Next: Naive QR algorithm

- Some variant QR algorithm is used in many commercial-grade eigenvalue computation routines.
- Algorithm: two part iteration:
  - QR decomposition: $A_n \rightarrow Q_n R_n$
  - Create next matrix: $A_{n+1} = R_n Q_n$
- Similar to simultaneous iteration because you re-orthogonalize at each step.
- Note difference between "QR decomposition" (matrix decomposition) and "QR algorithm" (finding eigenvalues)

Naive QR algorithm

1. Input: symmetric matrix A
2. Loop:
3. Compute $[Q, R] = \mathrm{qr}(A_n)$
4. Update $A_{n+1} = R*Q$
5. Check for convergence: $\mathrm{norm}(A_{n+1} - A_n) < \mathrm{tol}$?

   If yes, eigenvalues lie on diagonal of A_{n+1}.

   If no, continue looping.

To get real eigenvalues for demo purposes. Algorithm will also work on any square matrix after modifications.

Demonstration

```
>> A

A =

   -0.6412    1.2549   -3.0582    1.1469
    1.2549   -2.1334    1.1162    0.4487
   -3.0582    1.1162   -3.1301   -0.0432
    1.1469    0.4487   -0.0432   -1.4683

>> my_eig_qr(A)

ans =

   -5.9322
   -2.1390
    1.8135
   -1.1153

>> eigs(A)

ans =

   -5.9322
   -2.1390
    1.8135
   -1.1153
```

Start with symmetric matrix for real eigenvalues

Uses QR iteration to find all eigenvalues

Matlab built-in

Remarks on QR algorithm

- Magical!
- Works for all square matrices
- Convergence is slow.
  - Can be improved using shifting (next)
- Dealing with complex eigenvalues requires special effort.
- Real implementations usually start by reducing input matrix to Hessenberg form.

Triangular with additional set of non-zeros on first off-diag.

Why does QR work?

- Start with matrix A_0
- Consider the iteration:

$$\begin{tikzcd}
n=0 & A_0 \arrow[r] & Q_1 \arrow[dr] & R_1 \arrow[dl] & & & \\
n=1 & & R_1 & Q_1 \arrow[r] & A_1 \arrow[r] & Q_2 \arrow[dll] & R_2 \arrow[dllll] \\
n=2 & & R_2 & Q_2 \arrow[r] & A_2 \arrow[r] & Q_3 \arrow[dll] & R_3 \arrow[dllll] \\
n=3 & & R_3 & Q_3 \arrow[r] & A_3 \arrow[r] & Q_4 & R_4 \\
 & & etc.... & & & &
\end{tikzcd}$$

This is the calc we have performed written out in steps.

Powers of A

- Define

$$\hat{Q}_k = Q_1 Q_2 Q_3 \cdots Q_k \qquad \hat{R}_k = R_k R_{k-1} \cdots R_1$$

- Consider 3rd power of A_0:

$$\begin{tikzcd}[column sep=small]
A_0^3=(Q_1 \arrow[d] & R_1)(Q_1 \arrow[d, dash] & & R_1)(Q_1 \arrow[d, dash] & R_1) \arrow[d] \\
Q_1 \arrow[d] & Q_2 R_2 \arrow[d] \arrow[dr, dash] & & Q_2 R_2 \arrow[dl, dash] \arrow[d] & R_1 \arrow[d] \\
Q_1 & Q_2 & Q_3 R_3 & R_2 & R_1
\end{tikzcd}$$

$$\begin{tikzcd}
A_0 \arrow[r] & Q_1 R_1 \arrow[dl] \\
A_1 = R_1 Q_1 \arrow[r] & Q_2 R_2 \arrow[dl] \\
A_2 = R_2 Q_2 \arrow[r] & Q_3 R_3 \arrow[dl] \\
A_3 = R_3 Q_3 \arrow[r] & Q_4 R_4
\end{tikzcd}$$

For reference – from last slide

- Therefore:

$$A_0^3 = Q_1 Q_2 Q_3 R_3 R_2 R_1 = \hat{Q}_3 \hat{R}_3$$

QR decomposition of (input *A*)^3

Iterates of A

- Consider 3rd iterate of A_0:

$$A_3 = R_3 Q_3$$

$$A_2 = Q_3 R_3 \quad \Rightarrow \quad Q_3^T A_2 = R_3$$

- Therefore, $A_3 = Q_3^T A_2 Q_3$
- Similarly, $A_3 = Q_3^T Q_2^T A_1 Q_2 Q_3$

$$A_3 = Q_3^T Q_2^T Q_1^T A_0 Q_1 Q_2 Q_3$$

$$A_3 = \hat{Q}_3{}^T A_0 \hat{Q}_3$$

Every time I iterate, I get back a new matrix with the same eigenvalues as the old one.

Similarity transform of A_0 – preserves eigenvalues

- Or, $A_0 = \hat{Q}_3 A_3 \hat{Q}_3{}^T$

$$\begin{tikzcd}
A_0 \arrow[r] & Q_1 R_1 \arrow[dl] \\
A_1 = R_1 Q_1 \arrow[r] & Q_2 R_2 \arrow[dl] \\
A_2 = R_2 Q_2 \arrow[r] & Q_3 R_3 \arrow[dl] \\
A_3 = R_3 Q_3 \arrow[r] & Q_4 R_4
\end{tikzcd}$$

For reference – from last slide

Interpretation

$$A_0^n = \hat{Q}_n \hat{R}_n$$

Statement about powers of A

$$A_0 = \hat{Q}_n A_n \hat{Q}_n^T$$

Statement about iterates of A

- $A_0^n = \hat{Q}_n \hat{R}_n$ says $\hat{Q}_n$ is an orthonormal basis for A_0^n .
- Recall simultaneous iteration. In the limit, the cols of $\hat{Q}_n$ converge to the eigenvectors of A_0^n .

Similar to power iteration

- $A_0 = \hat{Q}_n A_n \hat{Q}_n^T$ says A_0 and A_n have same eigenvalues

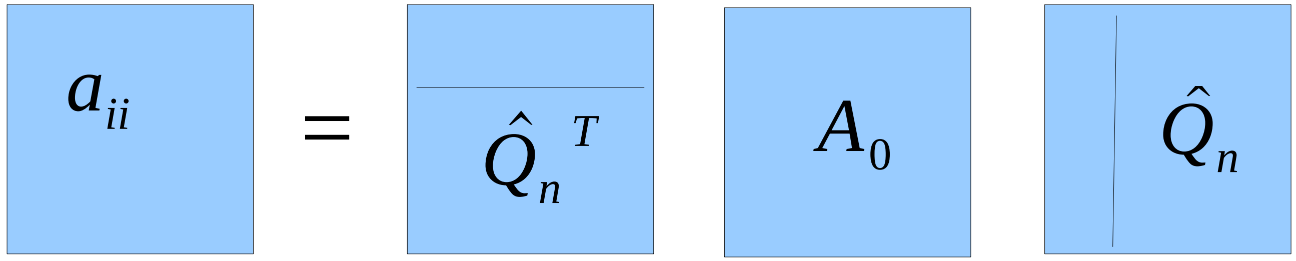
Because this is a similarity transformation

- How do we know the eigenvalues are on the main diagonal?

- Consider focusing on one column of $\hat{Q}_n$

$$A_n = \hat{Q}_n^T A_0 \hat{Q}_n$$

Columns are eigenvectors or A_0



Diagonals

$$a_{ii} = \hat{q}_i^T A_0 \hat{q}_i$$

q is eigenvector

$$a_{ii} = \hat{q}_i^T \lambda_i \hat{q}_i$$

$$a_{ii} = \lambda_i$$

Because q is unit vector (Q is orthonormal)

Off-diagonals

$$a_{ji} = \hat{q}_j^T A_0 \hat{q}_i$$

$$a_{ji} = \hat{q}_j^T \lambda_i \hat{q}_i$$

$$a_{ii} = 0$$

- Therefore, diagonals converge to eigenvalues of original matrix A_0, off diags go to zero.

Improving convergence

- Recall power method convergence:

$$A^n \vec{b}_0 = \lambda_1^n \left(\beta_1 \vec{q}_1 + \beta_2 \left(\frac{\lambda_2}{\lambda_1} \right)^n \vec{q}_2 + \beta_3 \left(\frac{\lambda_3}{\lambda_1} \right)^n \vec{q}_3 + \cdots + \beta_k \left(\frac{\lambda_k}{\lambda_1} \right)^n \vec{q}_k \right)$$

- Power method converges as: $(\lambda_2 / \lambda_1)^n$
- Recall simultaneous iteration converges as:

$$max(\lambda_{j+1}/\lambda_j)^n$$

- Analogously, QR algorithm converges as

$$max(\lambda_{j+1}/\lambda_j)^n \longleftarrow$$

Can converge slowly if eigenvalues are close in magnitude

Practical QR algorithm

- Improve convergence: Use “shifting” to make smallest eigenvalue converge extra fast.
- Also use “deflation” to shrink matrix as eigenvalues are found.
- Finally, most QR implementations first reduce input matrix to upper Hessenberg or other convenient form.

$$\begin{pmatrix} x & x & x & x & x & x & x \\ x & x & x & x & x & x & x \\ 0 & x & x & x & x & x & x \\ 0 & 0 & x & x & x & x & x \\ 0 & 0 & 0 & x & x & x & x \\ 0 & 0 & 0 & 0 & x & x & x \\ 0 & 0 & 0 & 0 & 0 & x & x \end{pmatrix}$$

Hessenberg = Upper triangular + first lower subdiagonal

Shifting

- Naive QR:

$$[Q,R] \leftarrow qr\left(A^{(n)}\right)$$

$$A^{(n+1)} \leftarrow RQ$$

- QR with shifting:

“Tail” of A – Move this element to zero to speed convergence of next eigenvalue

$$\sigma \leftarrow A^{(n)}_{end,end}$$

$$[Q,R] \leftarrow qr\left(A^{(n)} - \sigma I\right)$$

Take away smallest value before doing qr

$$A^{(n+1)} \leftarrow RQ + \sigma I$$

Then put it back to generate next A matrix.

Theorem

- If *u* = eigenvector & λ = eigenvalue of matrix A, we have:

$$Au=\lambda u$$

- Consider adding in scalar *p* on the digaonal:

$$(A+pI)u=\lambda u+pIu \longleftarrow Iu=u$$

$$(A+pI)u=(\lambda+p)u$$

- Therefore, eigenvalue of matrix (*A* + *p I*) is

$$\lambda+p$$

- This fact can be useful when dealing with singular matrices, etc.

Why shift?

- Each eigenvalue converges like $(\lambda_{j+1}/\lambda_j)^n$
- Idea: the shift at each step moves the end λ_{j+1}/λ_j close to zero, thereby accelerating convergence.

$$\sigma \leftarrow A^{(n)}_{end,end}$$

$$[Q,R] \leftarrow qr\left(A^{(n)} - \sigma I\right)$$

Move diagonal element on tail close to zero

$$A^{(n+1)} \leftarrow RQ + \sigma I$$

Recover original eigenvalues

- Then recover matrix A's eigenvalues before next iteration

Deflation

- Once the bottom right “tail” eigenvalue has converged, add it to a list of eigenvalues, then shrink matrix.

$$\overbrace{\begin{pmatrix} x & x & x & x & x & x & x \\ 0 & x & x & x & x & x & x \\ 0 & 0 & x & x & x & x & x \\ 0 & 0 & 0 & x & x & x & x \\ 0 & 0 & 0 & 0 & x & x & x \\ 0 & 0 & 0 & 0 & 0 & x & x \\ 0 & 0 & 0 & 0 & 0 & 0 & x \end{pmatrix}}^{7} \xrightarrow{\text{Shrink matrix (remove last col and row}} \overbrace{\begin{pmatrix} x & x & x & x & x & x \\ 0 & x & x & x & x & x \\ 0 & 0 & x & x & x & x \\ 0 & 0 & 0 & x & x & x \\ 0 & 0 & 0 & 0 & x & x \\ 0 & 0 & 0 & 0 & 0 & x \end{pmatrix}}^{6}$$

This value converged.

Deflation

- Example: run_myeig_qr_shifted

```
After iteration 3, W = 5, matrix =
   -2.0663   -1.5073   -0.1273    0.0272    0.0000
   -1.5073    4.6719    0.6218   -0.0402   -0.0000
   -0.1273    0.6218   -1.4940    0.6710   -0.0000
    0.0272   -0.0402    0.6710   -0.2151   -0.0000
    0.0000   -0.0000   -0.0000   -0.0000    1.6376

After iteration 4, W = 4, matrix =
    1.0400   -3.7089   -0.0730    0.0245
   -3.7089    1.6249    0.1008    0.0057
   -0.0730    0.1008   -1.1270    0.9171
    0.0245    0.0057    0.9171   -0.6413
```

Element converged

Matrix deflated at next iteration

QR algorithm convergence much faster with shift and deflate

- Example 5x5 matrix using naive QR:

```
--- We converged after 1356 iterations! ---
```

Extreme but real case

- Same matrix using QR with shift and deflate:

```
--- We converged after 11 iterations! ---
```

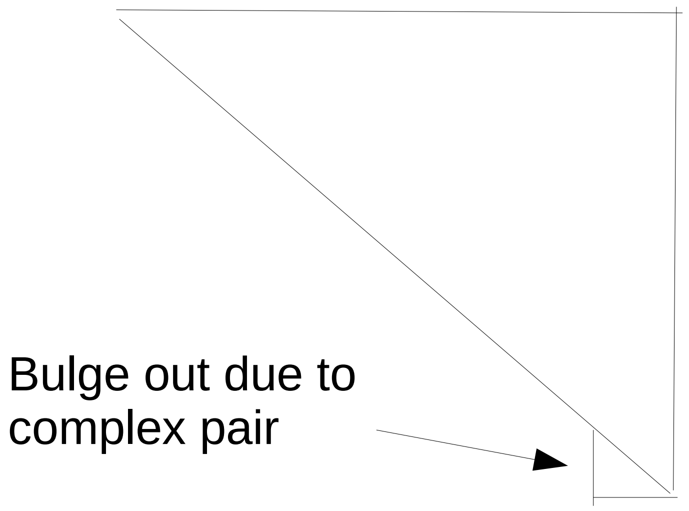
- Why so different?
  - Naive QR converges as $\left(\lambda_{j+1}/\lambda_j\right)^n$
  - If two eigenvalues are close to each other, convergence can be very slow.
  - Shifting fixes this problem.

Next problem: complex eigenvalues

- QR iteration on arbitrary square matrix gives:
  - Upper triangular
  - 2x2 blocks on main diagonal which correspond to complex eigenvalues.

```
Matrix after iteration 90 =
    1.0008    1.0151   -0.2210   -2.8003    0.0693   -0.3408
   -2.1101    1.8797   -0.2943    0.7947    1.5698   -1.5082
   -0.0000   -0.0000   -0.2707    1.6899    1.5577    0.5418
    0.0000    0.0000   -1.2961   -0.2603   -1.1709    0.9187
   -0.0000    0.0000    0.0000   -0.0000   -1.2889   -0.5015
   -0.0000   -0.0000    0.0000    0.0000   -0.0000    0.7922
```

Francis's double-shift algorithm



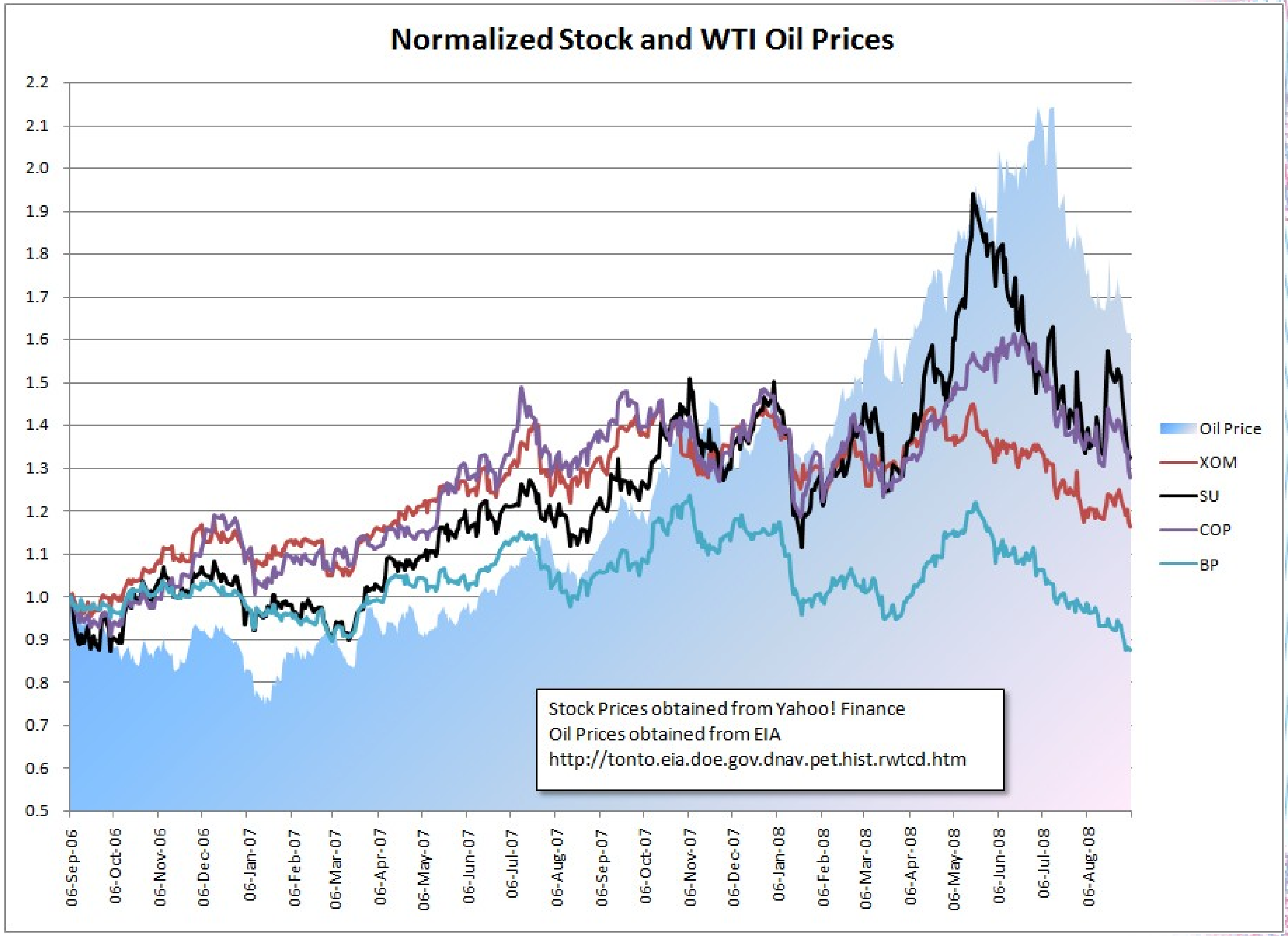
- Mini-project for somebody?

New topic: PCA
Principal Component Analysis

- Traditionally uses EVD.
  - Modern method uses SVD.
- Very useful data analysis technique.
- Looks for correlations between data sets, recommends way to convert correlated data sets into smaller set of uncorrelated data.
- Identifies “underlying dynamics”.
- Often first step in dimensionality reduction.

Example: Stock prices

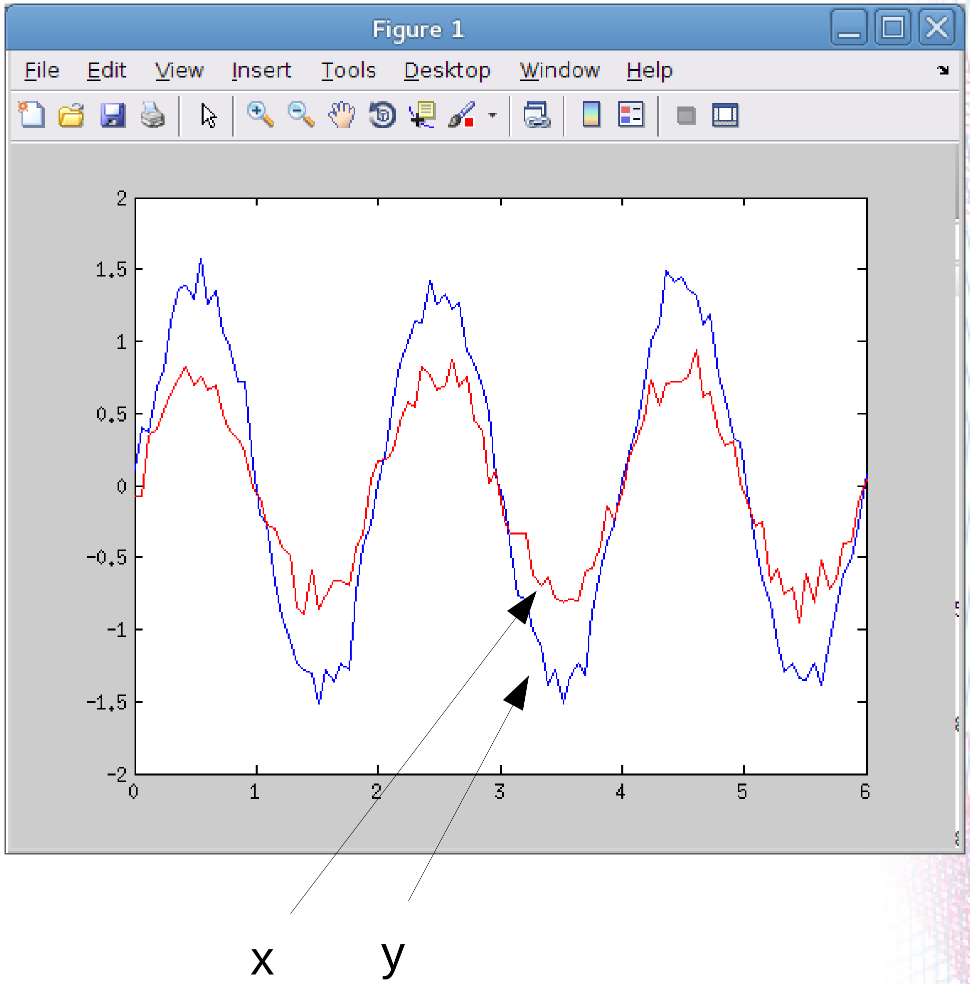
- Stock prices seem to move together, but also have some component of individual behavior.



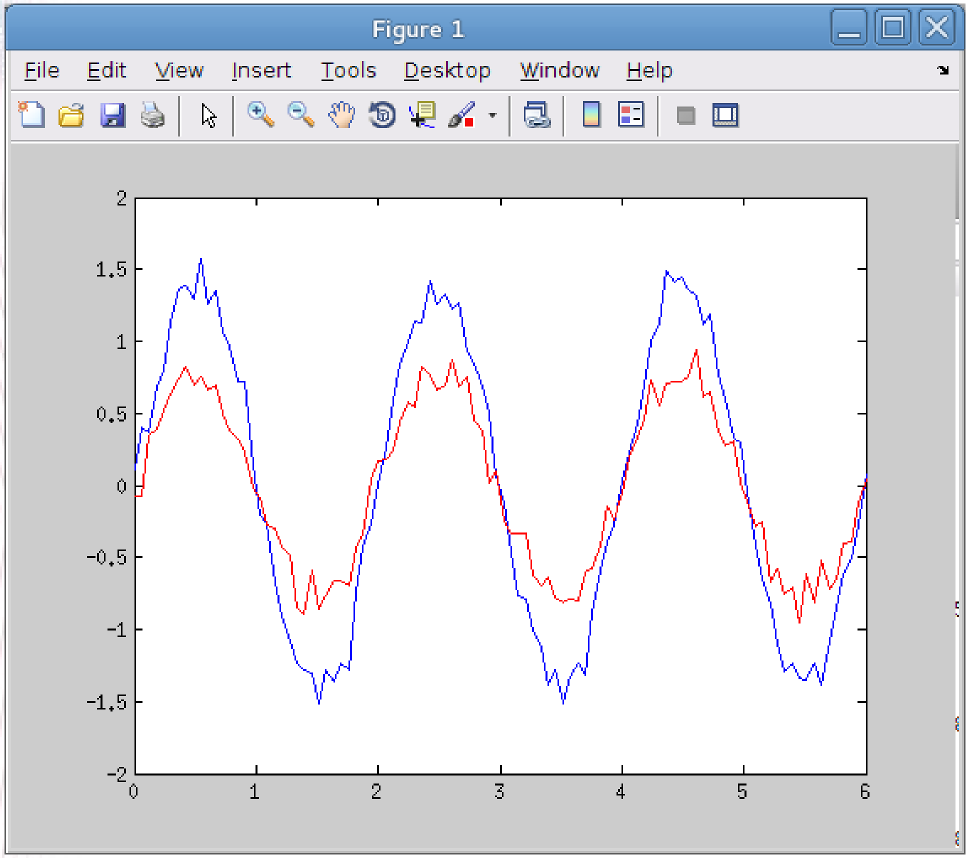
- Can we separate the common motions from the individual behaviors?

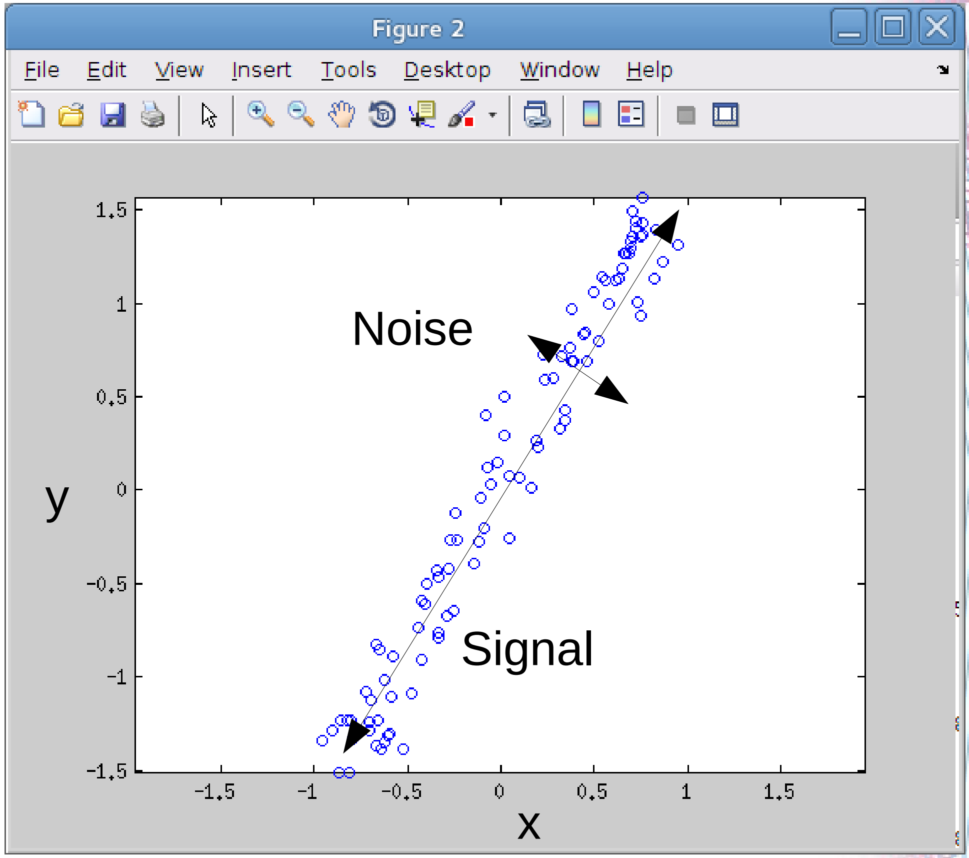
Concept: Covariance

- Consider two random variables, x and y.
  - Imagine they correspond to two different time series
- Suppose x and y track each other (statistically dependent).



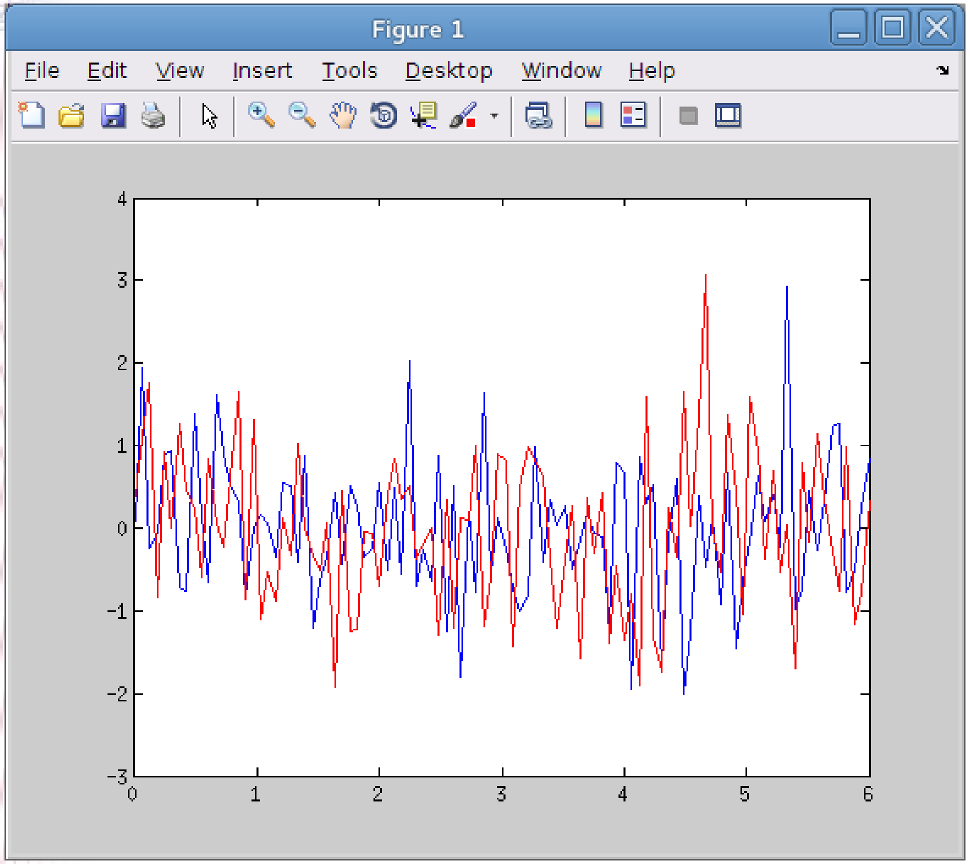
One way to see covariance

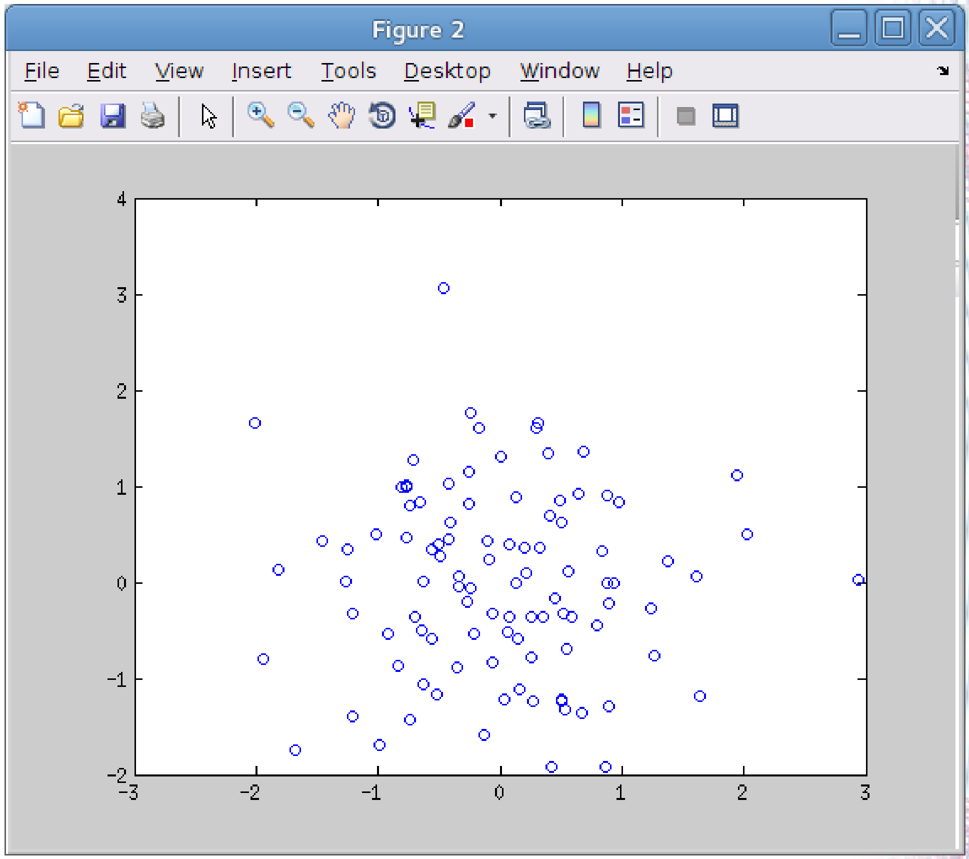




- Plot x vs. y – scatterplot
- Correlation between signals shows up as closely grouped points in scatterplot

Another signal – uncorrelated noise





- Uncorrelated noise signals show up as round scatterplot.

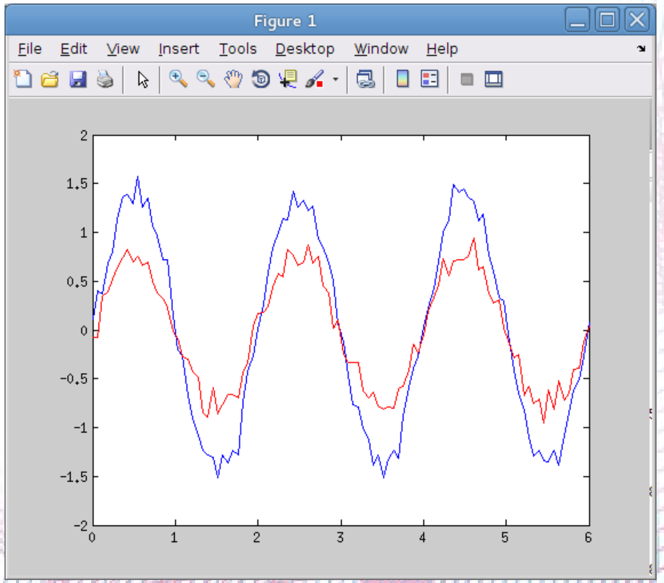
Convariance (scalar)

- We can calculate the degree to which the two variables track each other using covariance:

$$cov(x,y)=E\big((\underbrace{x-E(x)}_{}))(\underbrace{y-E(y)}_{})\big)$$

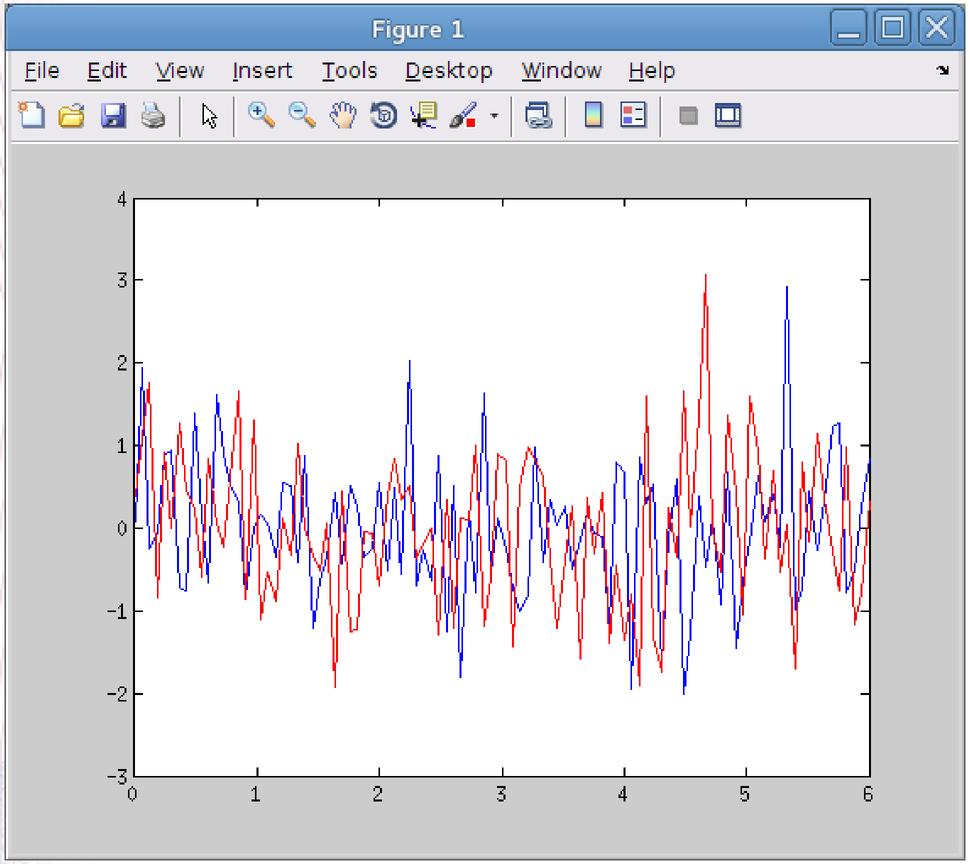
1. Subtract off mean
2. Multiply pairwise
3. Take mean of series

```
N = 100;
t = linspace(0, 6, N);

ny = .1*randn(size(t));
y = 1.4*sin(pi*t) + ny;

nx = .1*randn(size(t));
x = 0.8*sin(pi*t) + nx;

cxy = (x-mean(x))*(y-mean(y))'/N;
```

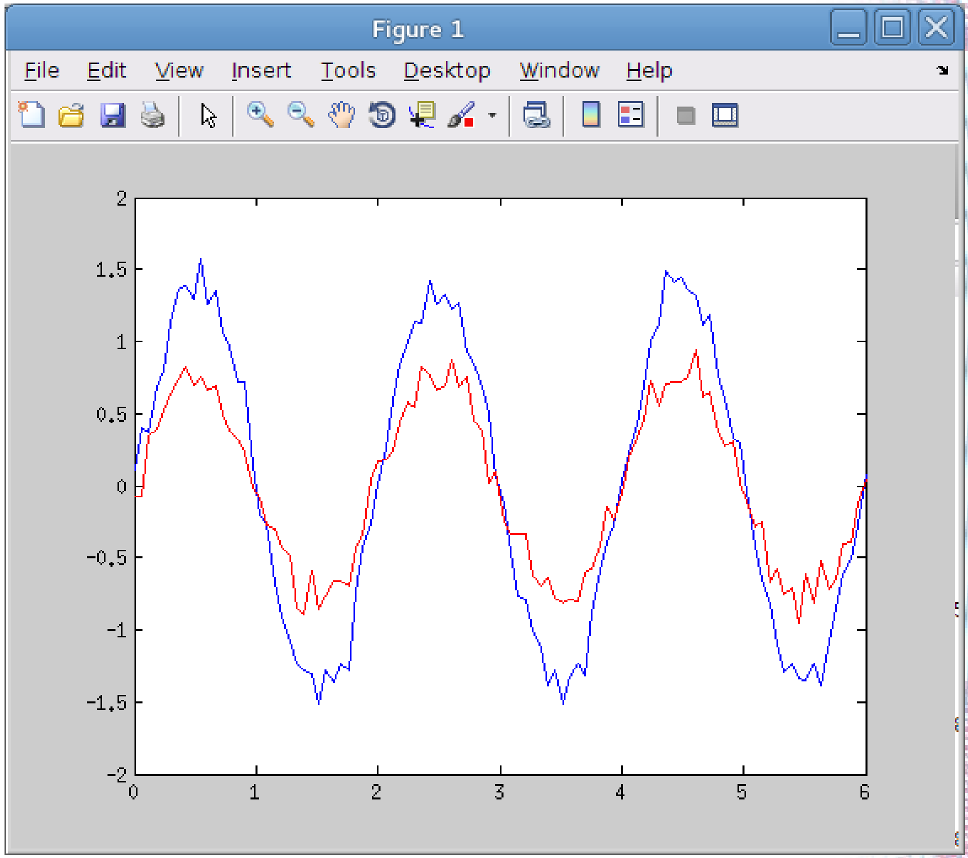


Consider two signals

$$cov(x,y)=E((x-E(x))(y-E(y)))$$



Covariance cxy = -0.026734



Covariance cxy = 0.546910

Variance and covariance

- Variance if same signal is used

$$var(x) = E((x - E(x))(x - E(x)))$$

- Covariance if different signals are used

$$cov(x, y) = E((x - E(x))(y - E(y)))$$

Covariance matrix

- We have 2 signals:

$$x_{1,} x_2$$

- Create matrix whose elements are

$$c_{ij} = cov(x_i, x_j)$$

$$= E((x_i - E(x_i))(x_j - E(x_j)))$$

- Matrix will be

$$S = \begin{pmatrix} var(x_{1,} x_1) & cov(x_{1,} x_2) \\ cov(x_{2,} x_1) & var(x_{2,} x_2) \end{pmatrix}$$

Three covariance trials

```
>> two_noise_series
Covariance cxy = -
0.031540

Covariance matrix C =
    0.6989   -0.0315
   -0.0315    0.8902

>> two_time_series
Covariance cxy =
0.556123

Covariance matrix C =
    0.3208    0.5561
    0.5561    0.9986
```

```
>> two_noise_series
Covariance cxy =
0.121277

Covariance matrix C =
    0.9234    0.1213
    0.1213    1.1106

>> two_time_series
Covariance cxy =
0.550047

Covariance matrix C =
    0.3183    0.5500
    0.5500    0.9912
```

```
>> two_noise_series
Covariance cxy = -
0.013757

Covariance matrix C =
    0.9980   -0.0138
   -0.0138    0.7295

>> two_time_series
Covariance cxy =
0.561387

Covariance matrix C =
    0.3404    0.5614
    0.5614    0.9624
```

- Covariance of noise signals is small -> statistically independent. Off-diag terms small.
- Covariance of sine waves is large -> statistically related. Off-diag terms large.

What if we have multiple signals?

- Consider 3 signals:

$$x_1, x_2, x_3$$

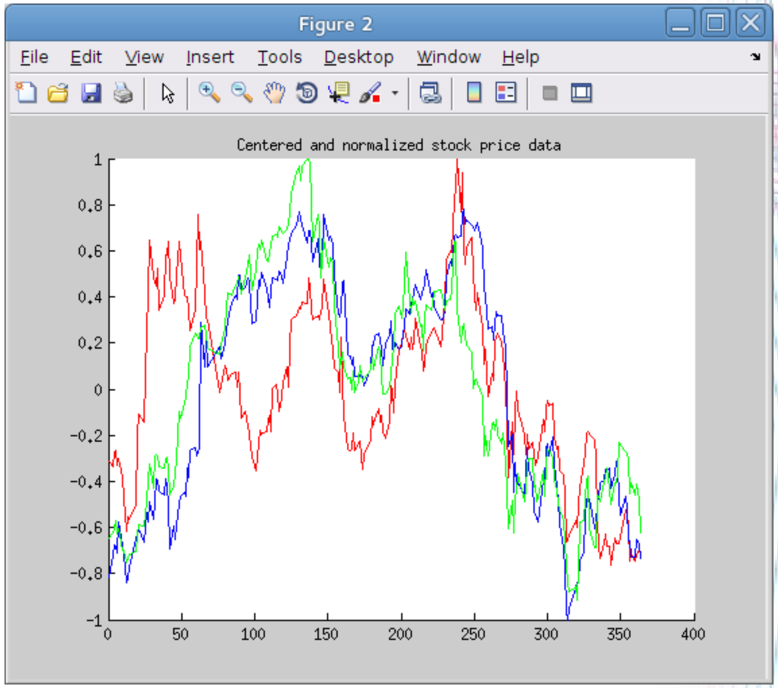
- Consider computing covariances for all signal pairs:

$$c_{ij} = cov(x_i, x_j)$$
$$= E((x_i - E(x_i))(x_j - E(x_j)))$$

- Arrange into matrix:

$$S = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix}$$

Covariance matrix



How to calculate covariance matrix?

1. Start with time series data arranged in rows in a data matrix M.

$$M = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & \cdots \\ y_1 & y_2 & y_3 & y_4 & y_5 & \cdots \\ z_1 & z_2 & z_3 & z_4 & z_5 & \cdots \end{pmatrix}$$

2. Subtract off mean for each row. (Zero-center the data)

```
for idx = 1:N
  r = M(idx, :);
  r = r-mean(r);
  M(idx, :) = r;
end
```

3. Compute matrix product (performs sums along rows)

$$S = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & \cdots \\ y_1 & y_2 & y_3 & y_4 & y_5 & \cdots \\ z_1 & z_2 & z_3 & z_4 & z_5 & \cdots \end{pmatrix} \begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \\ x_4 & y_4 & z_4 \\ x_5 & y_5 & z_5 \\ \vdots & \vdots & \vdots \end{pmatrix}$$

$$= M\, M^T$$

Written as matrix multiplications

4. Normalize:

Covariance matrix S

$$S = \frac{1}{N} M\, M^T$$

Use this expression when data is placed in rows of matrix M.

N = number of rows

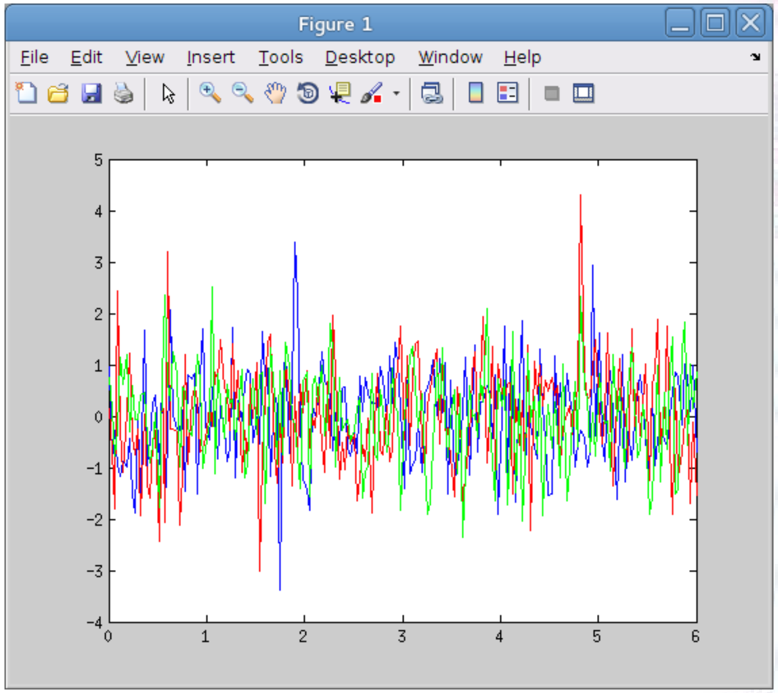
Covariance matrix properties

- On-diagonal elements are variances of each individual data series.
- Off-diagonal elements are covariances between dataset i and j.
- Covariance matrix is symmetric.
- Covariance matrix is positive semi-definite.
    - Eigenvalues are non-negative.

$$S=\begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix}$$

Covariance matrix properties....

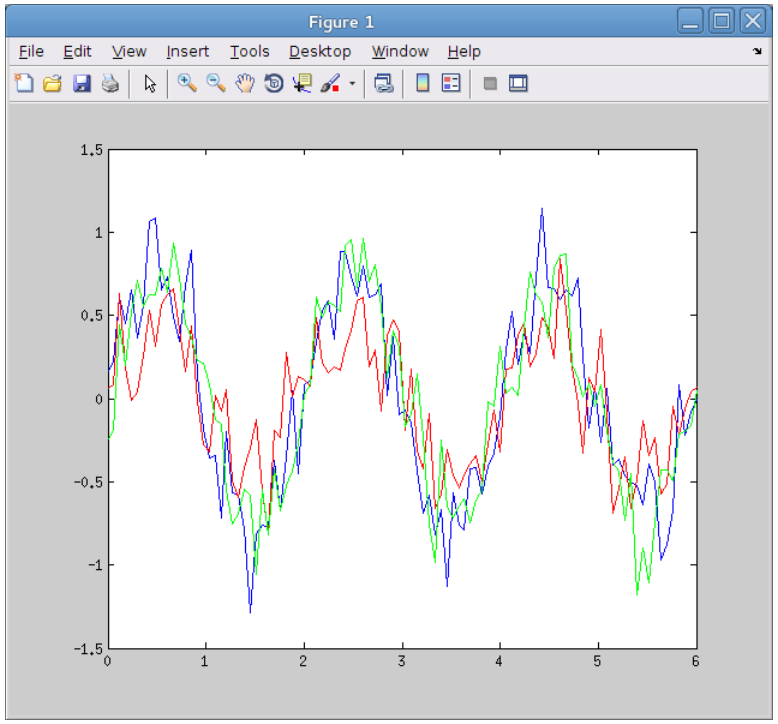
- Consider three uncorrelated random processes.
- On-diagonal elements of covariance matrix are large
- Off-diagonal elements of covariance matrix are small



```
>> three_noise_series

Covariance matrix C =
    0.9308    0.0027   -0.0903
    0.0027    1.0858    0.0709
   -0.0903    0.0709    0.8793
```

Covariance matrix properties...

- Consider three correlated random processes
- Off diagonal elements of covariance matrix are not small. This says that the processes are correlated.



```
>> three_time_series

Covariance matrix C =
      0.3352     0.1703     0.2759
      0.1703     0.1404     0.1631
      0.2759     0.1631     0.3114
```

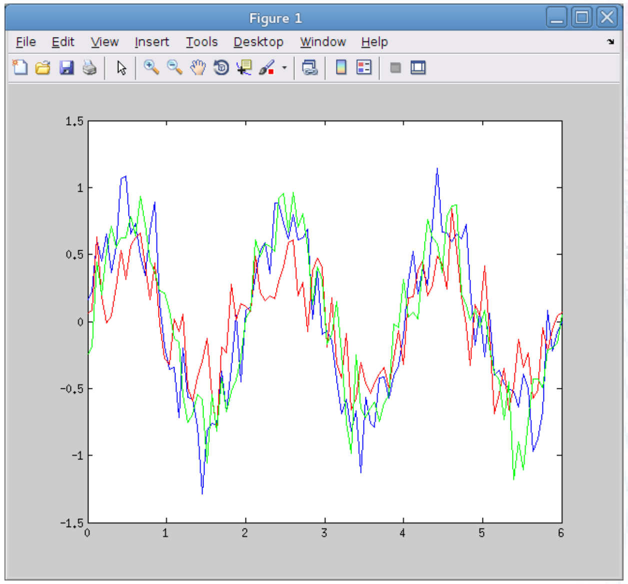
Symptoms of correlated signals

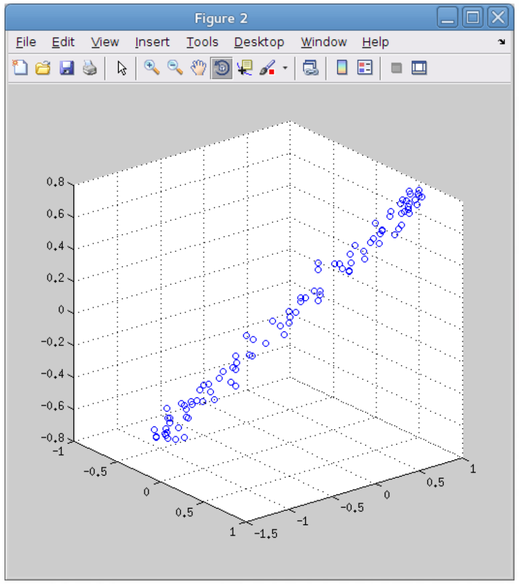
- Three time-varying signals evince correlated behavior

- Structure of scatterplot suggests signals are not independent.

- Off-diagonal terms in covariance matrix suggest correlation between signals

```
Covariance matrix C =
     0.3352     0.1703     0.2759
     0.1703     0.1404     0.1631
     0.2759     0.1631     0.3114
```

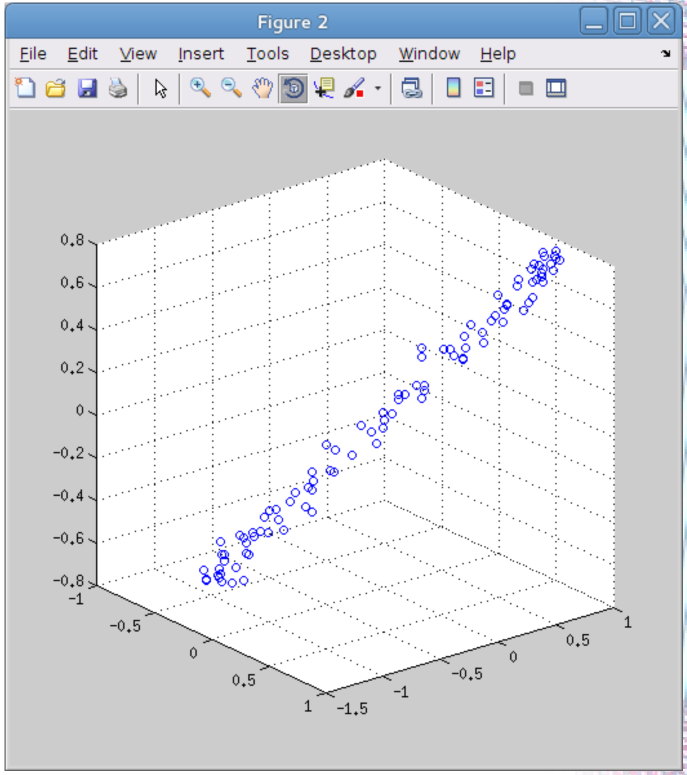




Correlated signals suggest underlying dynamics

Goal: find principal axes of correlation

- Can we find dominant axis of scatterplot?
- Yes – by finding eigenvectors of covariance matrix.
- This technique is called “principal component analysis” or PCA.



- PCA can be useful for identifying underlying dynamics of a signal or data source.

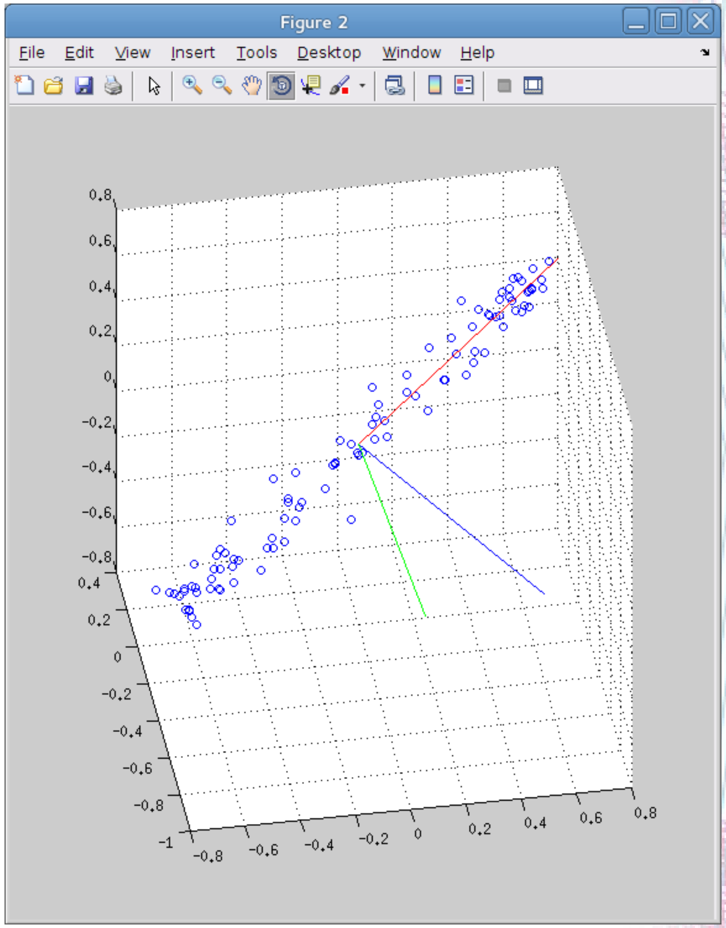
PCA algorithm

Data matrix

1. Put time series data into rows M
2. Subtract off mean from each row (zero-center data)
3. Form covariance matrix $S = M\,M^T$
4. Do eigenvalue decomposition of S: $S \rightarrow Q^T \Lambda Q$
5. Sort eigenvalues and eigenvectors from high to low.
6. Eigenvectors point along principal directions variance.

Example PCA run

```
C = M*M'/N;

% Do eigenvalue decomposition
[V,D] = eig(C);

% Sort eigenvalues in decreasing order
[Ds, idx]=sort(diag(D), 1, 'descend');
% Rearrange eigenvectors to
% match eigenvalues.
for cnt=1:length(Ds)
  Vs(:,cnt)=V(:,idx(cnt));
end

% Now plot eigenaxes
e1 = horzcat([0;0;0], Vs(:,1))';
plot3(e1(:,1), e1(:,2), e1(:,3), 'r')

e2 = horzcat([0;0;0], Vs(:,2))';
plot3(e2(:,1), e2(:,2), e2(:,3), 'b')

e3 = horzcat([0;0;0], Vs(:,3))';
plot3(e3(:,1), e3(:,2), e3(:,3), 'g')
```

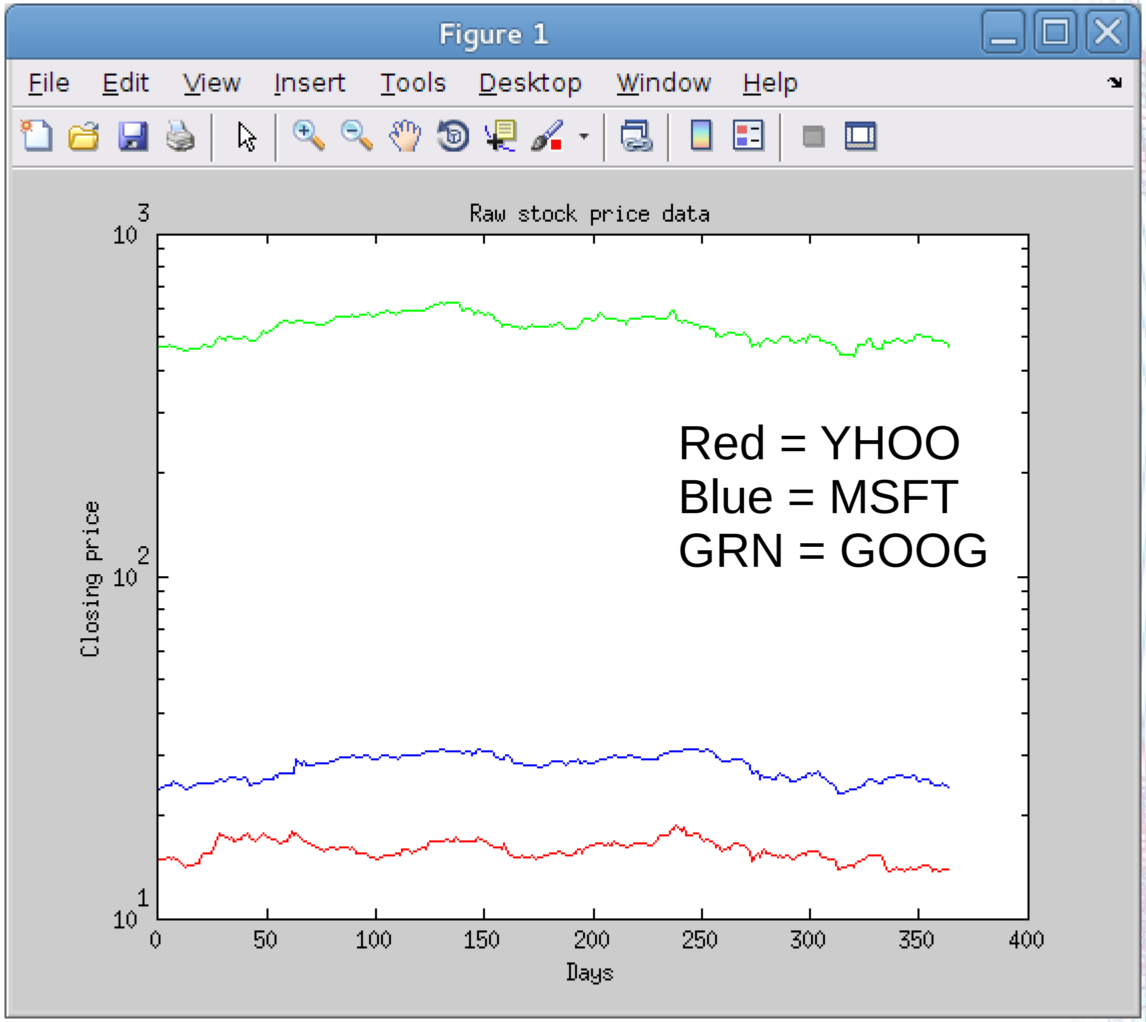


Remarks

- First eigenvector points in direction of maximum variance.
- Next eigenvector points in next direction of max variance, etc.
- These are the “principal components” of your data.
- If you have redundancy in your data (different measurements are correlated), PCA will show it to you.

Another example

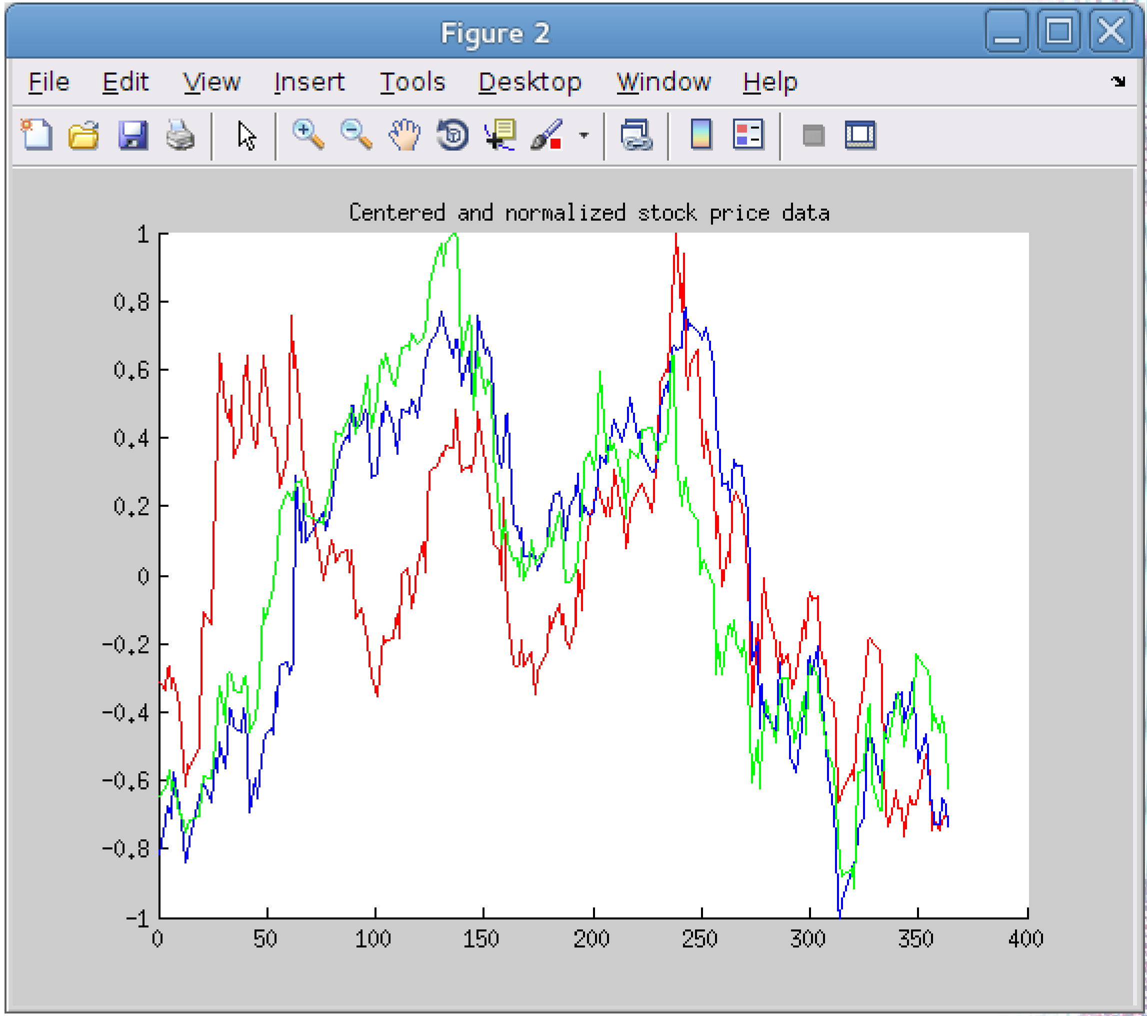
- One year of real stock market data
- Time series starts on 8.21.2009
- Ends on 8.20.2010



Source: http://pages.swcp.com/stocks/

Preprocessing

- Clean data
- Zero-center all data (i.e. subtract off mean)
- Normalize all data so abs(max(data)) = 1



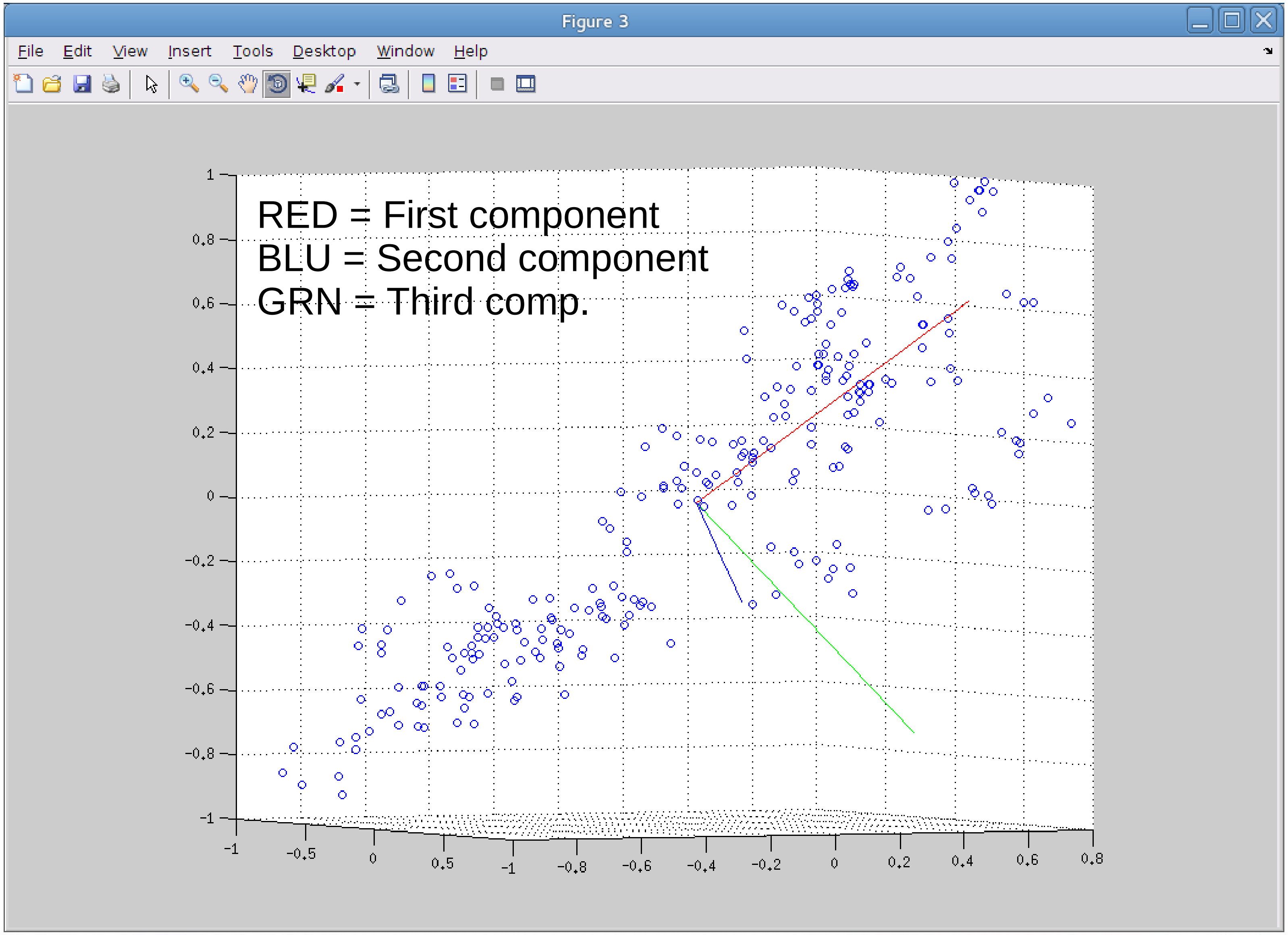
Covariance of different stocks is very evident

Preprocessing

```
% Normalize price data and make it zero-centered
% i.e. subtract off mean and normalize it.
for idx = 1:num_stocks
  S = A(idx, :);
  S = S-mean(S);
  peak = max(abs(S));
  S = S/peak;
  A(idx, :) = S;
end
```

- PCA requires zero-centered data.
- If you don't zero-center the data, then covariance matrix has large off-diagonal elements, and eigenvalue decomposition doesn't identify axes of variation.
- Normalizing data is optional but often recommended.

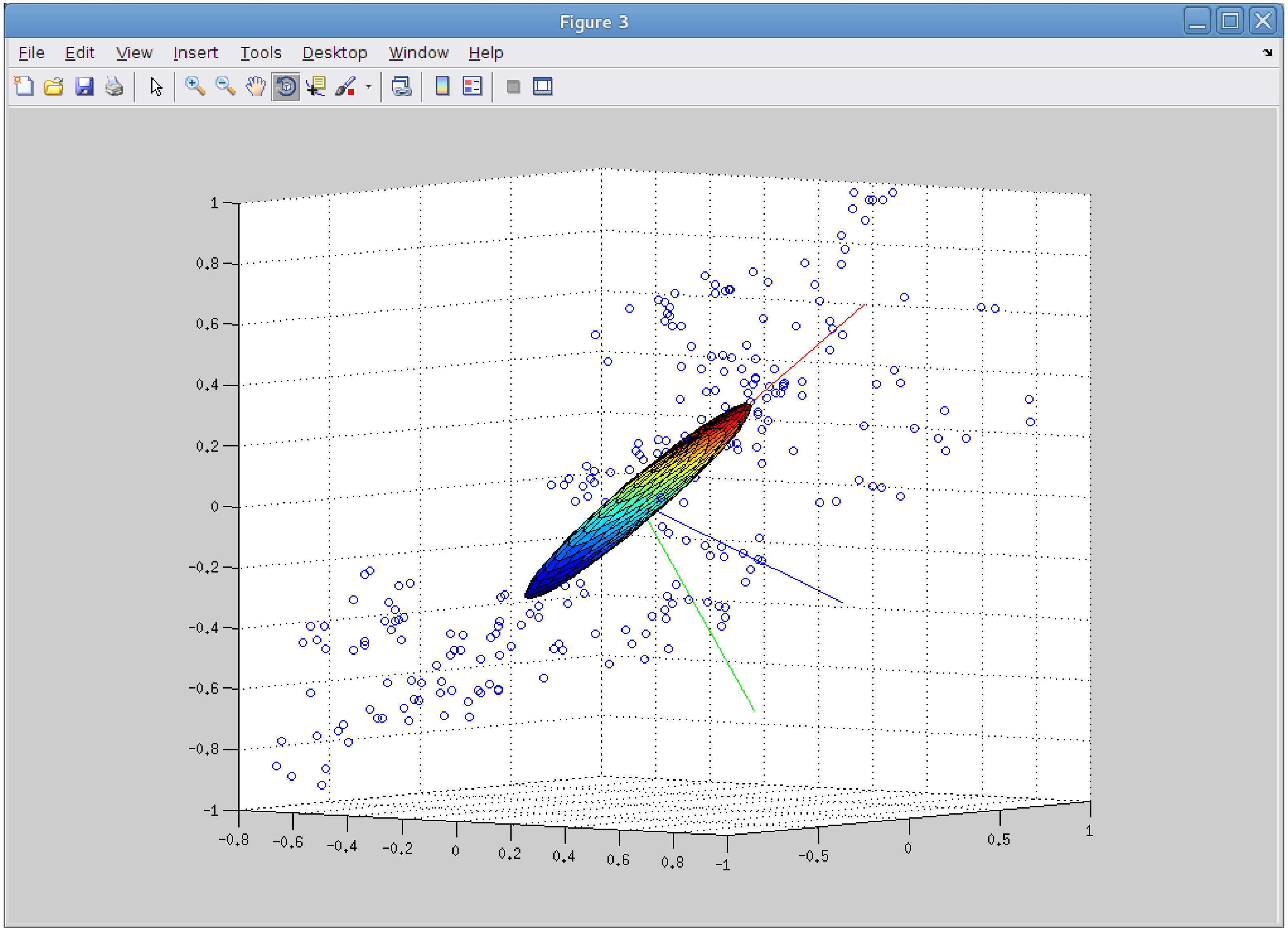
Perform PCA



First component corresponds to overall move of three stocks together

Why does PCA work?

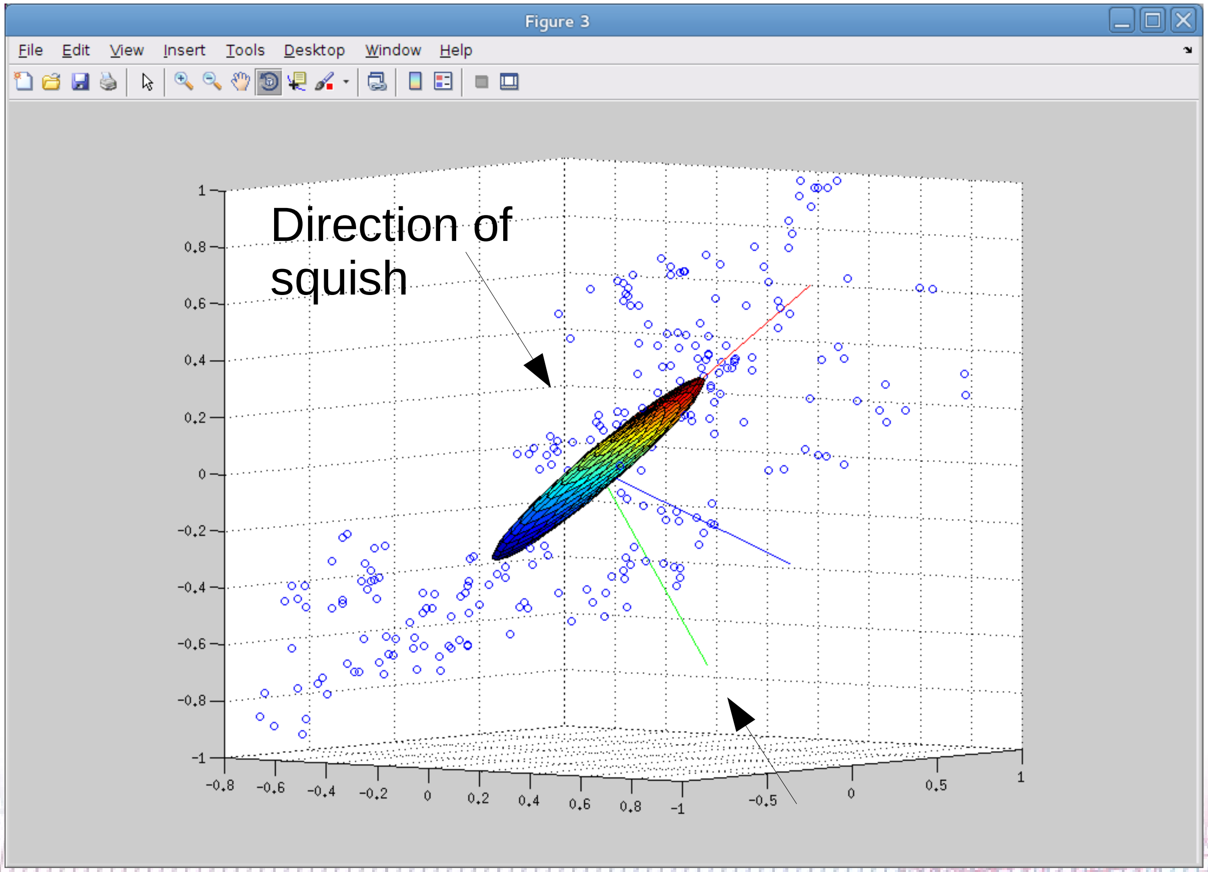
- Recall covariance matrix – symmetric, positive semi-definite.
- Recall how we visualized any matrix by looking at its action on a unit ball.
- In this case, the covariance matrix induces a 3D ellipse whose axes are aligned with the principal axes of the data's variation.
- The lengths of the axes correspond to the variance in each direction.



generate_data_ellipse_plot.m

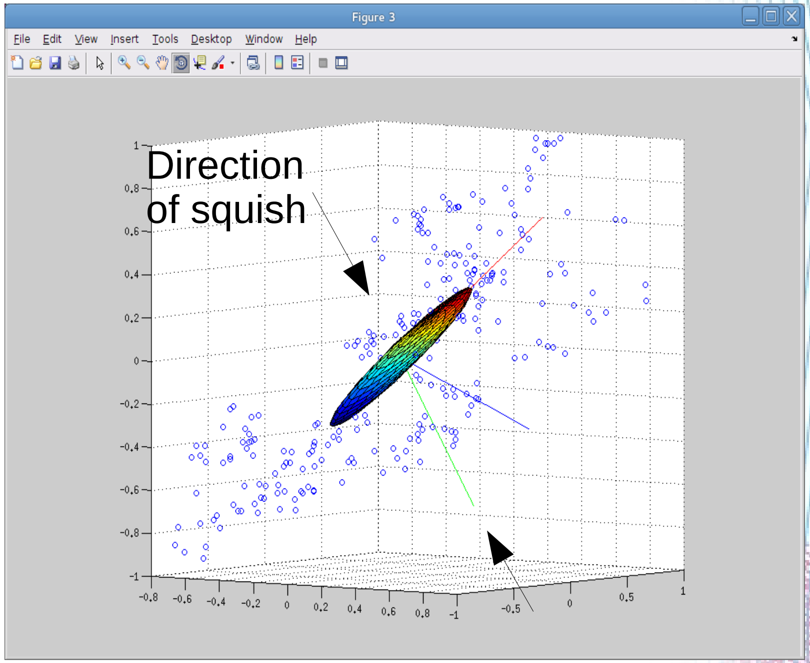
Next: Projecting to new basis

- The eigenvectors define a new basis set.
- We can project data to this new basis set.
- In this example, squish out third dimension



Projection onto subspace

- Project datapoints into new (reduced) basis.
- Only project onto first two axes – drop dependence on third (green) axis. This is dimensionality reduction.
- Projection is derived from eigenvectors.
- Projection operator is 2x3 matrix (dimensionality reduction)



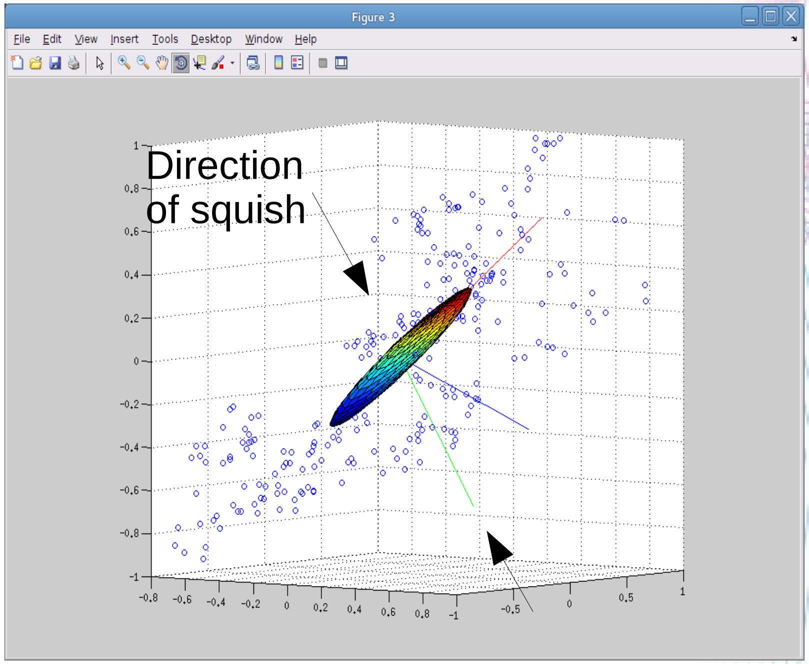
$$\underbrace{\begin{pmatrix} b_{1i} \\ b_{2i} \end{pmatrix}}_{\text{Projected data (2D)}} = \underbrace{\begin{pmatrix} \cdots \hat{e}_1 \cdots \\ \cdots \hat{e}_2 \cdots \end{pmatrix}}_{\text{First 2 eigenvectors}} \underbrace{\begin{pmatrix} a_{1i} \\ a_{2i} \\ a_{3i} \end{pmatrix}}_{\text{Input data (3D)}}$$

Projection matrix

- Consider what this does:

$$\begin{pmatrix} b_{1i} \\ b_{2i} \end{pmatrix} = \begin{pmatrix} \cdots \hat{e}_1 \cdots \\ \cdots \hat{e}_2 \cdots \end{pmatrix} \begin{pmatrix} a_{1i} \\ a_{2i} \\ a_{3i} \end{pmatrix}$$

Projected data (2D) — First 2 eigenvectors — Input data (3D)

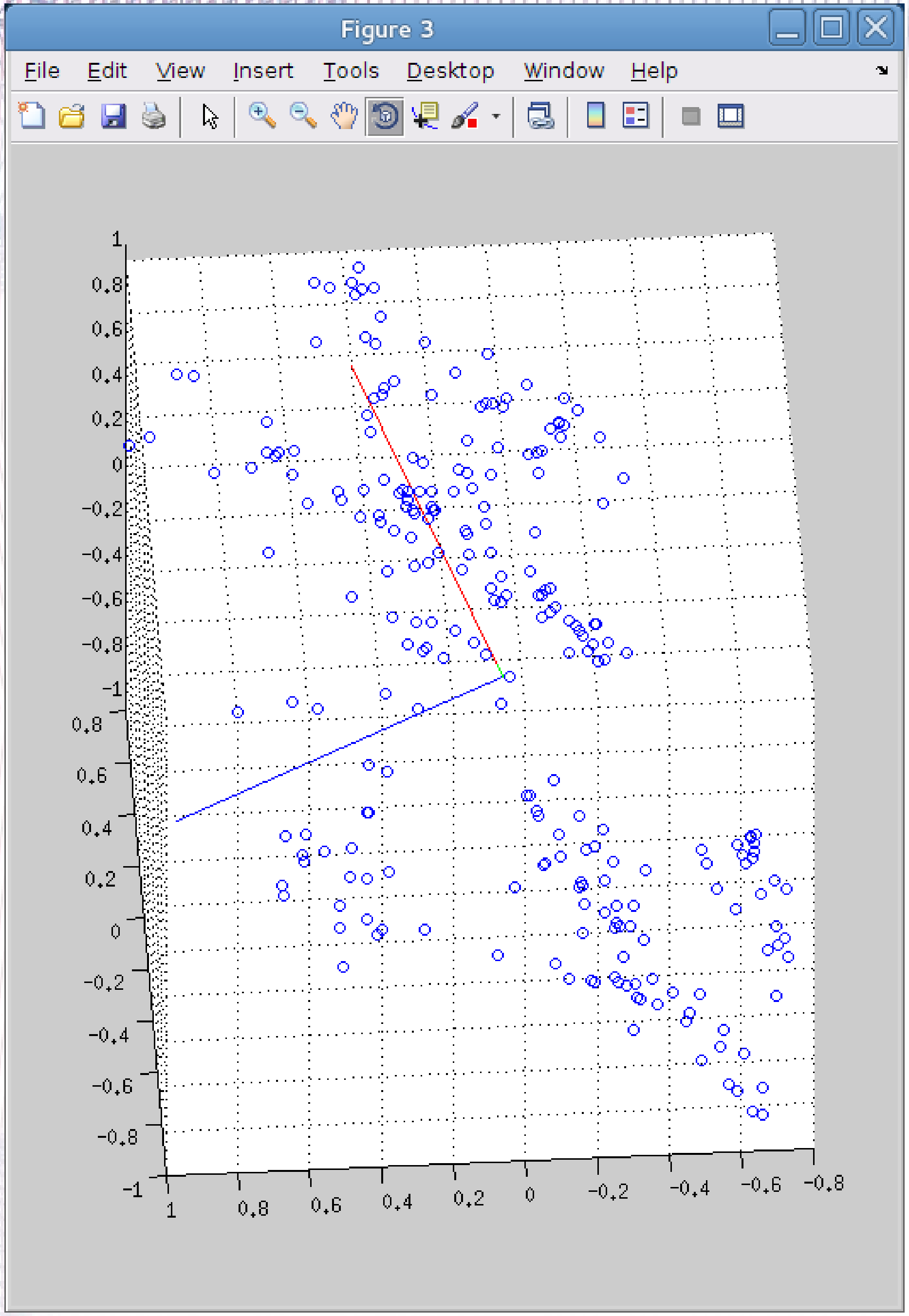


- Starts with data point $[a_1\ a_2\ a_3]$.
- Finds amount of *a* pointing in direction $\hat{e}_1^T a \Rightarrow b_1$
- Finds amount of *a* pointing in direction $\hat{e}_2^T a \Rightarrow b_2$
- Discards anything pointing in direction $\hat{e}_3$

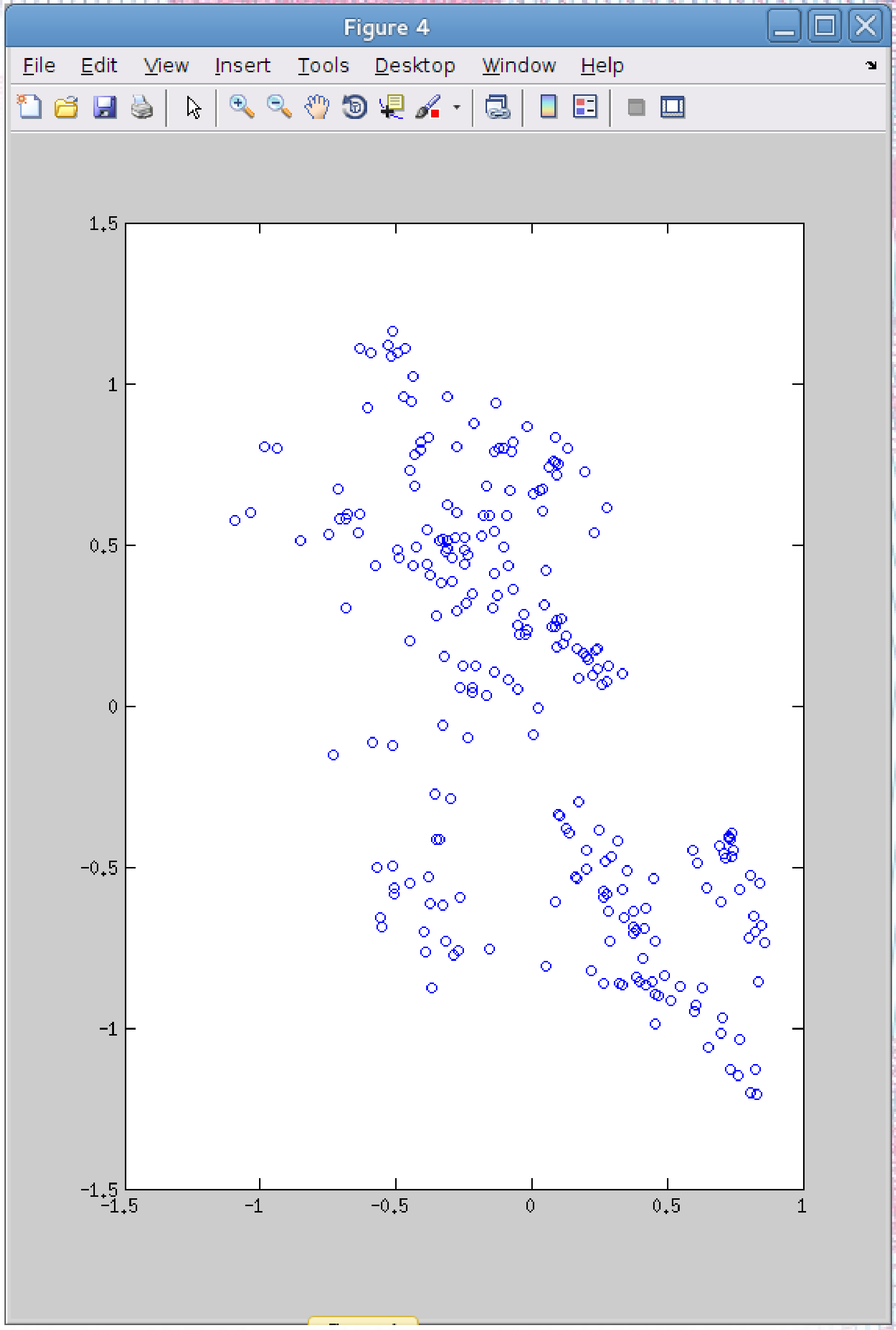
```
% Create matrix to hold projected data.
B = zeros(2, size(A,2));  % Projected space

% Create projection matrix from eigenvectors
VsT = Vs';
P = VsT(1:2,:);

% This does projection using the projection matrix P
for col = 1:size(A,2)
  B(:, col) = P*A(:,col);
end
```

Eigenvectors found in PCA step used to create projection matrix P

New datapoints



3D plot before projection



2D plot after projection

run_analysis.m

Sounds like the SVD?

- Classical PCA was framed in terms of doing an eigenvalue decomposition of the covariance matrix.
    - Perhaps because it was invented long before the SVD became well known.
- Dimensionality reduction and projection onto new bases are also associated with SVD analyses.
- You can also (almost) think of PCA as simply doing an SVD on the data matrix itself.

Relationship between PCA and SVD

- Recall relation of SVD to eigendecomposition:

$$svd(A) \Leftrightarrow eig(A^T A) = eig(A A^T)$$

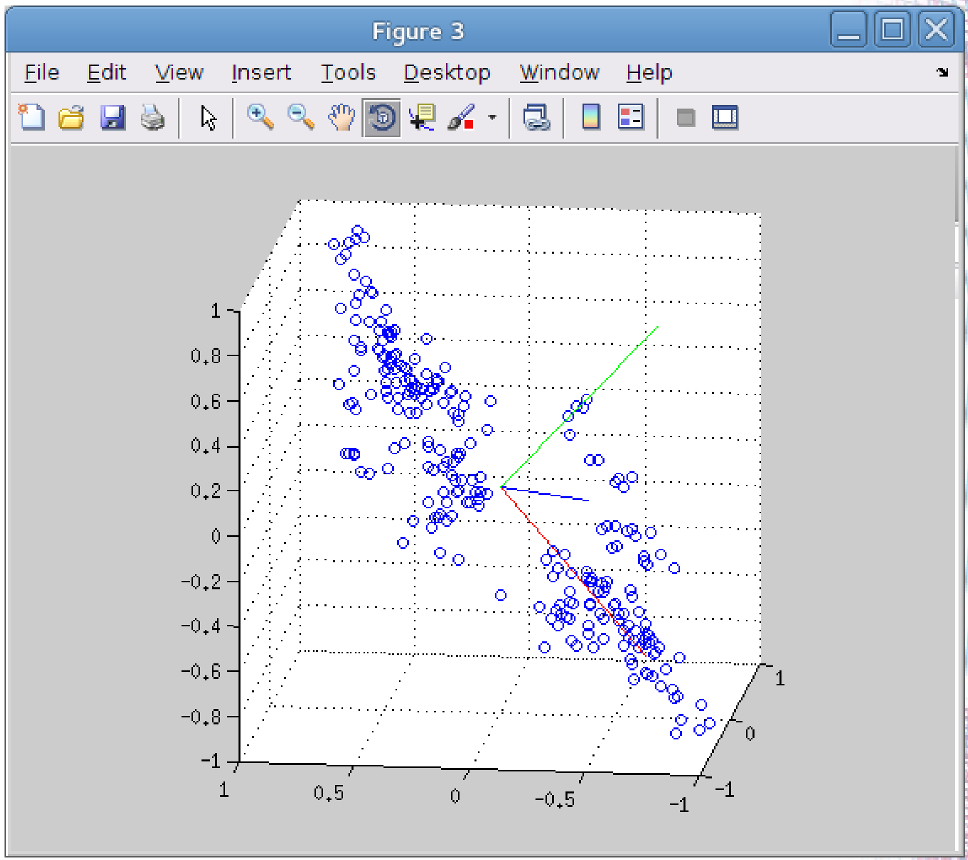
Homework problem

- Doing eigendecomposition of covariance matrix A^TA is almost same as doing SVD on original matrix A.
- One difference: Covariance matrix is zero-centered (subtract means from rows).
  - Therefore, subtract mean from each row prior to SVD.
- Otherwise, PCA using eigenvalue decomposition and SVD are equivalent.

PCA using SVD

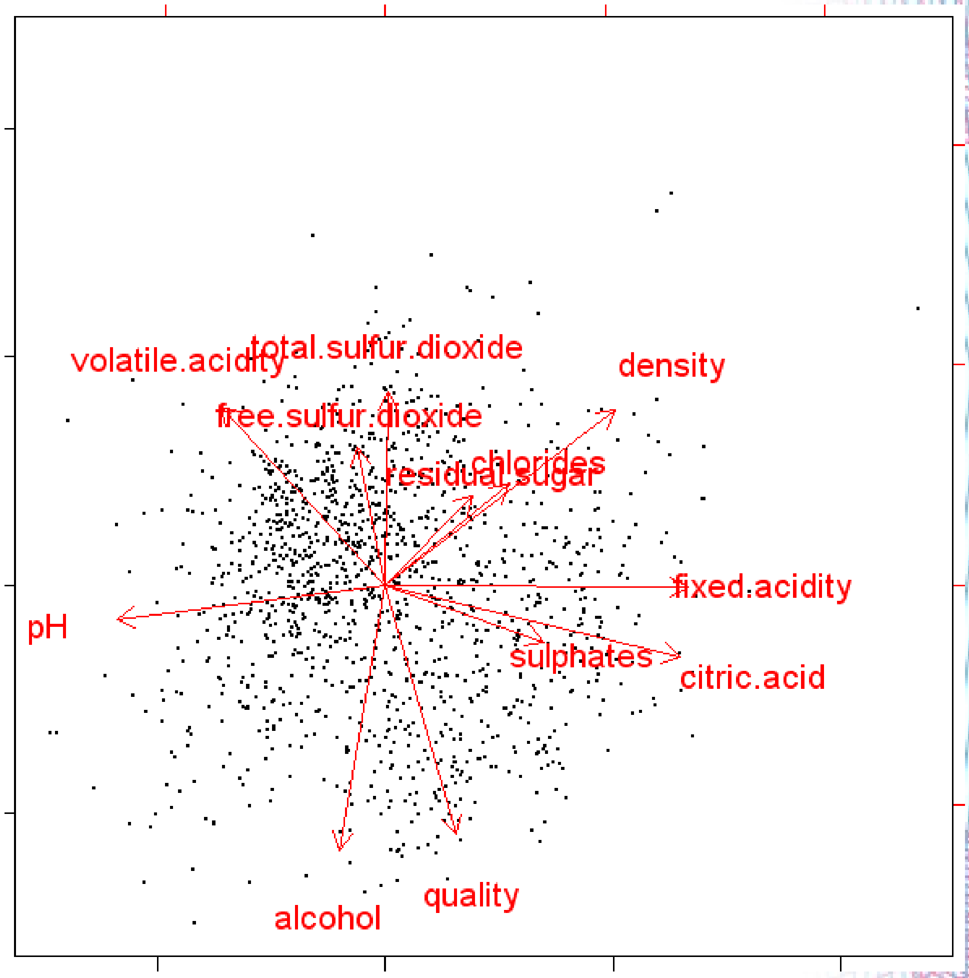
$$A \rightarrow U\,S\,V^T$$

- Since my data are in rows, the eigenvectors of the principal components show up in columns of U.
- Note that the axes may change sign (flip directions) between PCA runs using eigenvectors and svd.



Example PCA applications

- Examine and separate tastes in wine.
- Dimensions of manufactured parts.
  - Are more errors along certain dimensions?
- Epidemiology
- Genetics/bioinformatics



Session topics

- QR algorithm for eigenvalues
- Covariance matrix
  - Matrix is PSD
  - Zero off-diagonals indicate statistically independent data series.
  - Non-zero elements indicate correlation between corresponding data series.
- PCA
  - Finds main axes of interesting dynamics
  - Computed using eigenvalue decomposition.
  - Or computed using SVD.