

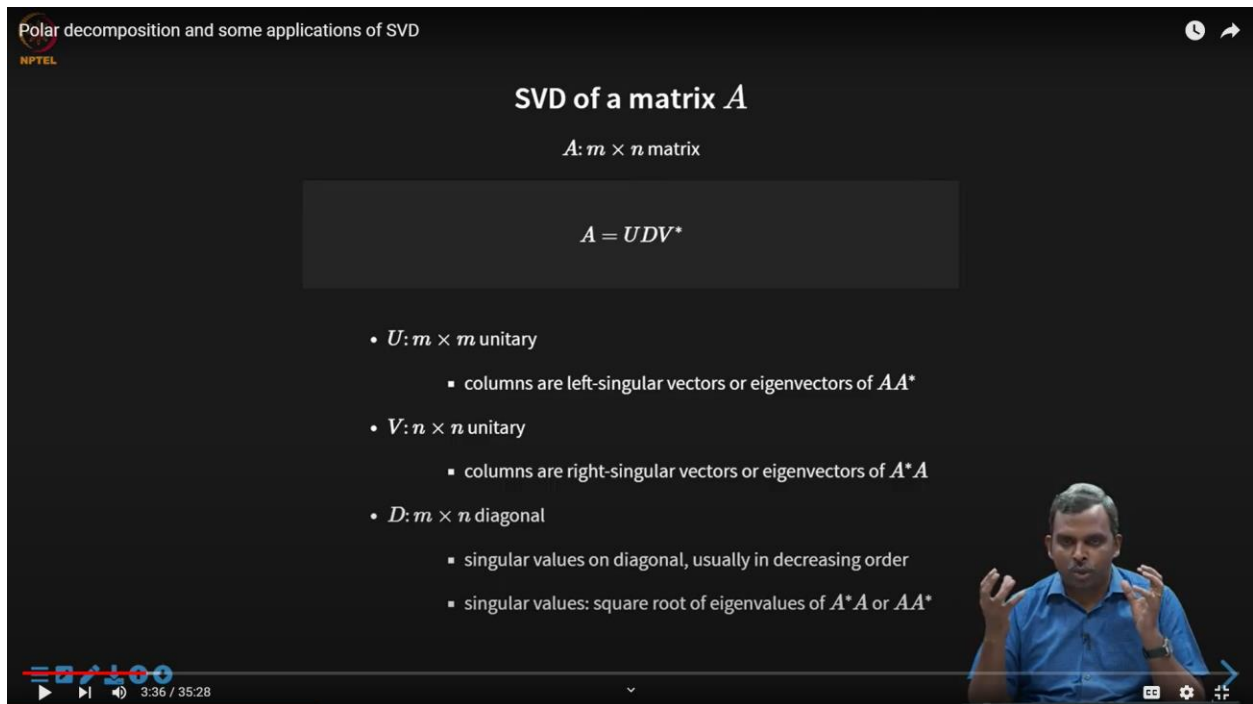
Applied Linear Algebra
Prof. Andrew Thangaraj
Department of Electrical Engineering
Indian Institute of Technology, Madras

Week 12

Polar decomposition and some applications of SVD

Hello and welcome to this lecture. We are going to talk about two different topics. One for a very quick wind up of this polar decomposition which I did not mention before. We will quickly look at that. It's a small and simple consequence of SVD and then we look at some two or three applications of SVD, how it's used in practice. And it'll also give you a sense of how to think of matrices, how to think of linear maps, how do they show up in real life from nowhere, right? You know? I mean we already saw in the page rank algorithm how, you know, applications rely on this intuition you build with Linear Algebra about vectors and matrices and linear maps and how you sort of look at, how do you sort of, you know, generate them in real life problems when maybe they don't seem to appear.

(Refer Slide Time: 03:36)



Polar decomposition and some applications of SVD

SVD of a matrix A

$A: m \times n$ matrix

$$A = UDV^*$$

- $U: m \times m$ unitary
 - columns are left-singular vectors or eigenvectors of AA^*
- $V: n \times n$ unitary
 - columns are right-singular vectors or eigenvectors of A^*A
- $D: m \times n$ diagonal
 - singular values on diagonal, usually in decreasing order
 - singular values: square root of eigenvalues of A^*A or AA^*

3:36 / 35:28

So there are some nice models and some principles here on how to construct these matrices in the real world and how to associate linear maps with them, how to think of SVD etc. in that context. So hopefully these applications will give you more of a flavour of how these things work in real life, okay? So let us go ahead and look at them. So once again this course is not primarily about

applications, it's mostly about the theory and all that, giving you a solid foundation there. But also I think it's good to sometimes see how in the applied world people pick up and connect the dots and use the theory in a very creative, clever fashion, okay? So let us go ahead and see that.

So a quick recap. I will skip the recap in this lecture. So we've been looking at SVD this week. And SVD of a matrix A has this very simple description. A is an $m \times n$ matrix. A can be written as a product of three matrices UDV^* . V^* , star denotes conjugate transpose. So that's UDV^* . U is an $m \times m$ unitary matrix, columns are left singular vectors or basically eigenvectors of AA^* . AA^* is as you know a self-adjoint operator. So it would have an orthonormal basis of eigenvectors. And those are called the left singular vectors of A . And V is $n \times n$ unitary. V^* like I told you is the conjugate transpose. Columns of V are right singular vectors or they are basically the eigenvectors of A^*A . A^*A again is a self-adjoint operator and it's like, it will have an orthonormal eigenvector basis and one of those becomes the set of right singular vectors for A . And that makes up the matrix V . And this D diagonal matrix is an $m \times n$ diagonal matrix sort of. D is like the heart of A , right? So essence of A and it's got the singular values on the diagonal. Usually you arrange it in decreasing order, singular values are non-negative so you put them in decreasing order along the main diagonal of D . Diagonal is basically i and j are the same, right? Row and column indices are the same, that's the main diagonal.

(Refer Slide Time: 05:44)

Polar decomposition and some applications of SVD
NPTEL

SVD of an operator and polar decomposition

$A: n \times n$ matrix representing T

$$A = UDV^* = (UV^*)(VDV^*)$$

- UV^* : unitary, represents an isometry
- VDV^* : self-adjoint, represents $\sqrt{T^*T}$

Handwritten notes on the right side of the slide:

$$(UV^*)^* = VU^*$$

$$UV^* (UV^*)^* = UV^* VU^* = I$$

Video player controls at the bottom show a progress bar at 5:44 / 35:28.

And what are singular values? Singular values are square root of the eigenvalues of A^*A or AA^* , right? So of course the numbers differ here. But, you know, you can figure out how many you need

for an $m \times n$ diagonal matrix, okay? So that is the thing. So you can see that, you know, V^* is just a unitary transform. It's not doing much. It's rotating in some way. And D is what is really doing the transform and it's just a scaling of the coordinates in the basis of the columns of V . And then U is also just a unitary operator which is again rotating. So you can think of U and V as not doing much, and D is the one that's really representing what A is doing. And so this is a very good way to picture a matrix which could be very big and unwieldy in big large values of m and n . But D you know has got very few entries and it gives you a very clear idea of what it is. So this is the power of SVD. It gives you a good feel for how to work with matrices, okay? So that's SVD. We have seen that before.

(Refer Slide Time: 07:06)

Polar decomposition and some applications of SVD
NPTEL

SVD of an operator and polar decomposition

A : $n \times n$ matrix representing T

$$A = UDV^* = (UV^*)(VDV^*)$$

- UV^* : unitary, represents an isometry
- VDV^* : self-adjoint, represents $\sqrt{T^*T}$

Polar decomposition

For any operator $T : V \rightarrow V$, there exists an isometry S such that $T = S\sqrt{T^*T}$

- analogous to $z = e^{i\arg(z)}|z|$ for a complex number z

7:06 / 35:28

So there is a connection between SVD and what's called the polar decomposition of an operator. So now supposing you take A to be a square matrix $n \times n$ representing some operator. Maybe in the standard basis, right? So then we know A has this UDV^* SVD again, right? And even if it is an operator that is true. Now I will do this little trick of introducing this V^*V in the middle, right? So this is UD . In between U and D we'll introduce this V^*V , right? So that is what I have done here. You can see between U and D I have introduced V^*V . V^*V is I , right? It is not doing anything. But then I'll bracket them differently. I'll bracket the VDV^* separately and bracket the UV^* separately. So now UV^* is unitary, okay? So you can check this very quickly that UV^* will be unitary. If you want I can check that for you, right? $(UV^*)^*$ is VU^* . So UV^* multiplied by $(UV^*)^*$, product of two unitary matrix operators, unitary matrices is unitary, okay? So that's something you should be able to easily show. V^*V is I then UU^* is I so this becomes I . So UV^* is unitary, so it basically represents

an isometry. So UV^* is an isometry. But then look at what is this. VDV^* . VDV^* is self-adjoint, right? And since D basically, in VDV^* , D is like the singular values which are the eigenvalues of $\sqrt{T^*T}$, the columns of V are the eigenvectors of $\sqrt{T^*T}$. So this is nothing but the operator $\sqrt{T^*T}$, okay? So VDV^* represents $\sqrt{T^*T}$, okay? So this is just a reinterpretation of what SVD represents when you have a being an operator $n \times n$, okay?

So the polar decomposition essentially states this fact, okay? It says for any operator $T:V \rightarrow V$, there exists an isometry S such that T becomes equal to $S\sqrt{T^*T}$, okay? So why is this called polar decomposition? So this is sort of like this, it is analogous to... For a complex number you can always write the complex number as its absolute value multiplied by $e^{i\theta}$, right? So θ is that argument of z . So this is what this is doing. So the $\sqrt{T^*T}$ is sort of like the absolute value of T in some sense because it is a positive square root, right? It's a positive operator. And then S is like a $e^{i\theta}$, right? It's a rotation. It's like a unitary operator S . So this is always true for operators. So this sort of advances the analogy that we've been talking about between complex numbers and operators in some sense, okay? So this is polar decomposition. I'm just mentioning it. It's a simple fact from SVD, okay? All right. So that's all the theory that we want to do in this class. So we've sort of concluded this whole course as far as the theory is concerned.

(Refer Slide Time: 10:54)

The video player shows a Jupyter Notebook titled "Matrix approximation through images". The code in the notebook is as follows:

```

In [ ]: import cv2
import numpy as np
import matplotlib.pyplot as plt

In [ ]: img = cv2.imread('/content/drive/My Drive/Colab notebooks/GC_IITM_lowres.jpg',0)

In [ ]: img1 = img[:, :, 0]

In [ ]: plt.imshow(img1, cmap='gray')

Out[ ]: <matplotlib.image.AxesImage at 0x7f58724dc50>

In [ ]: [U,D,Vs] = np.linalg.svd(img1)

```

The notebook also displays a grayscale image of palm trees and a building. The video player shows the time 10:54 / 35:28.

So to really close it out, I want to show you some two or three applications of SVD. In fact three applications for SVD maybe and then show you how people think of, you know, how people bring in these linear algebraic notions in practice and use them very effectively. We already saw them

with eigenvalue applications. So here we'll see with SVD applications, okay? So and with that we'll close the course. So this will be sort of the last lecture in that sense.

Okay. So the first thing I want to show you is through this Colab sheet that I've shared with you through Github. So this sort of shows you how matrices can be approximated and I'm depicting it with images, okay? So always when you think of a matrix, an image is an easy thing for you to represent, right? So if you have a large matrix, you can represent it as an image and I have done that here and sort of shown SVD in action. So this is, you can think of it as an application in image compression. People use SVD in some image compression sometimes. So this sort of illustrates how that would come about, okay? So this is a Python3 Jupyter Notebook, Colab notebook. So you have to import some packages. There is the cv2 package which I'm using. So this is a very, it's an OpenCV package. Open computer vision package. It's a very popular python package for reading images and all that. All I'm doing is for reading images, okay? So I took a Gajendra Circle picture, IITM GC picture in low resolution. JPEG picture. You can also take bigger images if you want. So this picture is available from the IITM website through the Instagram page that we have. It's one of those pictures. Or you can take any other picture also if you want to play around with it.

(Refer Slide Time: 13:21)

The video player displays a Jupyter Notebook titled "Matrix approximation through images". The notebook content includes:

- A plot of an image (Gajendra Circle) with axes from 0 to 400 on the x-axis and 0 to 300 on the y-axis.
- Code cell []: `[U,D,Vs] = np.linalg.svd(img1)`
- Code cell [70]: `plt.plot(D)` and `plt.yscale('log')`
- A log-linear plot of singular values. The x-axis ranges from 0 to 250, and the y-axis ranges from 10^0 to 10^4 . The plot shows a sharp initial drop followed by a gradual decay.
- Code cell [71]: `img_app = np.matmul(U[:, :10], D[:, :10, np.newaxis]**Vs[:, :10, :])` and `plt.imshow(img_app, cmap='gray')`
- Output [71]: `<matplotlib.image.AxesImage at 0x7f586f6a4438>`

The video player interface shows a progress bar at 13:21 / 35:28 and a small video inset of the presenter in the bottom right corner.

So I've put a comma 0 here. I believe the 0 just imports it as a grayscale image that's what this cv2.imread does. So basically I'm reading it in grayscale. If you don't read in grayscale, JPEG images have, you know, multiple attributes. Some of them have three attributes. RGB. Some of them have a fourth one also. So it can become, you know... So it will be, you know, it will be a

two dimensional matrix sort of thing. But each entry will have R value, G value, B value. So red green blue value. It can also have other values associated with that. So if you read grayscale, it will read only a matrix, okay? So that's why I am reading it in grayscale. So this is just grayscale. It's just from say 0 to 255 or something like that, okay? So from white to black, or black to white, 0 to 255. So that is the number that it would represent. And this picture actually happens to be a very large resolution picture. Some, I think, if I am not wrong, it's some 3000×4000 or some such very big resolution picture. So what I'm doing in the next step is to down sample it. So this is a way to down sample in python very easily. I can do `::8`. So this, when I do colon colon, some 0 in the last row are filled in automatically. Zero in the last columns are filled in automatically. So it will just keep the every 8×8 value. So I am just making it a smaller matrix just for convenience. You can have it as a large matrix also if you want. Today's computing algorithms are quite good, they don't mind working with large sizes. But anyway for us the size is not so important, so we can work with a smaller matrix here. So there is this `matplotlib.pyplot` package which you can import. And then you can do this `imshow` command and you would use the `cmap='gray'`. That's important. Otherwise it will show you some strange colour. So if you do `cmap='gray'` and plot this it will show you the image.

So hopefully you still remember the Gajandra Circle, the two elephants and the familiar picture. You can see it's sort of low resolution and all that. But it's okay, I mean at least you recognize the picture. So this is my matrix. So this `img1` is a matrix, okay? So it is some size matrix. It's just got values from 0 to 255 in it and it can be visualised as an image, okay? So I believe it's got 281 rows and 500 columns, okay? So that's this image, okay? So once I have a matrix, whatever that matrix may be, I can run SVD, okay? So on this image I run an SVD, okay? So I use the Numpy package for it which I imported there. `np.linalg` linear algebra module. Inside that there is an SVD function and I run `img1` on it. So you run any matrix through this, it will give you the U , D and V^* , okay? So I am just calling it $U D V_s$. And U is the, you know, unitary matrix with the left singular eigenvectors. V_s is the unitary matrix with the right singular eigenvectors and D is the singular values in decreasing order, okay? So it's good to plot D . So remember, it will have 281 singular values, right? So that's what it will show you. Or maybe 500, but the rest are all zeros, right? So it's not so interesting. So 281 is the non-trivial singular values. And it's good to nicely see them, plot them in log scale and see, okay? For such large matrices you can see look at this big drop, right? So there are a few eigenvalues which are, few singular values, should be careful, yes few singular values which are really, really large. Like, you know, 10^4 , 10^5 etc. And then there is the sharp drop, it comes down to 10^3 . 10^3 is still big. But remember the matrix entries are 0 to 255. So you have to have a sense of how big the, you know, the norm of every row or column is in the matrix. So it's quite big. So 10^3 maybe is not so big. Then definitely 10^2 and all is not big. But you can see this big drop. So this is sort of a nice feature you'll see. Any big matrix you take and find its singular values, it will have this kind of feature. It'll usually drop off quite quickly. So this is sort of showing you that it's, you know, the matrix, every entry in the matrix is not all that independent. There is some, you know, structure there when you see this drop, okay? So that's the, those are good interpretations to keep, okay?

So to bring out that interpretation I am going to try and approximate this image. How am I approximating this image? So normally when you have SVD, you have the full matrix, no? UDV^* , okay? I'll only take 10 columns of U and the top 10 singular values from D and the 10 rows of V^* , okay? So maybe I should write this down. So we are writing A , U and then you have D and then you have V , okay? So this, let's say this is V^* . U, D, V^* , right? So D will have the singular values on the diagonal like that, okay? This would have the, you know, the f_1, f_2, \dots , along the columns and this would have the e_1^*, e_2^*, \dots , along the rows, right? So e_1, e_2, \dots are the, you know, the right singular eigenvectors. f_1, f_2, \dots are the left singular eigenvectors. $\sigma_1, \sigma_2, \dots$ are the singular values, okay? So if you multiply the whole thing together, you will get A . Now what is this approximation doing? I am only going to take, say, the 10 columns here. I'll only take the 10×10 here, okay? So this is m let us say. So I will take $m \times 10$ and then here also I will only take, only on this side, I will take $10 \times n$, okay? So this is n . So if you do this, notice the largest sigma values are being accounted for, right? See A becomes $\sigma_1 f_1 e_1^* + \sigma_2 f_2 e_2^* + \dots$, right, all the way till the last one. The last one will be what? $\sigma \dots$

(Refer Slide Time: 15:49)

The slide content includes:

- Top left: "Polar decomposition and some applications of SVD" and "NPTEL" logo.
- Center: Three matrices. The first is A of size $m \times 10$ with columns f_1, f_2, \dots . The second is D of size 10×10 with diagonal elements $\sigma_1, \sigma_2, \dots, \sigma_m$. The third is V^* of size $10 \times n$ with rows e_1^*, e_2^*, \dots .
- Bottom center: The equation $A = \sigma_1 f_1 e_1^* + \sigma_2 f_2 e_2^* + \dots + \sigma_m f_m e_m^*$ with $\sigma_1 > \sigma_2 > \dots > \sigma_m$.
- Bottom right: A small video inset of a man in a blue shirt.
- Bottom left: Video player controls showing "15:48 / 35:28".

So you can sort of write this down. So let us say $\sigma_m f_m e_m^*$, right? So it stops there. There are 281. After that it's just zero eigenvalues. Let's think about how it will work out. The next ones do not matter. I am assuming m is smaller than n , okay? So all of them together will give you A . And I know $\sigma_1 > \sigma_2 > \dots$, okay? Okay. So if I want a good approximation, I can stop somewhere here, okay? So in this product I will stop somewhere here. Stop at the 10th point. Stop at $\sigma_{10} f_{10} e_{10}^*$, okay? So that is the same as in this matrix picture, stopping like this, okay? So when you do that,

I get a pretty reasonable, I hope I will get a reasonable approximation for A , okay? So that's the thinking when people do these kind of approximations. So keep that in mind, okay? So that is what I have done here. You can see I have taken matrix multiplication. I am multiplying U only the 10 columns, first 10 columns, D only the 10×10 and Vs also I am keeping only the 10 rows, okay? So this is a command for that. And then I am going to plot this `img_app` which is some approximation, okay? So you can see here I took only 10 eigenvalues. There are, singular values, there are 281 of them. I took only 10.

(Refer Slide Time: 17:46)

Polar decomposition and some applications of SVD
NPTEL

Matrix approximation through images

```
In [72]: img_app1 = np.matmul(u[:, :50], d[:, :50, np.newaxis]) * v[:, :50, :];  
plt.imshow(img_app1, cmap='gray')
```

```
Out[72]: <matplotlib.image.AxesImage at 0x7f586f9d4cf8>
```

imagecompressionsvd.ipynb hosted with ❤️ by GitHub

17:46 / 35:28

So notice here the, I mean some prime features have already come, right? So maybe there is some confusion here. On the top you can sort of maybe expect that. But at the bottom you see that this pattern got accurately captured. And then there are these, you know, the elephants and you sort of see the fountain. I mean all of us may be familiar with this picture, but still, you know, you can see some important characteristics sort of show up. And maybe there is some sky here. You can sort of see important features show up even with just 10 singular vectors taken into account, okay? Of course I can take more. I can take 50. and at 50 you start seeing really the, almost a pretty good depiction. If you take 50 singular values, you can see a pretty good depiction of the GC picture, okay? So this sort of visually shows you how these singular values help in matrix approximation. You have a big matrix and you sort of expect that there's not so much freedom in the matrix, that every entry was sort of independently generated, okay? So that's at the heart of assuming what A matrix should be, right? So if you assume that every matrix of that, every entry in that matrix is sort of independently generated, then you won't get any such great simplification. But you are

expecting that there's lots of connection between the entries and those are captured by this SVD sort of things with the singular values being very large. So this picture of how the singular values fall is very typical and it's very important. So this sort of quickly tells you that there is a big drop here and you can use it for approximation, okay? So once again this Colab notebook you can click on it and open it on Colab. It's shared with you. And this image you can replace with any other image. I'm not sharing this image, it's there on the IITM web page in case you want to go see it on the Instagram page, okay?

All right. So that's one illustration/application of SVD that we have seen. Let's see one more. This is from the world of wireless communications. In particular what's called Multiple Input Multiple Output wireless communications. MIMO wireless communications. So in wireless communications there is a transmitter and a receiver. I am not going to go into great detail here, but just from a high level, what is the goal? The transmitter will have multiple antennas in which it will send something. And each antenna it will send something and the receiver will have multiple antennas and it will receive whatever is being sent as a superposition at the receiver, okay? So this is at a high level. I will write down what is there in more detail here. So you have transmitter with n antennas and you have receiver with m antennas. What is transmitted at a particular instance is a vector from the complex vector space. So each entry that this antenna transmits can be complex. You can talk about why it can be complex etc. Just think of complex as ordered pair of reals, right? So you can send a complex number. It's okay. It's not wrong, okay?

(Refer Slide Time: 21:41)

Polar decomposition and some applications of SVD
NPTEL

MIMO wireless communications

- Transmitter with n antennas and receiver with m antennas
- Transmit $x = (x_1, \dots, x_n) \in \mathbb{C}^n$
- Gain from antenna j to antenna i is $h_{ij} \in \mathbb{C}$

$$\text{Channel } H = \begin{bmatrix} h_{11} & & h_{1n} \\ & \dots & \\ h_{m1} & & h_{mn} \end{bmatrix}$$

- Receive $y = Hx + z$, where z is random noise

Goal: recover x from y

21:39 / 35:28

So you can send complex numbers here on the antennas and the gain from antenna j on the transmitter to antenna i at the receiver we will denote as h_{ij} and this will also be complex, okay. So there is, so this needs some theory here, but I am skipping all that. Say it's complex, okay? So the channel is usually written as a matrix $m \times n$, okay? This m and n usually are not very high. I mean, not like thousands, you know, but at least in today's systems they could be 5, 10, things like that. It's not like, you know, very small. Today's systems, particularly the 5G systems, may even have very large values of m and n , okay? So they have some connections with the, you know, wavelength of the EM waves being used and all that. So let's not go into great detail there. But this matrix sort of captures what the wireless channel would do from the transmitter to the receiver, okay? So conveniently you can write what is being received as a $Hx + z$, okay? So this Hx is basically the, you know, map, right? x is getting transformed through the linear map H . H is like a linear map. So look, notice how these linear maps start showing up in a very interesting way. $+ z$ where z is random noise, okay? So you can see how these matrices and linear maps sort of show up in some way, you know. Physically what happens at the receiver is that, you know, the signals superpose and add, okay? Whatever electromagnetic fields that are generated at the antenna, they tend to superpose and add, right? So that's like the theory that we use for modelling it. And so linearity sort of comes in so many different ways. And by the time those signals propagate into your chips and all that, this is a pretty good model for what you want to receive, okay? So the received y is $Hx + z$. And z is random noise. So this noise comes from various electronics at the receiver and it's unavoidable. Usually unless you go to really low temperatures, okay? Anyway...

(Refer Slide Time: 22:47)

Polar decomposition and some applications of SVD
NPTEL

SVD in MIMO

SVD of channel matrix

$$H = UDV^*$$

- Transmit Vx
 - V is unitary, $\|Vx\| = \|x\|$
 - $y = UDx + z$

Handwritten: $y = UDV^*Vx + z$

22:47 / 35:28

So what is the goal? Finally you have $y = Hx + z$, okay? So all that theory is, I mean physics is fine, but finally the equation is $y = Hx + z$. And you have to recover x from y , right? So that's the goal. So you've been given y . You have to figure out what x is, okay? So you can say, you know, you want to do an inverse for H . But usually that inverse is done through SVD in a very clever way, because, you know, you have to be mindful of z , okay? So inverting something is always dangerous when you have random noise because the noise will get boosted up by that inverse. Inverse multiplies the noise also, right? So you have to be careful about it. So SVD gives you a very nice way of handling all this situation. So how is SVD used in MIMO? So this channel matrix H you can do UDV^* on it, right? SVD on it. And then you send the v to the transmitter, okay? So maybe you have some other means of sending it. You send it to the transmitter. So the transmitter, instead of sending x , when it wants to send x , it will send Vx , okay? Now Vx is okay because it's got the same norm as x , okay? So you will not be spending more energy than sending x , okay? So this norm sort of represents the energy of the transmission usually, okay? So Vx is pretty good. So you can send Vx instead of x , you are not losing much there.

(Refer Slide Time: 24:01)

The screenshot shows a video player interface with a slide titled "SVD in MIMO". The slide content is as follows:

Polar decomposition and some applications of SVD
 NPTEL

SVD in MIMO

SVD of channel matrix

$$H = UDV^*$$

- Transmit Vx
 - V is unitary, $\|Vx\| = \|x\|$
 - $y = UDx + z$
- At receiver, compute U^*y
 - $U^*y = Dx + U^*z$
 - U^* is unitary, $\|U^*z\| = \|z\|$
- Divide by singular values in D to estimate x

SVD plays an important role in practical cellular systems!

The video player shows a progress bar at 23:58 / 35:28 and a small video thumbnail of the presenter in the bottom right corner.

So now notice what happens when you send Vx instead of x . Your y which is $Hx + z$, okay, the V and V^* will cancel and you will simply get a UD , okay? So you can see that, right? So y equals UDV^* , you are not getting x , now you are getting Vx , right? $+ z$. So this will cancel and you will get this, okay? So that's that. So it's good. So y is already a little bit simple. Now at the receiver you compute U^*y , okay? So you know U also. So you compute U^*y . Now notice what happens. U^*y is just $Dx + U^*z$, okay? So notice the only thing that multiplies z is a unitary operator U^*

and that is quite okay because the norm doesn't change, okay? So the noise doesn't get blown up beyond what it was originally in norm. So that's very nice, okay? And what you're left with is just Dx . What is Dx ? Dx is just Dx , right? It's just, it's a diagonal operator. And you can divide by the singular values in D to estimate x , okay? So this turns out to be one of the optimal ways of receiving $y = Hx + z$. As in finding out x from $y = Hx + z$. And this gives tremendous intuition to engineers on how to design systems on the ground and how to build receivers to receive wireless transmissions, okay? And so much so that this kind of SVD and MIMO play an important role in practical cellular systems. So all your cell phones even on the communication side have SVD sitting there, not just on the image processing side like we saw before, okay? So this is SVD in action in communication. So people who do wireless communication courses probably will meet this at one point or the other, okay?

The last application I want to show you is what are called recommender systems. So all of us use these OTT apps now. Maybe not all of us, a significant fraction of us use these OTT apps like Netflix and Prime Video and all that. And there's always, there's something called recommended movies, okay? So maybe you're wondering how is it that people come up with those recommended movies, what is a strategy for doing that? It's called recommender systems. You can do it in various other systems also. You go to a retail company, retail website and buy on a retail website, buy something, it will give you some recommendations based on what you've bought before and based on other things. So all of that sort of depend on these recommender systems and ideas from this. So how does that work? So here is a picture of, here is a description of, a very loose simple description, not a very concrete description of how it works, okay? So let us take the movie user situation, right? So if you take Netflix or any other OTT platform, there are a lot of movies in that platform and it's got lots of users, right? Maybe lakhs and crores of users, right? And thousands of movies, okay? Now everybody is not going to watch every movie for sure, right? So it's not possible. But you can imagine a matrix A where a_{ij} would be the rating of the movie i by user j . In case user j watches movie i , a_{ij} would be the rating, okay? So now clearly this is not a complete matrix. As in, everybody has not watched every movie and everybody has not rated every movie. So the fact that they watched that movie itself is an indication to me. It shows that, you know, maybe they liked it, whatever, right? So that is at one level. So even at least if they watched it, you know, okay, they are interested vaguely in that movie. But if they give you a rating, then you know for sure that, you know, they really liked it or hated it or whatever. I mean, you know, the whole thing about how they like the movie. But even then everybody is not going to watch every movie, right? So this big matrix is sort of like an incomplete matrix, you don't know many entries, you know only some entries, right? So that's the crux of the matter, okay? So you should remember that. So notice how in OTT platform where there is no connection to Linear Algebra, how matrices show up, see? So you notice that there are these variables, there are these variables or entities like users and movies, right? And these are connected, you know? Users rate movies, they watch movies, they rate movies. So whenever things are connected, you want to think of these entities as vectors and and look at their, you know, operations and all that. So you will see how this works.

It is very neat how to think of, you know, the linear algebraic settings in places where there is no obvious way to have Linear Algebra, right? So this is something interesting, okay?

(Refer Slide Time: 27:15)

The screenshot shows a video lecture slide with a dark background. At the top left, it says 'Polar decomposition and some applications of SVD' and 'NPTEL'. The main title is 'Recommender systems'. Below that, it says 'm movies, n users'. Then, 'A: m x n matrix with a_{ij} = rating of movie i by user j'. Underneath, it says 'Matrix factorization and latent factors' and 'A = QP'. There are two bullet points: the first is 'Q: m x k, P: k x n, k << m, n' with sub-bullets for 'k latent factors (movie genres, say)', 'Q[i, :]: i-th row of Q or i-th movie expressed as vector over factors', and 'P[:, j]: j-th column of P or j-th user expressed as vector over factors'. The second bullet point is 'a_{ij} = <Q[i, :], P[:, j]>'. At the bottom, it says 'Goal: find best factorization that can predict missing a_{ij}'s'. A video player interface is visible at the bottom with a red progress bar and a timestamp of 27:15 / 35:28. A small inset video of a man in a blue shirt is in the bottom right corner.

So you already have a matrix. And what is the goal? The goal I will tell you. So the ultimate goal, let me... You can skip this matrix factorization, latent factors for a little while. The goal ultimately is to predict the missing a_{ij} s, okay? So forget about... I think I jumped the gun a little bit here. But anyway... Predict the missing a_{ij} s. So this is the goal, okay? So overall goal in any recommender systems is to predict the missing a_{ij} . So let me just go back to the original thing. So I want to predict the missing a_{ij} s from whatever I know from the matrix A , okay? So how do I go about doing it? So one very popular method uses something called matrix factorization, okay? So that is what I am showing you again. And use something called latent factors, okay? So you sort of imagine that these movies and users are vectors and sort of think of them as vectors or think of a vector as representing their characteristics, okay? So what is the characteristic of a movie? Maybe it has so much violence in it, maybe it has so much, you know, romance in it. Maybe it has so much comedy in it, okay? There's these genres and you want to, you want to put what fraction of the movie goes into the genre. Maybe it's positive-negative, right? It's not good comedy, maybe not good romance, not good violence, whatever. So you have positive and negative possibilities, right? So these are these latent factors and what people try to do from this matrix A , incomplete matrix A is to come up with a factorization of the matrix. Notice how subtle this is, okay? So this is where the critical idea comes in. They try to come up with the factorization of this matrix A as QP . So what is Q and what is P ? Q is an $m \times k$ matrix, P is a $k \times n$ matrix. So this would be like

something like this, right? So it will be a Q times a P , okay? So here are movies, here are users. And this is this k , right? So m , n and this is k , right?

(Refer Slide Time: 30:55)

Polar decomposition and some applications of SVD
NPTEL

Recommender systems

m movies, n users

A : $m \times n$ matrix with a_{ij} = rating of movie i by user j

Matrix factorization and latent factors

$$A \approx QP$$

- Q : $m \times k$, P : $k \times n$, $k \ll m, n$
 - k "latent" factors (movie genres, say)
 - $Q[i, :]$: i -th row of Q or i -th movie expressed as vector over factors
 - $P[:, j]$: j -th column of P or j -th user expressed as vector over factors
- $a_{ij} = \langle Q[i, :], P[:, j] \rangle$

30:55 / 35:28

So these k are sort of latent factors. So notice the data has come in and you do not really have to worry so much about what this k is and all that. Maybe experts would know. And they would come up with what fraction this is and all that. But we are just data driven, right? So you just take that data and try and do a factorization here. QP . Put ks , come up with some k columns here, k rows here. And there are multiple ways to do it. There's no unique way to do it, right? So so many different ways. And instead of even equals, usually it's only an approximation, okay? So it's not equal, okay? You may not be able to exactly get it. So it's only approximation. I put equal to here. Notice that this is only an approximation, okay? And you want to associate, sort of think that these k factors are genres. But they need not be, right? So it could be anything else. I don't know what it is. But notice what I am going to think of. The $Q[i, :]$, right, the $Q[i, :]$ is the column here. So this is $Q[i, :]$, okay? This is sort of MATLAB, Python representation for the i^{th} row of Q . So this is the i^{th} movie sort of represented as a vector here. Vector over these factors, okay? The same thing with the j^{th} user, right? So I am going to think of this as $P[:, j]$, the j^{th} column of P , okay? And that is, that sort of represents the user, right? It represents the users liking or disliking of different factors of these latent factors or genres that it may be, okay? And notice a_{ij} is simply the inner product of $Q[i, :]$ and $P[:, j]$, right? a_{ij} , $(i, j)^{\text{th}}$ entry, the rating or whether the j^{th} user is going to like the i^{th} movie is sort of like this inner product, okay? So notice how subtle and interestingly things like inner product, matrix factorization and, you know, factors that make

people like movies, not like movies are entering this picture and how this problem of building recommender systems for an OTT platform, right, is being expressed in linear algebraic terms, okay? So this is the nice thing about how when you know the theory and when you understand the theory and how you can relate it to how things in practice work, how these matrix matrices are supposed to look, then you can use all these things and come up with very nice ideas.

(Refer Slide Time: 32:46)

Polar decomposition and some applications of SVD
NPTEL

SVD in recommender systems

SVD of A
 $A = UDV^*$

can be used for approximations such as
 $A \approx PQ$
 $P: m \times k, Q: k \times n, k \ll m, n$

Recommender systems use SVD as a starting point to optimize the factorization!

32:46 / 35:28

So what is the challenge in recommender systems? So find best approximate factorization that can predict the missing a_{ij} s properly. So you get data. So you will have this data, you can generate some... Take a partial A , and you would also know the ratings of some movies, keep some as training data, keep some as test data, right? And then try and come up with an algorithm for doing these kind of factorizations, okay? So a very nice application of Linear Algebra and SVD into... I mean, how does SVD come in? I have to show you that. So we can see how matrix factorization helps in picturing how recommender systems may be built, okay? Okay. So how does SVD come in? SVD, I mean, maybe current systems, really high flying systems today maybe they don't use SVD totally but SVD is a very good starting point here. Why is that? Because SVD also gives you some sort of an approximate factorization, right? So $A = UDV^*$, if you write it like that, this can be used to build an approximate factorization which is what we did in the image also, right? You take only some columns, some rows, and you get an approximate factorization with k being much much smaller than m or n , okay? So, I mean, there are many more ideas today in this recommended systems. SVD is a good starting point for optimising such factorizations, okay? So hopefully these three applications give you a better grounding of how to think of matrices and how to think of

these linear algebra and vectors and factorization, inner products and how they show up in various places in totally unobvious ways, okay?

(Refer Slide Time: 34:28)

Polar decomposition and some applications of SVD

Recap

- Vector space V over a scalar field $F = \mathbb{R}$ or \mathbb{C}
- $m \times n$ matrix A represents a linear map $T : F^n \rightarrow F^m$
 - $\dim \text{null } T + \dim \text{range } T = \dim V$
 - Solution to $Ax = b$ (if it exists): $u + \text{null}(A)$
- Four fundamental subspaces of a matrix
 - Column space, row space, null space, left null space
- Eigenvalue λ and Eigenvector $v: Tv = \lambda v$
 - There is a basis w.r.t. which a linear map is upper-triangular
 - If there is a basis of eigenvectors, linear map is diagonal w.r.t. it
- Inner products, norms, orthogonality and orthonormal basis
 - There is an orthonormal basis w.r.t. which a linear map is upper-triangular
 - Orthogonal projection: distance from a subspace
- Adjoint of a linear map: $\langle Tv, w \rangle = \langle v, T^*w \rangle$
 - $\text{null } T = (\text{range } T^*)^\perp$
- Self-adjoint: $T = T^*$, Normal: $TT^* = T^*T$
- Complex/real spectral theorem: T is normal/self-adjoint \leftrightarrow orthonormal basis of eigenvectors
- Positive operators: self-adjoint with non-negative eigenvalues
- Isometries: normal with absolute value 1 eigenvalues
- Singular values/vectors of T : Eigenvalues/eigenvectors of T^*T
 - $A = UDV^*$, diagonal w.r.t. singular vector basis

34:28 / 35:28

So this is the last lecture and is a good place to stop in the course. We've seen quite a few wide ranging topics, okay? And these have all been summarised in the recap here. So let's just quickly go through that recap, okay? So all these things we saw in theory. We started with vector spaces over a scalar field being real or complex. And then we saw how linear maps are very interesting and how they are represented by matrices and this Fundamental Theorem of Linear Maps which relates the null space and range space dimension. And we used it to solve linear equations say $Ax = b$. So that's very powerful. Linear equations show up in so many different places. And then we saw four fundamental subspaces. And then very importantly eigenvalues, eigenvectors, how these one dimensional invariant subspaces of a linear map really capture what the linear map is. And this helped us in so many applications, right? So particularly eigenvalues, the eigenvectors, they show how, you know, linear systems evolve over time. And then even importance of certain graph nodes and all that. Page rank and all that. Very interesting algorithms are possible with eigenvalues and eigenvectors. Then we move down to inner products, norms, orthogonality orthonormal basis. So this gives you, this really, projections and inner products are very, very interesting. They measure correlations, they measure so many connections between vectors and they're very intuitive and very useful. And then we looked at adjoint, self-adjoint. Normal operators are very important classes of operators and general operators are related to them in very nice ways. And this complex, real spectral theorem. So it's very important how, you know, self-adjoint, normal operators have

an orthonormal basis of eigenvectors and they become diagonal in orthonormal basis, okay? Then we saw positive operators, how they are connected to quadratic forms, maximising, minimising quadratic forms. That has a wealth of applications as well, right? Quadratic forms are also very useful to capture some real-life behaviour, okay? Then we saw isometries, operators which, you know, are like rotation, they do not change the relative behaviour of vectors, okay? So this is very important to know. And finally the last thing we saw was singular vector, singular value decomposition, singular values for a general linear map and how you get this wonderful UDV^* . And how that gives you factorization and how that gives you abilities to use it in so many different applications. Approximating a matrix, factoring a matrix, in communications, how to, you know, modify your signals that are transmitted and received so that you can easily decode in the presence of noise, okay? So that's a good recap. And thanks for being along in this course. And hope you have enjoyed it as much as I've enjoyed putting this course together for you. And all the best. Thank you.