McMaster University ENGINEERING

Chemical Engineering 4H03

Mathematical Derivation of PCA

Jake Nease
McMaster University

Objectives for this Class

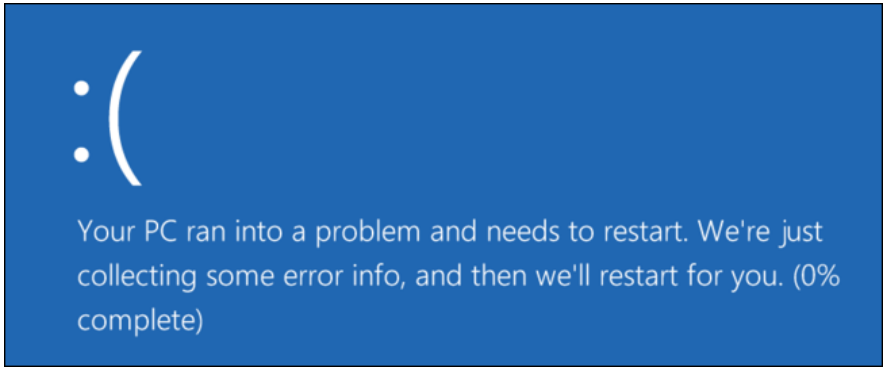
- We want to compute the values p_k in each loadings vector for our principle components
  - This will complete our understanding of how PCA works
  - Will allow us to code our own PCA functions… Both in this course and)hopefully) your careers!

- How?
  1. Discussing model residuals (graphically)
  2. Discussing model residuals (numerically)
  3. Connection of these concepts into **Eigenvalue Decomposition** (fundamental math basis of PCA)

McMaster University ENGINEERING

Error Analysis

Learn from your mistakes

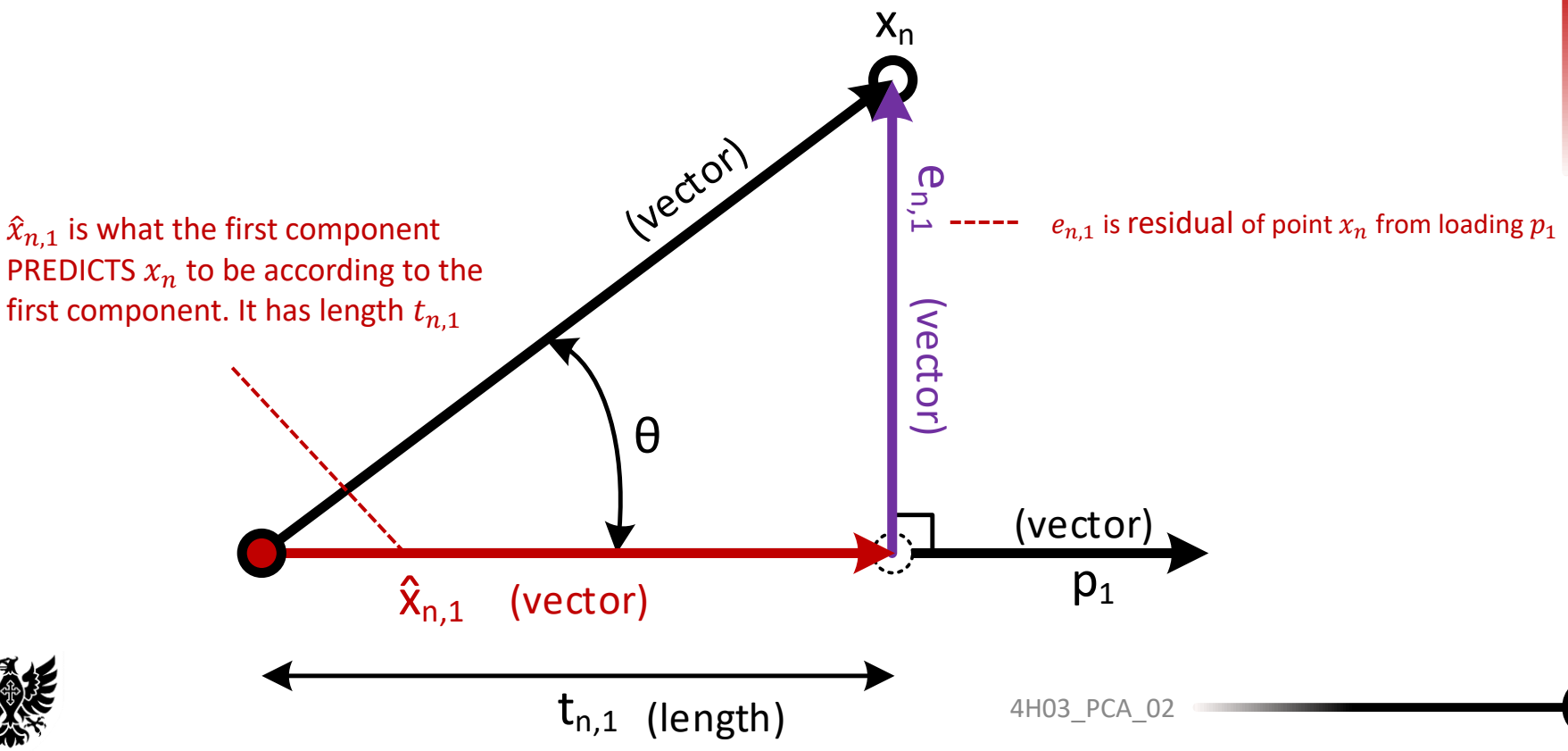


https://www.howtogeek.com/wp-content/uploads/2013/05/windows-8-blue-screen-header.png

Consider the PCA Model Prediction

- A point x_n may be reconstructed as the sum of two vectors
  - One is the vector (projection) of length t along the loading p
  - The second represents the orthogonal distance (we'll call that the **residual** e) from the loading to the point

- Example: residual of point x_n from first component



Consider the PCA Model Prediction

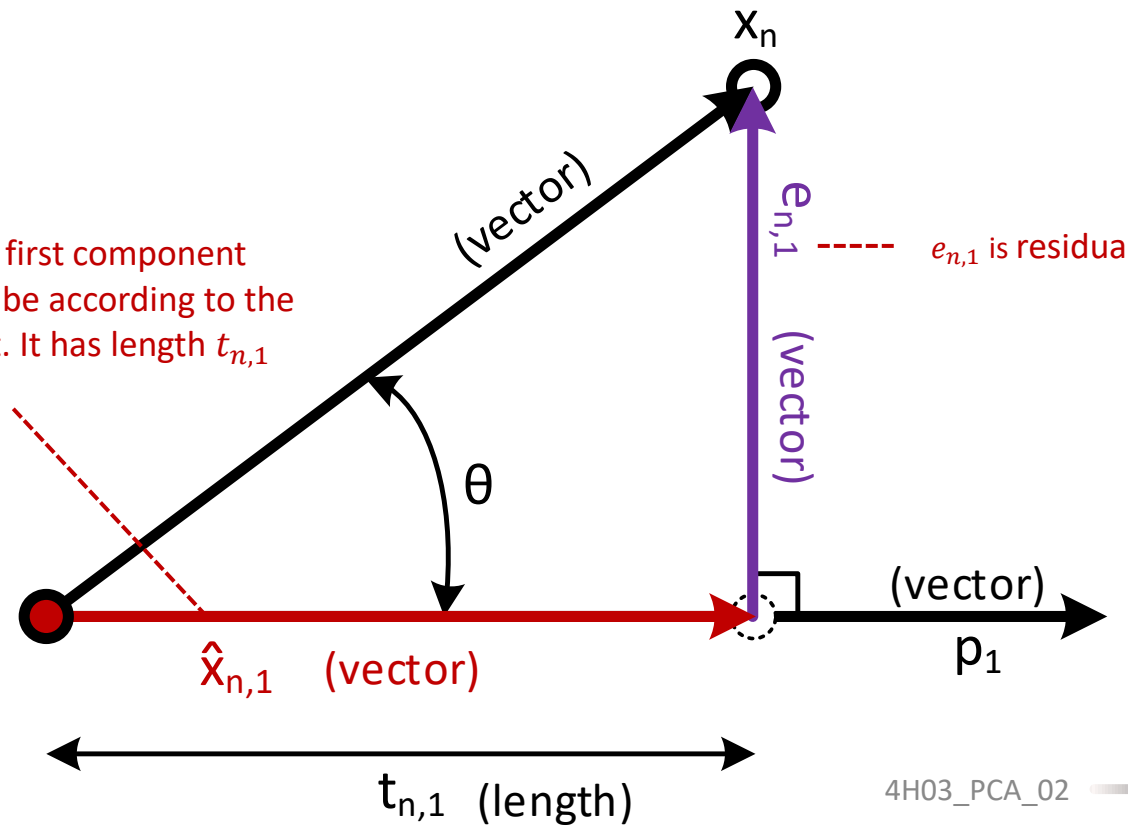
- We may therefore compute $\hat{x}_{n,1}$ as:

$$\hat{x}_{n,1}^T = t_{n,1}\, p_1^T$$
$$(1 \times K) = (1 \times 1)(1 \times K)$$

Conveniently, $t_{n,1}$ is a scalar that represents the DISTANCE along p_1…
So we are just multiplying the p_1 vector by a scalar here

- The value $\hat{x}_{n,1}$ is our "best" prediction of x_n according to the first component

$\hat{x}_{n,1}$ is what the first component PREDICTS x_n to be according to the first component. It has length $t_{n,1}$



$e_{n,1}$ is residual of point x_n from loading p_1

Calculating the Residual

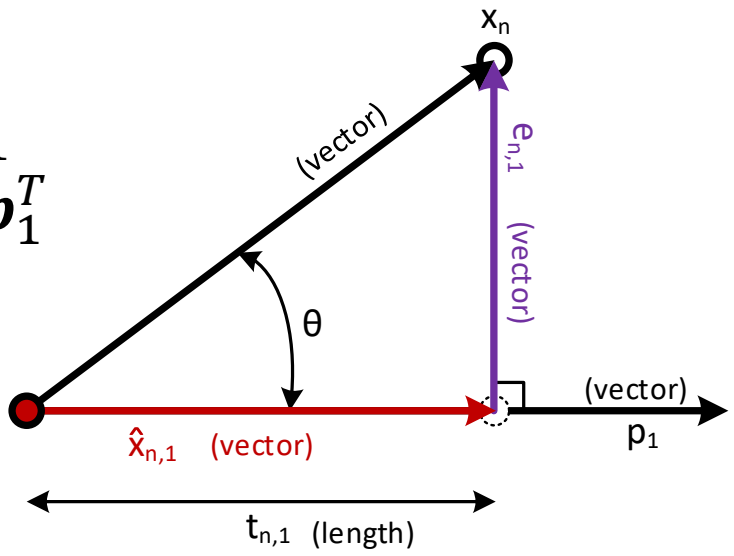
- We can therefore compute x_n as the sum of the prediction of x_n according to the first component plus the **residual error** left behind from that component:

$$\boldsymbol{x}_n = \hat{\boldsymbol{x}}_{n,1} + \boldsymbol{e}_{n,1}$$

- So, the entire **ROW** of residuals for a point x_n is $e_{n,1}$

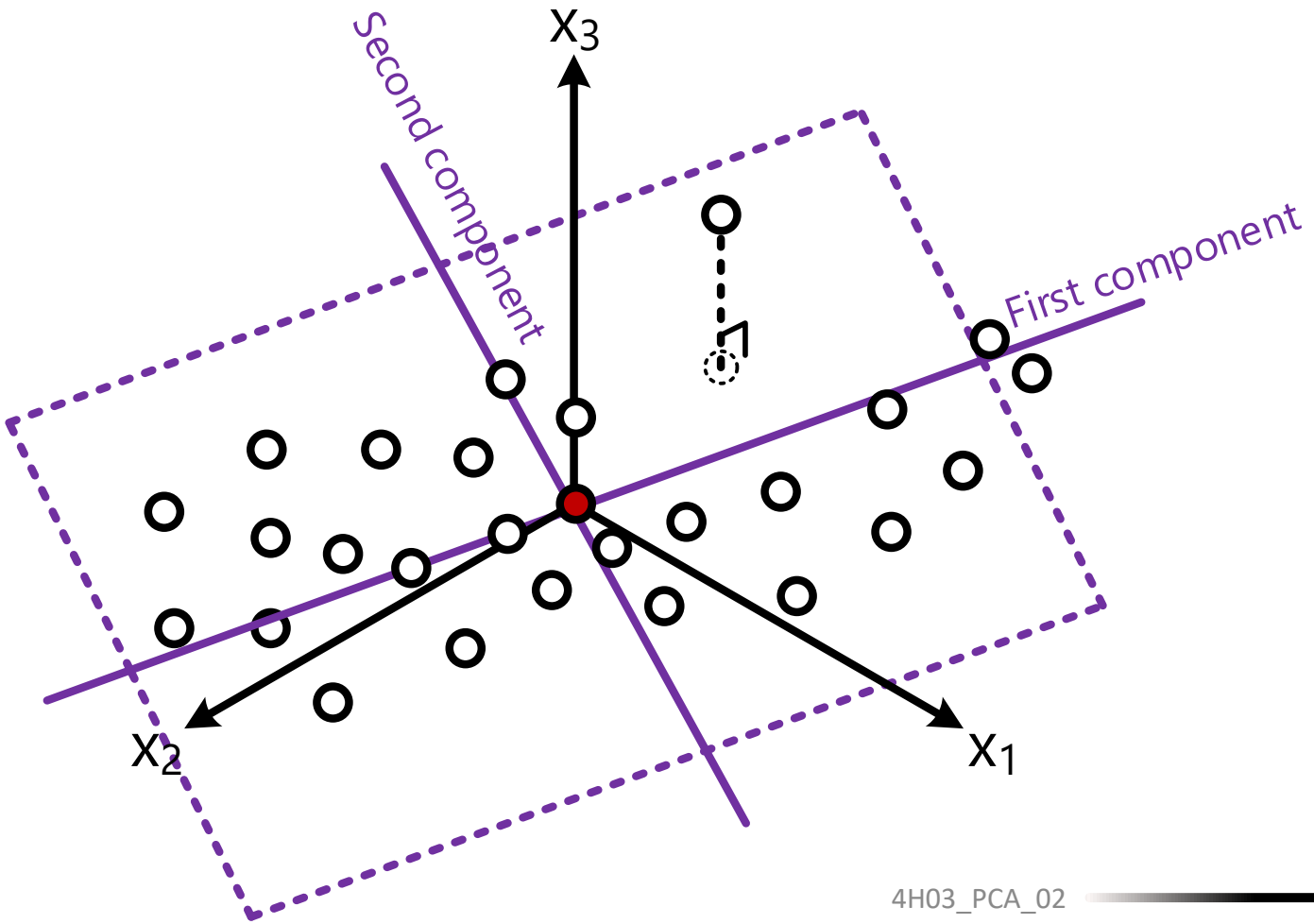
$$\boldsymbol{e}_{n,1}^T = \boldsymbol{x}_n^T - \hat{\boldsymbol{x}}_{n,1}^T$$
$$\boldsymbol{e}_{n,1}^T = \boldsymbol{x}_n^T - t_{n,1}\,\boldsymbol{p}_1^T$$

Recall that this is how we compute $\hat{x}_{n,1}^T$



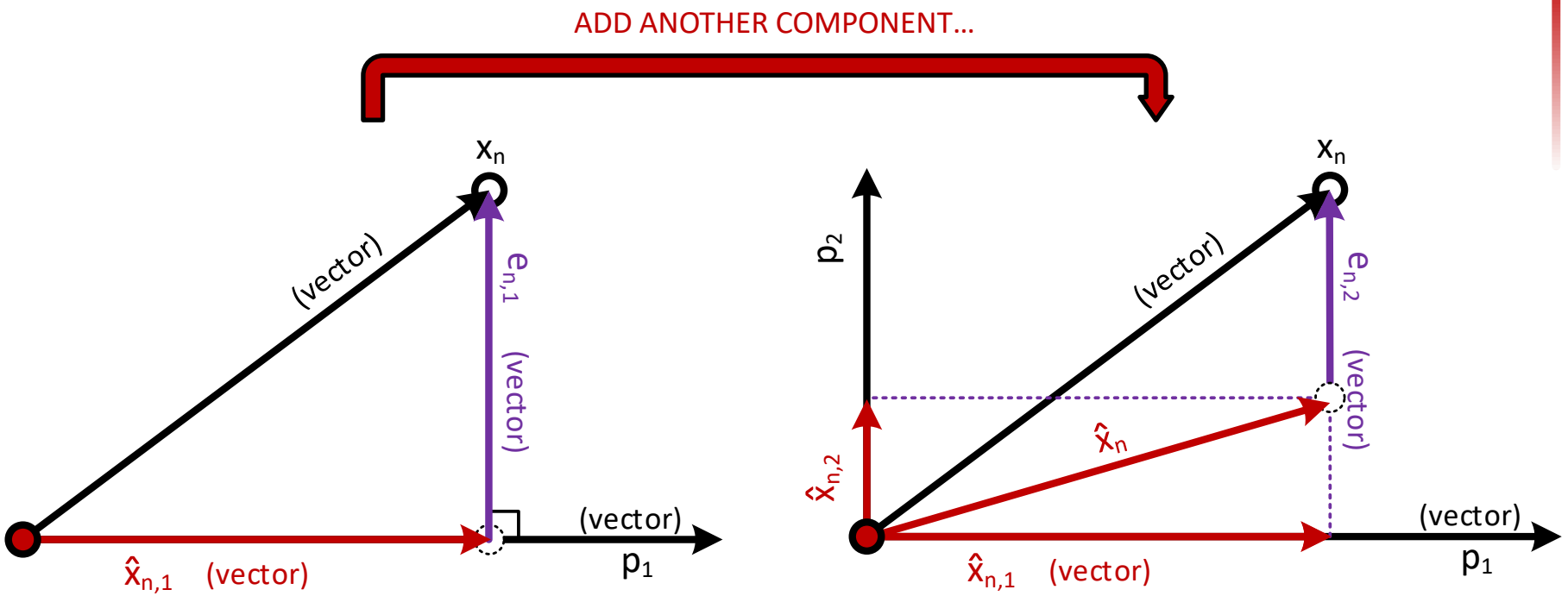
Question

- So if that is how the residual is computed… what do you expect to happen if we add another component?



Question

- So if that is how the residual is computed… what do you expect to happen if we add another component?
  - We should find the residual error $\boldsymbol{e}_{n,2}$ is LOWER since we have another dimension attempting to explain the data:



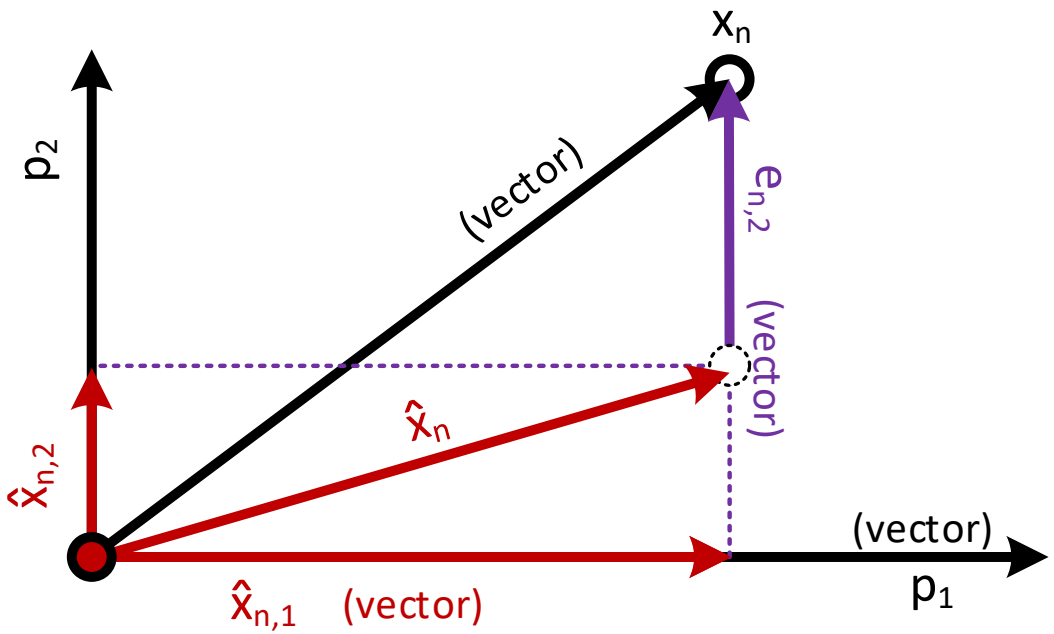
Overall Prediction of $\hat{X}$

- At the end of the day, if we have A components, the model predicts $\hat{X}$ as:

$$\hat{X} = TP^T$$

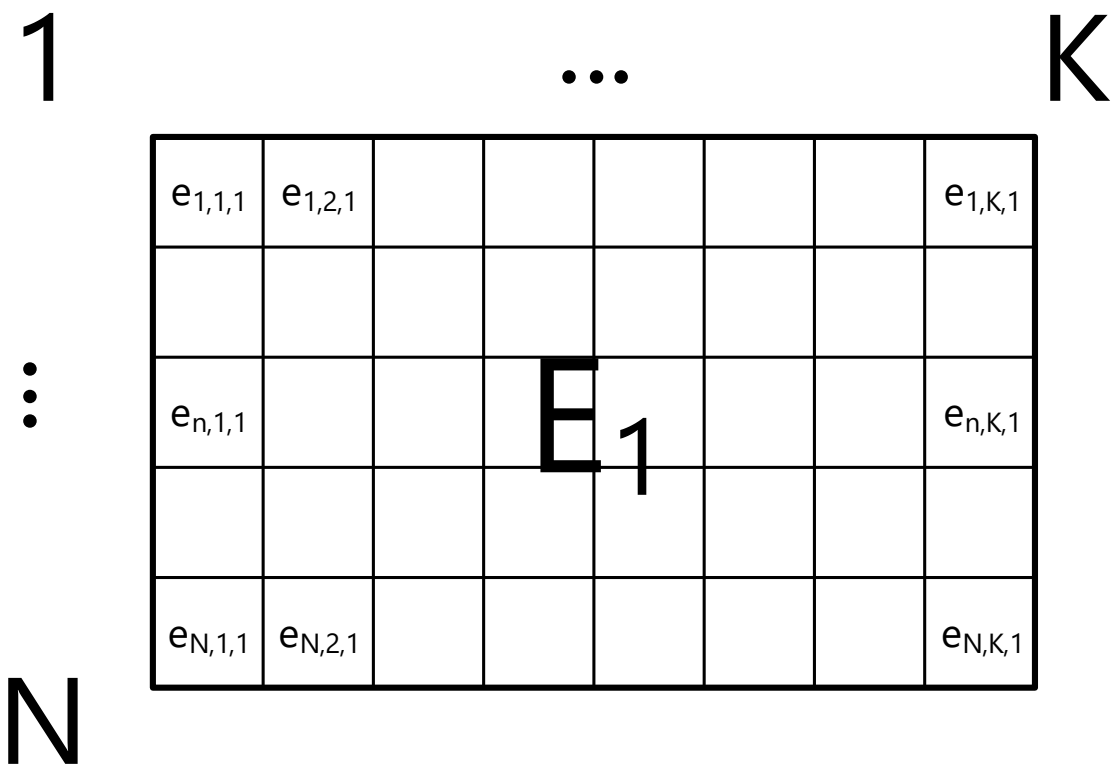
MUCH easier than computing each individual $\hat{x}$ and adding them up (math is the same)

$$(N \times K) = (N \times A) \times (A \times K)$$



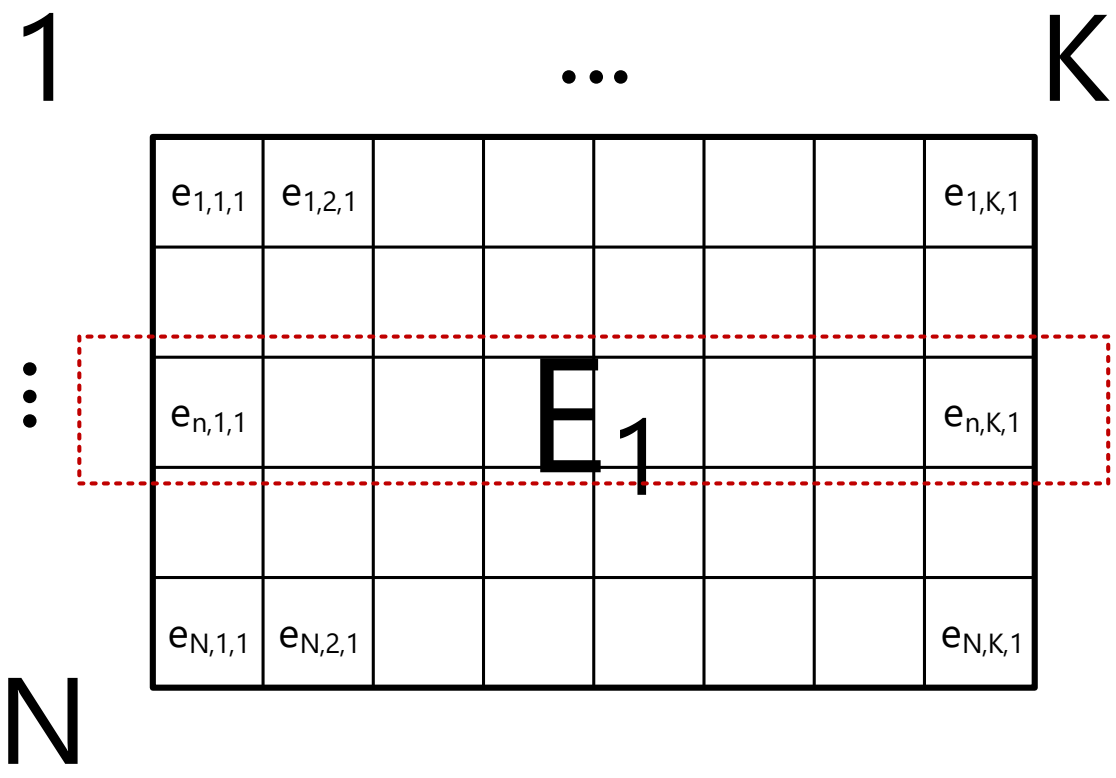
Quantifying Residuals

- Recall each residual $\boldsymbol{e}_{n,1}$ is a row from **X**
- We can assemble all residuals into a matrix that is the same size as **X** and represents the residual of our loading to each variable
  - For our first component, we'll call this $E \in \mathbb{R}^{N \times K}$



Quantifying Residuals

- Recall each residual $\boldsymbol{e}_{n,1}$ is a row from **X**
- We can assemble all residuals into a matrix that is the same size as **X** and represents the residual of our loading to each variable
  - For our first component, we'll call this $E \in \mathbb{R}^{N \times K}$



So What Do We Know?

- We know that we can compute $\hat{X}$ (all predicted measurements) using the convenient expression:

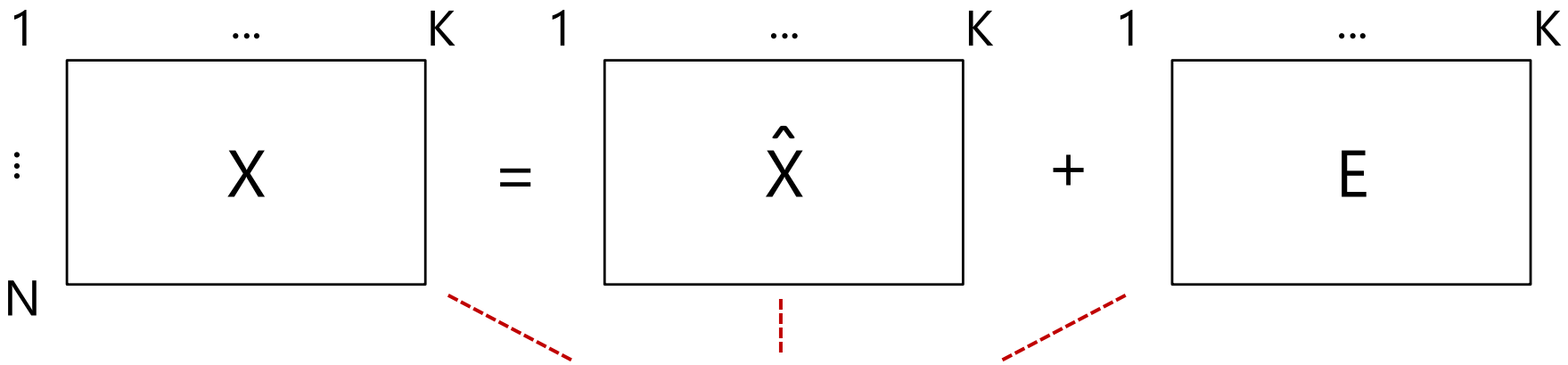
$$\hat{X} = TP^T$$

- We know that the TOTAL error of our model predictions $\hat{X}$ are the difference between X and the model prediction:

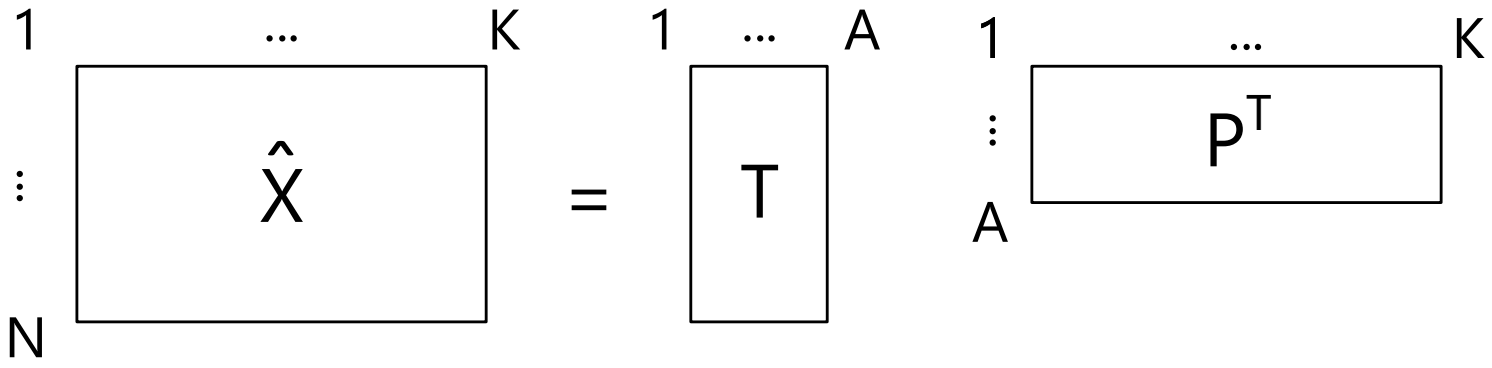
$$X = \hat{X} + E \quad \Rightarrow \quad X = TP^T + E \quad \Rightarrow \quad E = X - TP^T$$

- We also know that the more components (columns in T) we use, the close $\hat{X}$ becomes to X, and thus the smaller E becomes!

So What Do We Know?



It is important that these are all the same dimensions, else we could not add them!



$$(N \times K) = (N \times A) \times (A \times K)$$

Looking Ahead…

- So, if I were to ask you right now how we calculate the values in matrix P…
  - AKA the loadings, for component a: $\boldsymbol{p}_a$
- Where do you think this is going…?
  - Yup, you'd best believe that we want to minimize the variance in E. That is… minimize the **sum of squared-errors***
  - If we can find the values of P that make E as **small as possible** (for some known number of components A), we can say that the LV model as described by P optimally **explains the variance in** X
- In other words: perform a regression.

**MWAHAHAHAHAHAHAHAHAHAHAHA*

McMaster University ENGINEERING

Determining Model Accuracy

If the glove fits…

https://upload.wikimedia.org/wikipedia/commons/d/d3/NES-Power-Glove.jpg

Determining Model Fit

- You may ask why we are bothering to compute $\hat{X}$ since we already have the data for X
  - Good question… I have two answers:
    1. To BACK-CALCULATE values of X from a given or desired score T (reverse-engineering approach)
    2. To compute the **goodness of fit** of our model!

- Recall the metric R^2?
  - $R^2_{a=0} = 0$ (no variance explained)
  - R^2_a ALWAYS ↑ as a ↑ (less each time)
  - $R^2_{a=1} < R^2_{a=2} < \cdots < R^2_{a=A}$
  - Overall, $R^2 \leq 1$

R^2 of the model after a components are added

$$R^2_a = 1 - \frac{\mathcal{V}(X - \hat{X})}{\mathcal{V}(X)}$$

E_a are residuals after $\hat{X}$ is predicted using a components

$$R^2_a = 1 - \frac{\mathcal{V}(E_a)}{\mathcal{V}(X)}$$

IMPORTANT
$\mathcal{V}(X)$ is using the centered/scaled X
Then we can say $\bar{X} \equiv 0$

Workshop – Model Fit of Pastries

- Now that we know how to do it…
  - Use PCA function to determine loadings/scores for pastries
  - Compute R^2 using different numbers of components
  - Live demo in MATLAB

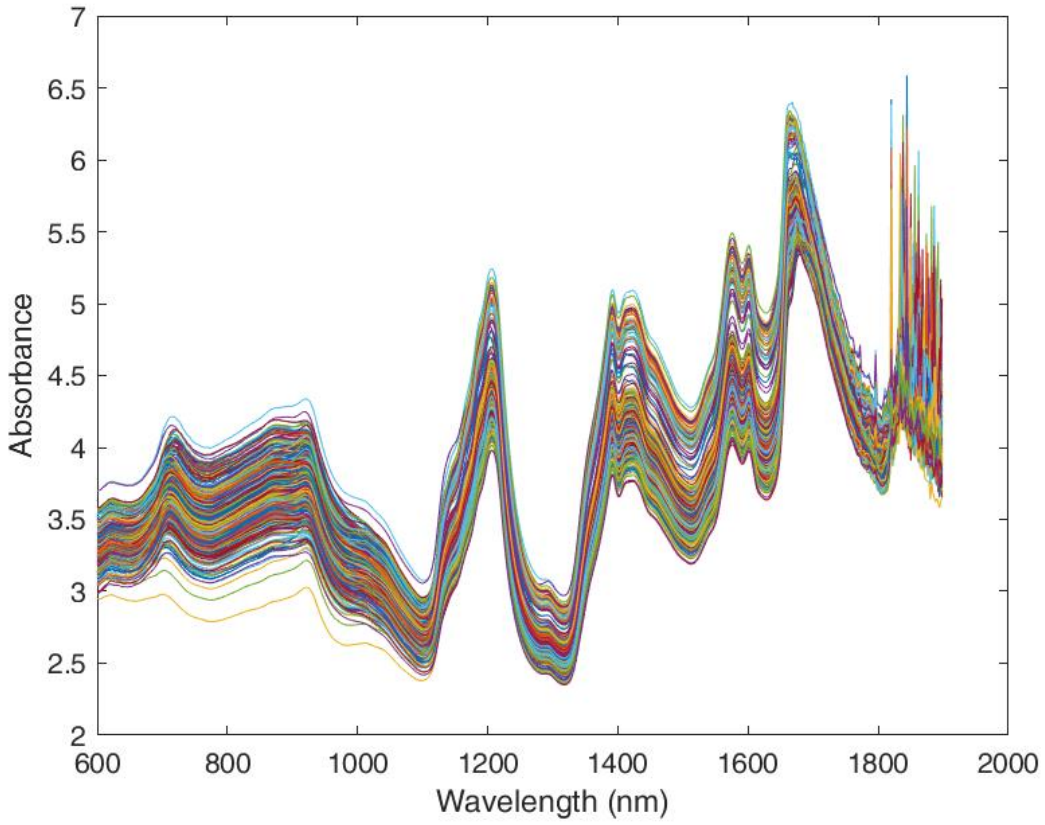
$$R_a^2 = 1 - \frac{\mathcal{V}\left(X - \hat{X}\right)}{\mathcal{V}(X)}$$

$$R_a^2 = 1 - \frac{\mathcal{V}(E_a)}{\mathcal{V}(X)}$$

Example: Spectral Data

- Recall our spectral data set (N = 460, K = 650)
  - It turns out we need ONLY THREE latent variables to describe a significant portion of the variance in the data

- $R^2_{a=1} = 0.737$
- $R^2_{a=2} = 0.922$
  - Adds 18.5%
- $R^2_{a=3} = 0.942$
  - Adds 2%



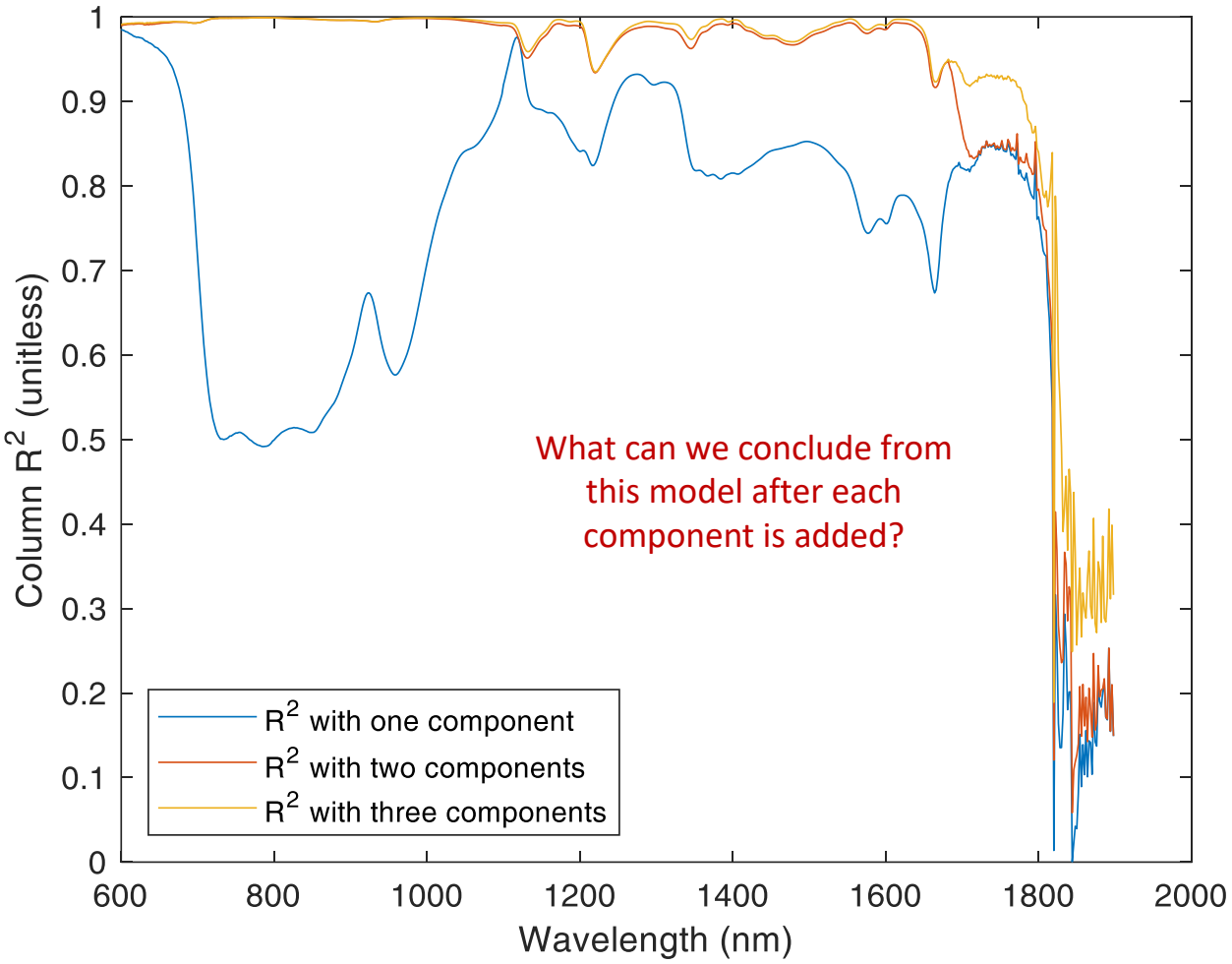
Column Residuals

- R^2 may be calculated for each **column** instead of the entire data set
  - Represents the predictive capability of our LVM space for one variable in particular

- $R_k^2 = 1 - \frac{\mathcal{V}(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k)}{\mathcal{V}(\boldsymbol{x}_k)}$
  - More components will always increase how well model explains the column
  - $R_k^2 = 0$ when there are no components (no shocker)
  - R_k^2 increases for every component added
  - **Question**: does it increase a lot if the new component has a loading $p_{k,a} \approx 0$?

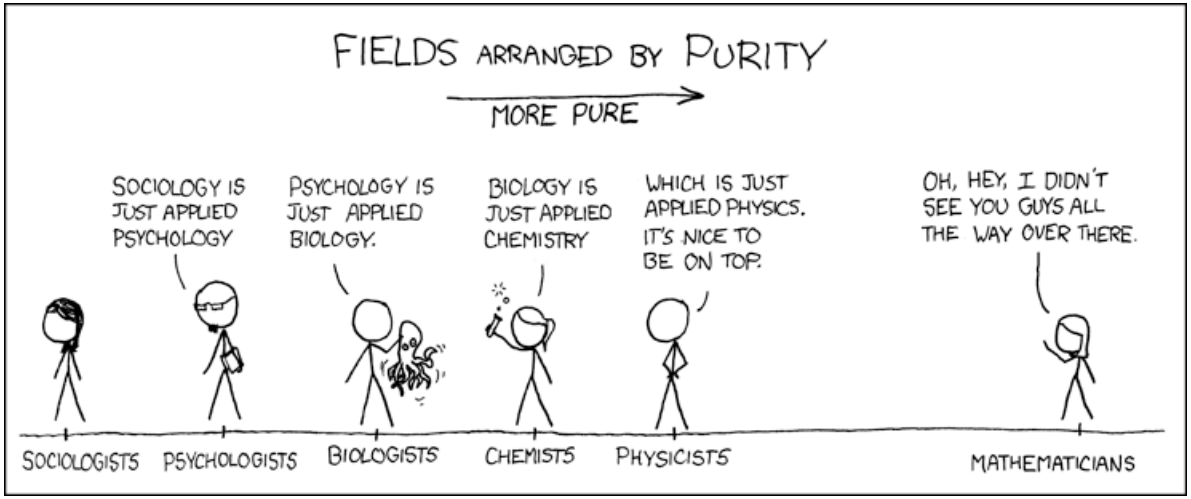
Column Residuals

- Example for spectral data in MATLAB



Derivation of PCA Components

It's math… IN (eigen)SPAAAAAAAAAACE!!



https://math.tufts.edu/faculty/equinto/students/purity.png

Optimization is Everywhere

- Similar to what we did in the regression section, we can set up an **optimization** to find $\boldsymbol{p}_a$
  - Let's just do this for the first component for now, so $\boldsymbol{p}_1$
- What does this optimization look like?

IN GENERAL:

$$\min_{???} \phi = f(???)$$

- For PCA, what are some possible…
  - Objective functions?
  - Constraints?

Eigenvalue Decomposition

- **FIRST**: WHY?
  - Because it gives us great insight as to what we are doing
  - Because it relates closely to regression
- **SECOND**: HOW?
  - On the board for now
  - Summary of critical results follow, but…
  - Will make no sense unless you can follow the math ☺
- Warning: I am going to make some hand-wavy claims
  - It is not the purpose of this class to get too detailed
  - If you have questions, I'm always game ☺

Eigenvalue Decomposition

$\max_{\boldsymbol{p}_1} \phi = \boldsymbol{t}_1^T \boldsymbol{t}_1$	$\max_{\boldsymbol{p}_1} \phi = (X \boldsymbol{p}_1)^T X \boldsymbol{p}_1$	$\max_{\boldsymbol{p}_1} \phi = \boldsymbol{p}_1^T X^T X \boldsymbol{p}_1$
(subject to)	(subject to)	(subject to)
$\boldsymbol{p}_1^T \boldsymbol{p}_1 = 1$ $\boldsymbol{t}_1 = X \boldsymbol{p}_1$	$\boldsymbol{p}_1^T \boldsymbol{p}_1 = 1$	$\boldsymbol{p}_1^T \boldsymbol{p}_1 = 1$

- This is **nearly** a convex optimization program
  - You'll notice that the max looks like a sum of squares
  - This is very similar to our regression
- The constraint really screws us up
  - However, the fact that it is an equality constraint gives us some hope… Maybe there is a way to reflect it in the objective?

Eigenvalue Decomposition

- Our strategy is to ELEVATE the constraint $\boldsymbol{p}_1^T \boldsymbol{p}_1 = 1$ to the objective and penalize a deviation from that unity
  - However, instead of choosing some arbitrary penalty factor, we will add a *decision variable* as the penalty factor
  - There are *reasons* for this, which we can discuss
  - The penalty factor is known as a **Lagrange Multiplier** $\lambda_1 \geq 0$

You might ask… Why not make $\lambda_1 = 0$? That would be equivalent to ignoring the constraint, which means p_1 can be infinite magnitude (no unique solution!

So if we deviate from $\boldsymbol{p}_1^T \boldsymbol{p}_1 = 1$, we *subtract* from the objective function

$$\max_{\boldsymbol{p}_1} \phi = \boldsymbol{p}_1^T X^T X \boldsymbol{p}_1 - \lambda_1 (\boldsymbol{p}_1^T \boldsymbol{p}_1 - 1)$$

(Unconstrained)

Eigenvalue Decomposition

- This problem is unconstrained, and allows us to identify a **stationary point** (like SSE for regression)
  - That is, we take the derivative of ϕ with respect to all decision variables and set them equal to 0
  - So, we take the derivative WRT all entries in $\boldsymbol{p}_1$ AND WRT λ_1
  - How many equations and unknowns will I have?

$$\frac{\partial \phi}{\partial \boldsymbol{p}_1} = 2X^T X\boldsymbol{p}_1 - 2\lambda_1 \boldsymbol{p}_1 = \mathbf{0}$$ ------- That's a $(K \times 1)$ vector of 0

$$\frac{\partial \phi}{\partial \lambda_1} = p_1^T \; p_1 - 1 = 0$$ ---------- That's a (1×1) scalar of 0

Eigenvalue Decomposition

- Look what happened thanks to λ_1
  - I am still obliged to choose $\boldsymbol{p}_1^T \boldsymbol{p}_1 = 1$
  - BUT the gradient of ϕ WRT $\boldsymbol{p}_1$ includes λ_1
  - Now, the only way to get $\frac{\partial \phi}{\partial \boldsymbol{p}_1} = 0$ is to choose $\boldsymbol{p}_1 = \mathbf{0}$ OR find a magical value of λ_1 and $\boldsymbol{p}_1$ that balances both pieces of $\frac{\partial \phi}{\partial \boldsymbol{p}_1}$
  - Note I can't even choose $\boldsymbol{p}_1 = \mathbf{0}$ because the second equation would be violated!

$$\frac{\partial \phi}{\partial \boldsymbol{p}_1} = 2X^T X \boldsymbol{p}_1 - 2\lambda_1 \boldsymbol{p}_1 = \mathbf{0}$$ ------- That's a $(K \times 1)$ vector of 0

$$\frac{\partial \phi}{\partial \lambda_1} = p_1^T \; p_1 - 1 = 0$$ ---------- That's a (1×1) scalar of 0

Eigenvalue Decomposition

- Focusing only on the $\frac{\partial \phi}{\partial \boldsymbol{p}_1}$ term:

$$X^T X \boldsymbol{p}_1 - \lambda_1 \boldsymbol{p}_1 = \mathbf{0}$$ ------- I divided out the 2. Hurray!

- Factoring out $\boldsymbol{p}_1$ on the RHS of the matrices:

$$(X^T X - \lambda_1 I_K)\boldsymbol{p}_1 = \mathbf{0}$$ ------- Also note that $\boldsymbol{p}_1^T \boldsymbol{p}_1 = 1$

- Hang on just a hot minute…
  - Where did this magical I_K come from?
  - Does that expression look familiar?

Eigenvalue Decomposition

- CONCLUSION: The first component of X (p_1), that…
  - Has unit length
  - Describes maximum variance in X
  - Correspond to scores $\boldsymbol{t}_1$

- … Is EXACTLY equal to the first Eigenvector of $X^T X$
  - It has corresponding Eigenvalue λ_1 such that $X^T X\, \boldsymbol{p}_1 = \lambda_1 \boldsymbol{p}_1$
  - λ_1 is exactly the variance described by the first component

- Subsequent components are the remaining eigenvectors of $X^T X$
  - Provided $\boldsymbol{p}_i \cdot \boldsymbol{p}_j = \boldsymbol{0}\ \ \forall\{i, j\}$ (all components are orthogonal)
  - The eigenvectors λ_i for each component $\boldsymbol{p}_i$ are the variances in $X^T X$ in the direction $\boldsymbol{p}_i$

Eigenvalue Decomposition Summary

- For the first component, our optimization problem to choose the first loadings vector $\boldsymbol{p}_1$ is:

These are the same…

$$\max_{\boldsymbol{p}_1} \phi = \boldsymbol{t}_1^T \boldsymbol{t}_1 \qquad OR \qquad \min_{\boldsymbol{p}_1} \phi = \sum_{i=1}^{N} e_{i,1}^2$$

(subject to)

$$\boldsymbol{p}_1^T \boldsymbol{p}_1 = 1$$ ------ Recall that we are forcing p_1 to have unit length

This is how we relate the decisions ($\boldsymbol{p}_1$) to the objective ------ $\boldsymbol{t}_1 = X\boldsymbol{p}_1$

- ALSO note that $\boldsymbol{p}_1$ has K values ($\boldsymbol{p}_1 \in \mathbb{R}^{K\times 1}$)

Eigenvalue Decomposition Summary

- Using this optimization problem and extending to multiple components (we'll skip for components 2+):
  - Loadings $\boldsymbol{p}_a$ are the **Eigenvectors** of $X^T X$
  - Scores $\boldsymbol{t}_a = X\boldsymbol{p}_a$ *OR* $T = XP$
  - Use scores to compute $\hat{X}_A = TP^T$
  - Determine residuals $E_A = X - \hat{X}_A$
  - Finally, can compute model fit as $R^2 = 1 - \frac{\mathcal{V}(E_A)}{\mathcal{V}(X)}$

- Fun facts about this decomposition…
  - Eigenvalues λ_a are the variances of the scores $\boldsymbol{t}_a$
  - Sum of all eigenvalues of $X^T X$ is trace(X) is $\mathcal{V}(X^T X)$

Workshop

- So we know we can calculate the loadings as the Eigenvectors of $X^T X$…
  - Explain why this works very well for “tall” matrices where $N \gg K$
  - Explain what might happen if I have a large number of columns such as the spectral data set… $N \ll K$

Eigenvalue Decomposition Remarks

- If you have a "short" matrix X, you might want to consider avoiding $X^T X \in \mathbb{R}^{K \times K}$
  - Instead, recognize that the Eigenvectors of $XX^T \in \mathbb{R}^{N \times N}$ are actually the scores of X, just not necessarily scaled correctly

- So our (more computationally efficient) strategy becomes:
  - Determine $\boldsymbol{t}_a$ as the Eigenvectors of XX^T
  - Determine "unscaled" loadings as $\boldsymbol{p}_a = X\boldsymbol{t}_a$
  - Normalize loadings as $\boldsymbol{p}_a = \frac{\boldsymbol{p}_a}{\|\boldsymbol{p}_a\|}$
  - Re-calculate scores as $\boldsymbol{t}_a = X\boldsymbol{p}_a$

Eigenvalue Decomposition Remarks

- There are several **disadvantages** of EV Decomposition to determine the loadings of a PCA model
  - We calculate all K Eigenvectors when we usually want $A \ll K$
  - Any missing data completely breaks this method
    - WORKSHOP: how would you handle missing data?

- Moreover, we have problems with **large matrices**
  - Calculating $X^T X$ can be cumbersome
  - Large data sets can cause overflow/loss of memory
  - We have to keep X in memory to compute scores

Eigenvalue Decomposition Remarks

- But there are distinct **advantages** too
  - EV decomposition gives us terrific insight into what a PCA model means and what we are doing to extract that info
    - An amazing application of an abstract math concept
  - All properties of PCA are derived from this method
  - Are slightly more accurate than convergence-based methods such as NIPALS
    - Only source of error is numerical precision
    - Error is “spread out” over all components
    - In NIPALS, the algorithm becomes less accurate the more components that are added

Final Remarks

- OK! So we have seen some cool stuff here
  - Geometric interpretation of model residuals
  - Mathematical interpretation of model residuals
  - Eigenvalue decomposition derivation
  - Connection of EV decomposition to PCA statistics

- Where are we going next?
  - NIPALS algorithm, which…
    - Can be much more efficient than EV decomposition
    - Can handle missing data
  - PLS
    - Extension of PCA to also predict outputs
    - NIPALS will be coded by YOU for this one ☺