

Scientific Computation

Spring, 2019

Lecture 12

Overview of time-marching methods

- Discussed: implicit/explicit, fixed time-step and variable time-step methods
 - Fixed time-step methods:
 - Accuracy characterized by truncation error
 - Also considered stability (model equation, $dy/dt = ay$, $\text{Real}(a) \leq 0$)
- Explicit Euler: truncation error $\sim \Delta t$, *unconditionally unstable* for model eqn.
- RK4: truncation error $\sim \Delta t^4$, *conditionally stable* for model eqn.
- Implicit Euler: truncation error $\sim \Delta t$, *unconditionally stable* for model eqn.

Explicit Euler: truncation error $\sim \Delta t$, *conditionally stable* for model eqn.
Unconditionally unstable if $\text{Real}(a)=0$

Maximum growth

- We need to be more careful with our problem statement:

Given a matrix A , find x so that $|b|$ is maximized with $|x|=1$

1. We first set an upper bound for the growth
2. Then, we see if we can reach that upper bound

Maximum growth

- We need to be more careful with our problem statement:

Given a matrix A , find x so that $|b|$ is maximized with $|x|=1$

1. We first set an upper bound for the growth
2. Then, we see if we can reach that upper bound

Task 1:

We have: $Ax = b$ which implies, $x^T A^T A x = b^T b$

Note that $A^T A$ is symmetric and thus can be orthogonally diagonalized:

$$A^T A = V S V^T$$

Here, V is the orthogonal eigenvector matrix and S is a diagonal matrix with the eigenvalues of $A^T A$ on the diagonal.

Maximum growth

Task 1:

We have: $Ax = b$ which implies, $x^T A^T Ax = b^T b$ (1)

Note that $A^T A$ is symmetric and thus can be orthogonally diagonalized:

$$A^T A = V S V^T$$

Here, V is the orthogonal eigenvector matrix and S is a diagonal matrix with the eigenvalues of $A^T A$ on the diagonal.

- Note that the eigenvalues of $A^T A$ are real and non-negative, and we construct S so that the eigenvalues appear in descending order,

$$S_{i,i} = \lambda_i, \quad \lambda_1 \geq \lambda_2 \dots \geq \lambda_N$$

- So (1) can be re-written as: $x^T V S V^T x = b^T b$
- Now, if we take $x = V z$, we have: $z^T S z = b^T b$

Maximum growth

Task 1:

- So (1) can be re-written as: $x^T V S V^T x = b^T b$
- Now, if we take $x = V z$, we have: $z^T S z = b^T b$ which can be rearrange as: $b^T b = \lambda_1 z_1^T z_1 + \lambda_2 z_2^T z_2 + \dots + \lambda_N z_N^T z_N$ and z_i is the i^{th} element of z

Then: $b^T b \leq \lambda_1 (z_1^T z_1 + z_2^T z_2 + \dots + z_N^T z_N)$

$$b^T b \leq \lambda_1 z^T z$$

$$x^T x = z^T z$$

$$b^T b \leq \lambda_1 x^T x$$

Task 2: Find x such that $x^T x = 1$ and $b^T b = \lambda_1$

Maximum growth

Task 2: Find x such that $x^T x = 1$ and $b^T b = \lambda_1$

This is easier!

We know $A^T A v_1 = \lambda_1 v_1$

and $v^T v = 1$, so if $x = v_1$, $|Ax| = |Av_1| = v_1^T \lambda_1 v_1 = \lambda_1$

The ‘most dangerous’ x is the “first” eigenvector of $A^T A$

Computational cost

- Eigenvalues and eigenvectors of $F = A^T A$ are needed
- Implementations typically use variations of QR algorithm
 - Basic idea: iteratively obtain a matrix *similar* to F which is nearly triangular
 - $F_k = Q^{-1} F Q$ after k iterations with approximations of the eigenvalues of F_k on its diagonal
 - Cost is generally estimated to be $O(N^3)$, though this depends on rate of convergence of iterations
 - Can we do better?

Singular value decomposition

- The eigenvalues and eigenvectors of F can be found from the singular value decomposition (SVD) of A :

$$A = U\Sigma W^T$$

- Recall that $F = A^T A = V S V^T$
- From the SVD: $A^T A = W \Sigma^2 W^T$ (U and W are orthogonal – as they are left and right eigenvectors of F which is symmetric)
- Comparing: $\Sigma^2 = S$

$$W = V$$

- The key point here is that variants of QR have been developed specifically to compute the SVD (avoids explicitly constructing $A^T A$)
- Asymptotic cost is similar to QR: $O(M^2N) + O(N^3)$ for A with size $M \times N$

QR vs SVD

- Approximate costs are similar, what do we see in practice?

```
In [9]: A = np.random.randn(800,800)
```

```
In [10]: A.shape
```

```
Out[10]: (800, 800)
```

```
In [11]: C = np.dot(A.T,A)
```

```
In [12]: timeit l,v = np.linalg.eig(C)
```

```
543 ms ± 13.4 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
In [13]: timeit u,svh = np.linalg.svd(A)
```

```
300 ms ± 1.62 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

QR vs SVD

- Approximate costs are similar, what do we see in practice?

```
In [9]: A = np.random.randn(800,800)
```

```
In [10]: A.shape
```

```
Out[10]: (800, 800)
```

```
In [11]: C = np.dot(A.T,A)
```

```
In [12]: timeit l,v = np.linalg.eig(C)
```

```
543 ms ± 13.4 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
In [13]: timeit u,svh = np.linalg.svd(A)
```

```
300 ms ± 1.62 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
In [22]: l[0]
```

```
Out[22]: 3195.472688027676
```

```
In [23]: s[0]**2
```

```
Out[23]: 3195.4726880276557
```

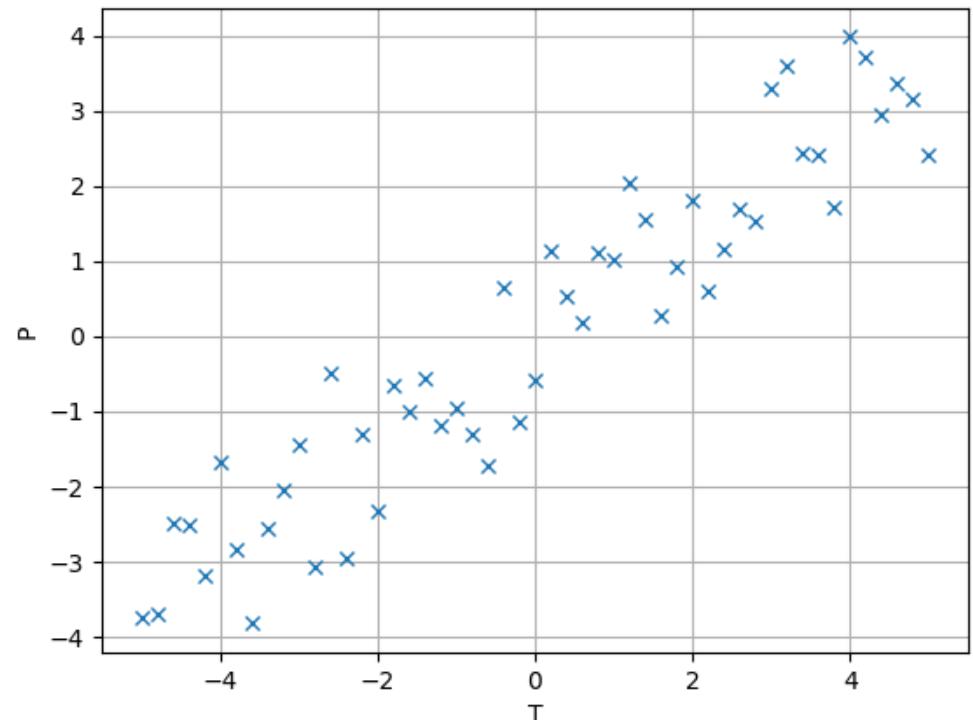
Note: Should use `scipy.sparse.linalg` when working with large sparse matrices

Data science

- Step 1: Data collection/acquisition
 - Lab measurements
 - Online databases (e.g. Kaggle)
 - Simulation results
- Typically have data corresponding to multiple variables
 - Temperature, pressure, humidity, wind speed and direction
 - IP address, location, duration of visit, number of visits, pages visited
- But often don't know in advance if all of these variables are “important” – how many variables do we actually need to capture the essential trends or dynamics?

Data analysis

- Example: Consider the data in the figure corresponding to measurements of temperature and pressure
 - Before looking at the data, we might conjecture that the underlying dynamics are connected to a general function of these two variables, $f(T,P)=c$ with c some constant
 - But do we really need two independent variables?



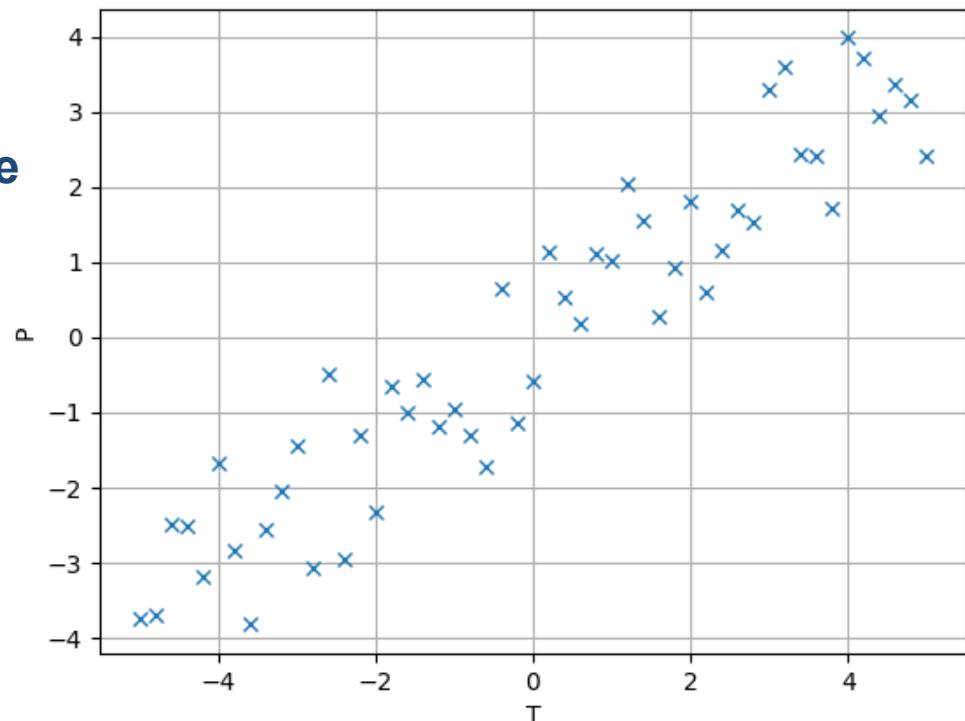
Data analysis

- Example: Consider the data in the figure corresponding to measurements of temperature and pressure
 - Before looking at the data, we might conjecture that the underlying dynamics are connected to a general function of these two variables, $f(T,P)=c$ with c some constant
 - But do we really need two independent variables?

Basic ideas:

Interesting/important dynamics are large-variance dynamics

The less correlation between independent variables, the better



Data analysis

- Example 2: Say you are tracking a meteor heading towards Earth

- How many variables do you need?
- Three position coordinates? $x(t)$, $y(t)$, $z(t)$
- Maybe three velocity components as well? $u(t)$, $v(t)$, $w(t)$

Sweet Meteor O'Death
@smod4real

Perennial Candidate for President, Precambrian Conservative, Tough on Putin & Iran, Everyone's going to die, #SMOD2020

Tweets 977 Following 1,050 Followers 143K Likes 19K Lists 7 Moments 1

Tweets Tweets & replies Media

Sweet Meteor O'Death @smod4real · Feb 12
Farewell to the extraterrestrial robot who lived twice as long as the average American marriage

Jacob Margolis @JacobMargolis · Feb 12
Sad news. Mars rover #Opportunity is probably done. Sometime tonight, a team @NASAJPL will make their final attempt to contact #Oppty. If they can't, they'll likely call the mission. Here's what happened... 1/
Show this thread

But what if the meteor is just moving in a circular orbit?

Let's translate ideas of extracting “important” variables into the language of linear algebra

PCA

- We measure the meteor coordinates at: N times and store them in a 3 x N array:

- Data: $A = \begin{bmatrix} x_1 & x_2 & \dots & x_N \\ y_1 & y_2 & \dots & y_N \\ z_1 & z_2 & \dots & z_N \end{bmatrix}$

- x, y, z are each N-element column vectors, so: $A = \begin{bmatrix} \mathbf{x}^T \\ \mathbf{y}^T \\ \mathbf{z}^T \end{bmatrix}$

We assume that each has been scaled to have zero mean, e.g. $\sum_{i=1}^N x_i = 0$

Then, $C = AA^T$ is the covariance matrix (technically, it should be scaled with N)

The diagonal elements are the variances of x, y, and z. The off-diagonal elements

contain covariances such as: $\mathbf{x}^T \mathbf{z} = \sum_{i=1}^N x_i z_i$

PCA

- **Method:** Project data onto new variables so covariance matrix is diagonalized
 - With zero covariance – the new variables will be “independent”
 - The variable with the larger variance is more “important”
- Let $F A = G$ -- need to find a “projection matrix”, F , so that $GG^T=D$ is diagonal
- **A contains projection weights**
 - **The first row transforms** $x \rightarrow \tilde{x} : \alpha_1 x + \alpha_2 y + \alpha_3 z = \tilde{x}$
 - **And the full projection matrix is:** $F = \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 \\ \beta_1 & \beta_2 & \beta_3 \\ \gamma_1 & \gamma_2 & \gamma_3 \end{bmatrix}$
- **We need** $D = F A A^T F^T = F C F^T$
- **Note:** if A has zero-mean, so does G

PCA

- We need $D = F A A^T F^T = F C F^T$
- The covariance matrix is symmetric and orthogonally diagonalizable...

$$A A^T = V S V^T$$

- And by inspection, we see that $F = V^T$ gives $D = S$

Notes:

1. V and S contain the eigenvectors and eigenvalues of $A A^T$
2. Eigenvalues of $A A^T$ and $A^T A$ are the same (but one may have more zeros)
3. Here, V corresponds to U in the SVD of A : $A = U \Sigma W^T$
4. Our transformed data is $G = V^T A$ and the eigenvalues of S indicate the “importance” of each row of G
5. Since F is orthogonal, G and A have the same variance ($G G^T$ and $A A^T$ have the same trace)

Data analysis

- Python implementation (2 variables):
 1. Set up, solve eigenvalue problem:

Data analysis

- Python implementation:

1. Set up, solve eigenvalue problem:

```
In [45]: A.shape
```

```
Out[45]: (2, 51)
```

```
In [46]: A.mean(axis=1)
```

```
Out[46]: array([-6.26949473e-16,  
5.22457894e-17])
```

```
In [47]: U,S,VT = svd(A)
```

2. Compute transformed data:

```
In [48]: F = U.T
```

```
In [49]: G = np.dot(F,A)
```

```
In [50]: np.dot(G,G.T)
```

```
Out[50]:  
array([[ 8.33602821e+02,  
-1.18785463e-13],  
[-1.18785463e-13,  
1.50729071e+01]])
```

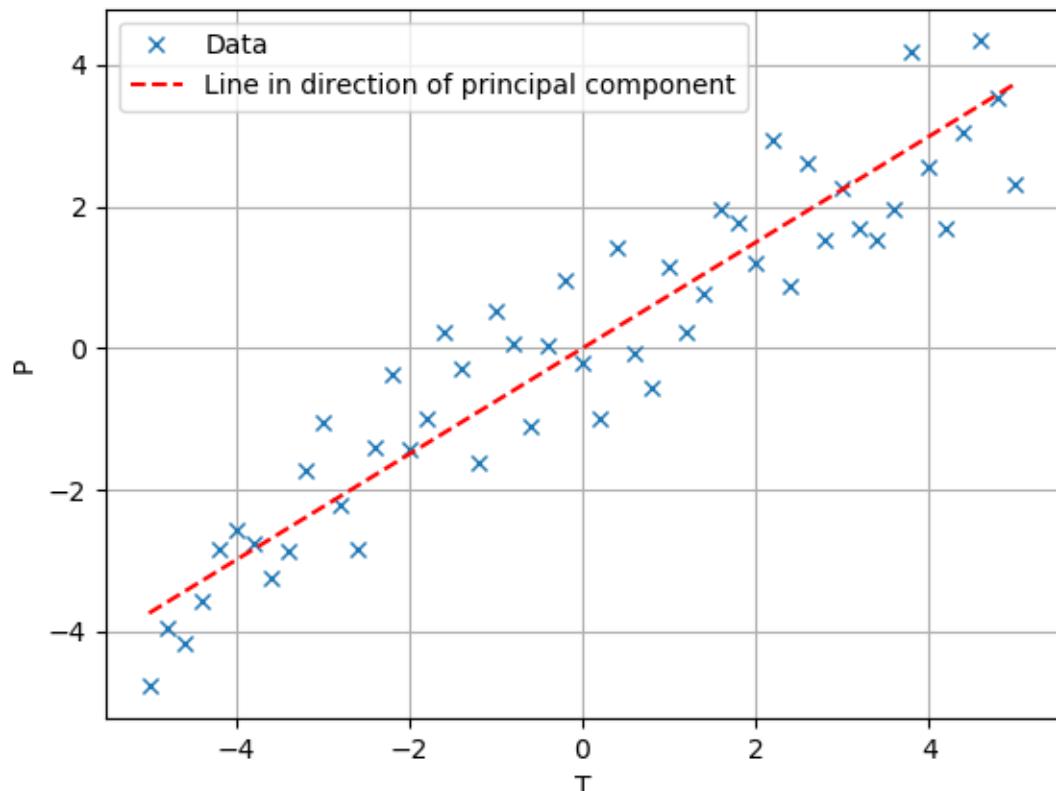
Data analysis

- Python implementation:

3. Compare first eigenvector (corresponding to largest eigenvalue) to data:

In [51]: `plot(t, U[1,0]*t/U[0,0]-5, 'r--')`

- The eigenvectors of C are the *principal components*
- The 1st component points in the direction of maximum variance
- What is the direction of the 2nd component?



Dimension reduction

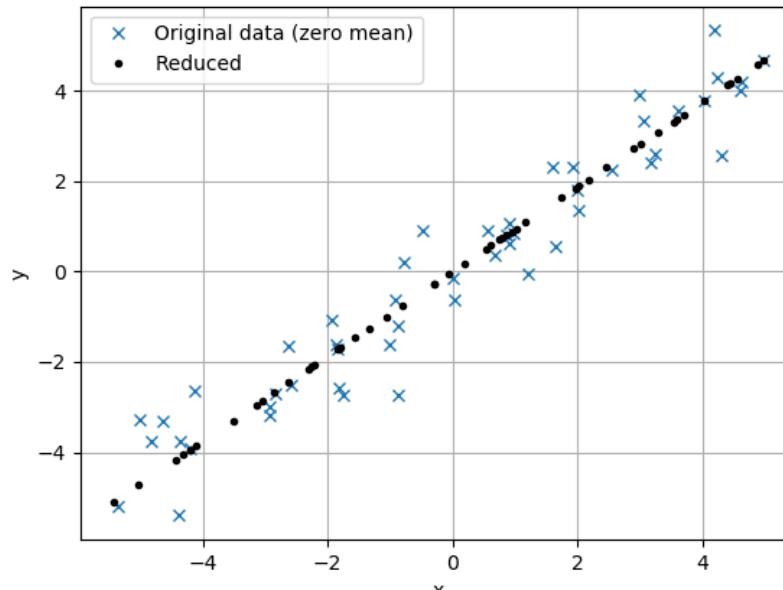
- The eigenvalues tell us how much of the variance is associated with the principal component: In [52]: S
Out [52]: array([28.87218075, 3.8823842])
- Dimension reduction: discard variables (row of G) associated with low variance
For this example, discard 2nd row of G and retain 1st row
- How do we express reduced data in original variables? Recall:
 $U^T A = G$ or $A = UG$

The **reduced** data, in our original variables, is then the outer product of the 1st eigenvector of U with the 1st row of G:

```
In [64]: A2=np.outer(U[:,0],G[0,:])
```

```
In [65]: A2.shape  
Out [65]: (2, 51)
```

```
In [66]: plot(A2[0,:],A2[1,:],'k.')
```



Dimension reduction

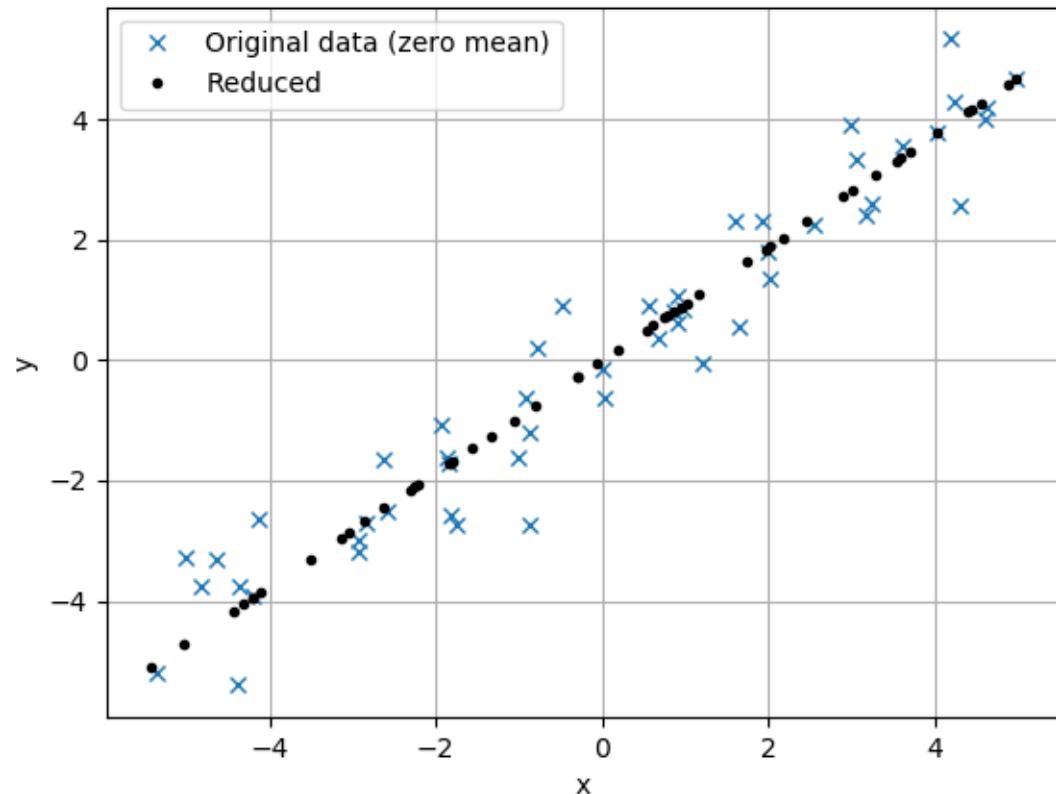
The eigenvalues tell us how much of the variance is associated with the principal component: In [52]: S

Out [52]: array([28.87218075, 3.8823842])

Dimension reduction: discard variables (columns of G) associated with low variance

For this example, discard 2nd row of G and retain 1st row

- Can reduce redundancy and simplify models for underlying dynamics
- This example with just two variables is not a practical application
- Many problems have 10s, 100s, or even 1000s of variables



Notes

- The presented procedure for PCA then is:
 - 1) Arrange data into a matrix (subtracting means if needed)
 - 2) Compute eigenvalues and eigenvectors of covariance matrix
 - 3) Order eigenvalues and corresponding eigenvectors (if needed)
 - 4) “Focus” on the components corresponding to the largest eigenvalues
- What is the cost of PCA?