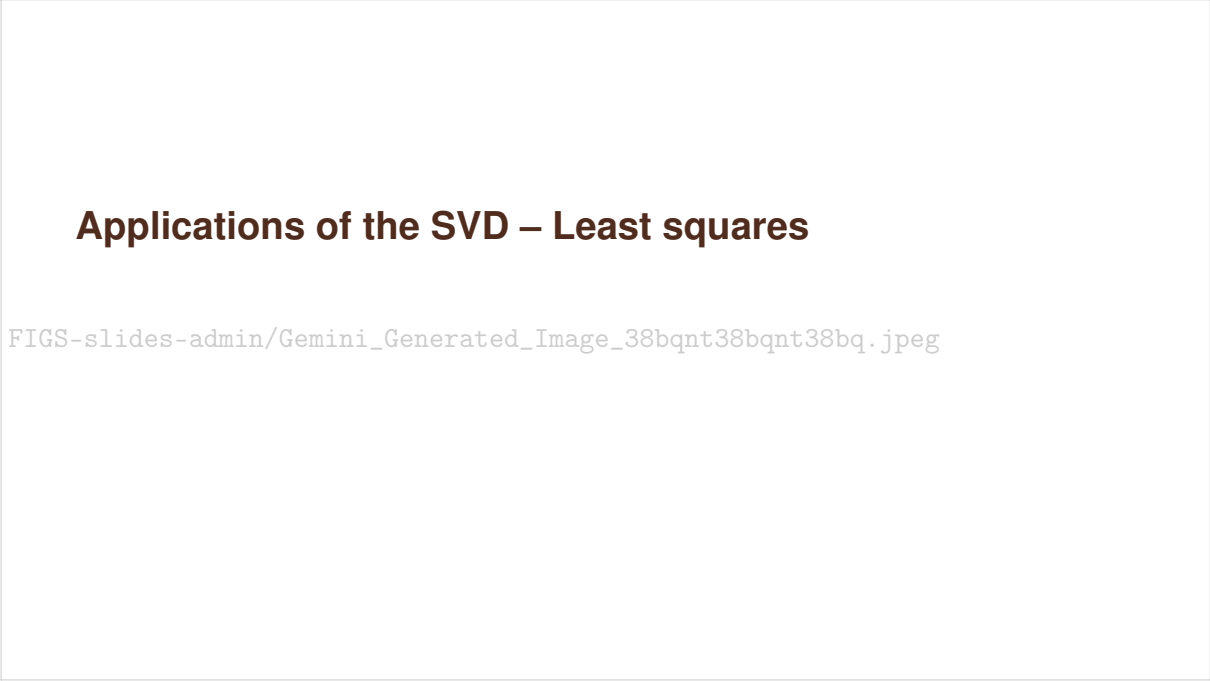
Fall 202X

The University of Manitoba campuses are located on original lands of Anishinaabeg, Ininew, Anisininew, Dakota and Dene peoples, and on the National Homeland of the Red River Métis. We respect the Treaties that were made on these territories, we acknowledge the harms and mistakes of the past, and we dedicate ourselves to move forward in partnership with Indigenous communities in a spirit of Reconciliation and collaboration.

Outline

Applications of the SVD – Least squares

Applications of the SVD – Least squares



Applications of the SVD

Many applications of the SVD, both theoretical and practical..

1. Obtaining a unique solutions to least squares when $A^T A$ singular
2. Image compression

Least squares revisited

Theorem 7

Let $A \in \mathcal{M}_{mn}$, $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{b} \in \mathbb{R}^m$. The least squares problem $A\boldsymbol{x} = \boldsymbol{b}$ has a unique least squares solution $\tilde{\boldsymbol{x}}$ of minimal length (closest to the origin) given by

$$\tilde{\boldsymbol{x}} = A^+ \boldsymbol{b}$$

where A^+ is the pseudoinverse of A

Definition 8 (Pseudoinverse)

$A = U\Sigma V^T$ an SVD for $A \in \mathcal{M}_{mn}$, where

$$\Sigma = \begin{pmatrix} D & 0 \\ 0 & 0 \end{pmatrix}, \text{ with } D = \text{diag}(\sigma_1, \ldots, \sigma_r)$$

(D contains the nonzero singular values of A ordered as usual)

The **pseudoinverse** (or **Moore-Penrose inverse**) of A is $A^+ \in \mathcal{M}_{nm}$ given by

$$A^+ = V\Sigma^+ U^T$$

with

$$\Sigma^+ = \begin{pmatrix} D^{-1} & 0 \\ 0 & 0 \end{pmatrix} \in \mathcal{M}_{nm}$$

Applications of the SVD – Least squares

Applications of the SVD – Compressing images

FIGS-slides-admin/Gemini_Generated_Image_6dnw706dnw706dnw.jpeg

Compressing images

Consider an image (for simplicity, assume in shades of grey). This can be stored in a matrix $A \in \mathcal{M}_{mn}$

Take the SVD of A. Then the small singular values carry information about the regions with little variation and can perhaps be omitted, whereas the large singular values carry information about more “dynamic” regions of the image

Suppose A has r nonzero singular values. For $k \leq r$, let

$$A_k = \sigma_1 \boldsymbol{u}_1 \boldsymbol{v}_1^T + \cdots + \sigma_k \boldsymbol{u}_k \boldsymbol{v}_k^T$$

For $k = r$ we get the usual outer product form (**??**)

Load the image using `bmp::read.bmp`

```
my_image = bmp::read.bmp("../CODE/Julien_and_friend_1000x800.bmp")
my_image_g = pixmap::pixmapGrey(my_image)
my_image_g
```

```
## Pixmap image
##   Type          : pixmapGrey
##   Size          : 800x1000
##   Resolution    : 1x1
##   Bounding box  : 0 0 1000 800
```

Doing the computations “by hand”

```
M = my_image_g@grey
MTM = t(M) %*% M
# Ensure matrix is symmetric
MTM = (MTM+t(MTM))/2
ev = eigen(MTM)
```

Given the size and nature of the entries, the matrix M^TM is symmetric only to `1e-5` precision, so we use a little trick to make it symmetric no matter what: take the average of M^TM and its transpose MM^T

Which version of the algorithm to use?

Make zero the eigenvalues that are close to zero (200 out of 1000)

```
ev$values = ev$values*(ev$values>1e-10)
```

Can we use the algorithm for all eigenvalues being distinct or do we have repeated ones?

```
any(duplicated(ev$values[ev$values>1e-10]))

## [1] FALSE
```

So we can use the standard algorithm

Computing the SVD

```
idx_positive_ev = which(ev$values>1e-10)
sv = sqrt(ev$values[idx_positive_ev])
```

Computing the SVD

Then $D = \text{diag}(\sigma_1, \ldots, \sigma_r)$, V is the matrix of normalised eigenvectors in the same order as the σ_i and for $i = 1, \ldots, r$

$$\mathbf{u}_i = \frac{1}{\sigma_i} A\mathbf{v}_i$$

ensuring that $\|\mathbf{u}_i\| = 1$

```
D = diag(sv)
V = ev$vectors[idx_positive_ev, idx_positive_ev]
c1 = colSums(V)
for (i in 1:dim(V)[2]) {
    V[,i] = V[,i]/c1[i]
}
```

Computing the SVD

Finally, we compute the $\boldsymbol{u}_i$'s

```
U = M %*% V %*% diag(1/sv)

## Error in M %*% V: non-conformable arguments

r = length(sv)
im = list(u=U, d=sv, v=V)

## Error:  object 'U' not found
```

Using built-in functions

We can also use the built-in function `svd` to compute the SVD of M

```
M.svd = svd(M)
```

The results are stored in a list with components `u`, `d` and `v`

Make function to recreate an image from the SVD

Given the SVD `im` of an image and a number of singular values to keep `n`, we can recreate the image using the function `compress_image`

We output the new image, but also, the amount of information required to encode this new image, as a percentage of the original image size

```
compress_image = function(im, n) {
  if (n > length(im$d)) {
    # Check that we gave a value of n within range, otherwise
    # just set to the max
    n = length(im$d)
  }
  d_tmp = im$d[1:n]
  u_tmp = im$u[,1:n]
  v_tmp = im$v[,1:n]
  # We store the results in a list (so we can return other information)
    out = list()
    # First, compute the resulting image
    out$img = mat.or.vec(nr = dim(im$u)[1], nc = dim(im$v)[1])
    for (i in 1:n) {
        out$img = out$img + d_tmp[i] * u_tmp[,i] %*% t(v_tmp[,i])
    }
```

```
    # Values of the "colours" must be between 0 and 1, so we shift and resc
    if (min(min(out$img)) < 0 ) {
        out$img = out$img - min(min(out$img))
    }
    out$img = out$img / max(max(out$img))
    # Store some information: number of points needed and percentage of the
    out$nb_pixels_original = dim(im$u)[1] * dim(im$v)[2]
    out$nb_pixels_compressed = length(d_tmp) + dim(u_tmp)[1]*dim(u_tmp)[2] +
    out$pct_of_original = out$nb_pixels_compressed / out$nb_pixels_original
    # Return the result
    return(out)
}
```

Recreating the image

We can now recreate the image using the function `compress_image`

```
new_image = my_image_g
M.svd = svd(M)
M_tmp = compress_image(M.svd, 2)
new_image@grey = M_tmp$img
plot(new_image)
```

Using $n = 2$ singular values

Uses 0.56% of the original information

Using $n = 5$ singular values

Uses 1.41% of the original information

Using $n = 10$ singular values

Uses 2.81% of the original information

Using $n = 20$ singular values

Uses 5.63% of the original information

Using $n = 50$ singular values

Uses 14.07% of the original information