Final_Project

December 14, 2018

Math693a
Final Project
Subspace Minimization: From the Lanczos Method to PCA
Matteo Polimeno

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        from matplotlib.pyplot import figure
        from numpy import linalg as LA
        %matplotlib inline
```

A projection method for solving a linear system of the form

$$A\mathbf{x} = \mathbf{b}$$

is a method which looks for an approximate solution $\mathbf{x_m}$ from an affine subspace $\mathbf{x_0} + \mathcal{K}_{\Updownarrow}$ of dimension $m << dim(A)$, by imposing the so-called Petrov-Galerkin condition:

$$b - A\mathbf{x_m} \perp \mathcal{L}_{\Updownarrow}$$

where $\mathcal{L}_{\Updownarrow}$ is another subspace of dimension m and $\mathbf{x_0}$ represents an arbitrary initial guess to the solution. A Krylov subspace method is simply a method where $\mathcal{K}_{\Updownarrow}$ is the Krylov subspace:

$$K_m(A, \mathbf{r_0}) = span\{\mathbf{r_0}, A\mathbf{r_0}, A^2\mathbf{r_0}, ..., A^{m-1}\mathbf{r_0}\}$$

where $\mathbf{r_0} = \mathbf{b} - A\mathbf{x_0}$. Since K_m then has dimension m, then we can approximate m eigenvalues and eigenvectors. The best approximations are called *Ritz Values* and *Ritz Vectors*. For further reference see *Lesson 11, slide 9*.

An important tool to accomplish subspace minimization is represented by Iterative methods. Four of the most important algorithms and their relatioships are shown in the following table.

```
In [52]: from IPython.display import Image
         Image(filename = "Lanczos.png", width=500, height=500) #image from Trefethen, Bau: Nu
```

Out[52]:

	$Ax = b$	$Ax = \lambda x$
$A = A^*$	CG	Lanczos
$A \neq A^*$	GMRES CGN BCG et al.	Arnoldi

As we saw in class (HW 4), the CG method is guaranteed to solve a linear system if the eigenvalues are clustered away from the origin. In the same way, the Lanczos iteration computes certain eigenvalues of a real symmetric matrix if those eigenvalues are well-separated from the rest of the spectrum (see *Numerical Linear Algebra, Trefeten, Bau*).

Alright, so given that we saw the CG method in class and that we do not like non-symmetric matrices, we are going to focus on the Lanczos method and show some, hopefully, interesting results. Essentially, the goal of our algorithm is to build a tridiagonal matrix of the from:

```
In [53]: from IPython.display import Image
         Image(filename = "T_matrix.png", width=500, height=500) #image from Trefethen, Bau: N
```

Out[53]:

$$T_n = \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \beta_2 & \alpha_3 & \ddots & \\ & & \ddots & \ddots & \beta_{n-1} \\ & & & \beta_{n-1} & \alpha_n \end{bmatrix}.$$

where the entries are derived through the iterations. Let's take a look.

```
In [ ]: def Lanczos_demmel(A,v,m,n):
            bjj = 0
            vo = np.zeros(n)
            T = np.zeros((m,m))
            V = np.zeros((m,n))
```

```
            for jj in np.arange(m-1):
                w = np.dot(A,v)
                a = np.dot(w,v)
                w = w - a*v - bjj*vo
                bjj = np.sqrt(np.dot(w,w))
                vo = v
                v = w/bjj
                eigT,evcT = LA.eig(T)
                eigT[::-1].sort()
                T[jj,jj] = a
                T[jj+1,jj] = bjj
                T[jj,jj+1] = bjj
```

In [24]:
```
def Lanczos_demmel(A,v,m,n):
    bjj = 0
    vo = np.zeros(n)
    T = np.zeros((m,m))
    V = np.zeros((m,n))

    for jj in np.arange(m-1):
        w = np.dot(A,v)
        a = np.dot(w,v)
        w = w - a*v - bjj*vo
        bjj = np.sqrt(np.dot(w,w))
        vo = v
        v = w/bjj
        eigT,evcT = LA.eig(T)
        eigT[::-1].sort()
        T[jj,jj] = a
        T[jj+1,jj] = bjj
        T[jj,jj+1] = bjj
        V[jj,:] = v
        eigA,evcA = LA.eig(A)
        eigA[::-1].sort()
        jet= plt.get_cmap('jet')
        colors = iter(jet(np.linspace(0,1,m)))
        for ii in np.arange(m):
            if jj==0:
                figure(num=2, figsize=(12, 8))
                plt.plot(jj+1,eigT[ii], '+',color=next(colors),mew=2)
            if jj>0 and eigT[ii] !=0:
                figure(num=2, figsize=(12,8))
                plt.plot(jj+1,eigT[ii],'+',color=next(colors),mew=2)

    figure(num=2, figsize=(12, 8))
    plt.plot((m)*np.ones(n),eigA,'k+')
    plt.title('Convergence of Ritz Values',fontsize=25)
```

```
plt.xlabel('Iterations',fontsize=20)
plt.ylabel('$\lambda(T)$',fontsize=20)
plt.xlim([0,m])
plt.ylim([-4,4])
plt.grid()
figure(num=0, figsize=(12, 8))
plt.plot(eigA,'k-',linewidth=3)
plt.title('Eigenvalues of A',fontsize=25)
plt.xlabel('Row/Column Elements',fontsize=20)
plt.ylabel('$\lambda(A)$',fontsize=20)
plt.ylim([-4,4])
plt.grid()
figure(num=1, figsize=(12, 8))
plt.plot(eigT,'b-',linewidth=3)
plt.title('Ritz values as function of iterations',fontsize=25)
plt.xlabel('Iterations',fontsize=20)
plt.ylabel('$\lambda(T)$',fontsize=20)
plt.ylim([-4,4])
plt.grid()
err4 = np.str((np.abs(eigT[:4]-eigA[:4]))/np.abs(eigA[:4]))
errm4 = np.str((np.abs(eigT[-4:]-eigA[-4:]))/np.abs(eigA[-4:]))
eT = np.str(eigT)
eT4 = np.str(eigT[:4])
eA4 = np.str(eigA[:4])
eTm4 = np.str(eigT[-4:])
eAm4 = np.str(eigA[-4:])
print("Number of iterations: " + np.str(m))
print("Eigenvalues of T: " + eT)
print("Greatest 4 Eigenvalues of T: " + np.str(eT4))
print("Greatest 4 Eigenvalues of A: " + np.str(eA4))
print("Global Error for the greatest 4 Eigenvalues: " + np.str(err4))
print("Smallest 4 Eigenvalues of T: " + np.str(eTm4))
print("Smallest 4 Eigenvalues of A: " + np.str(eAm4))
print("Global Error for the smallest 4 Eigenvalues: " + np.str(errm4))
```

All the is left to do is to define a symmetric matrix, an initial vector v_1 and start iterating. Here we tried to recreate some of the results from *Applied Numerical Linear Algebra, Demmel*.

```
In [25]: #------- Define symmetric matrix A
         d = np.random.randn(1000,1000)
         A = np.diag(np.diag(d))
         n = len(A)
         v1 = np.random.randn(n); v1 /= np.sqrt(np.dot(v1,v1))
```

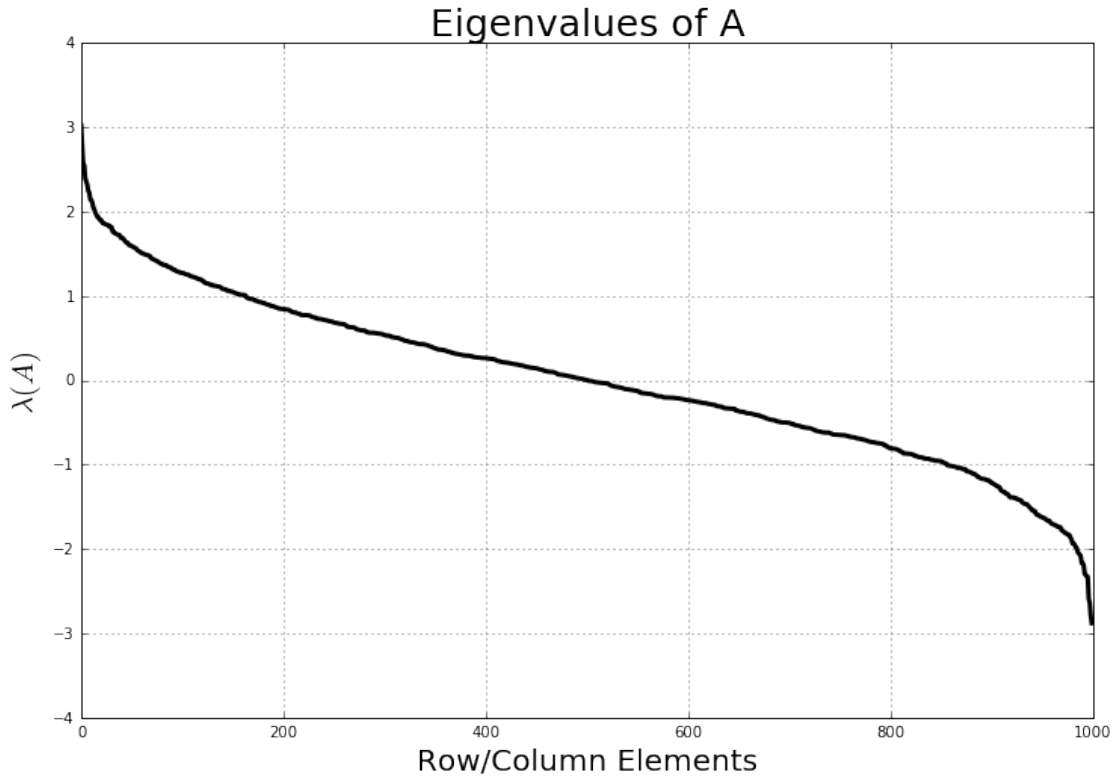
We will analize our results in terms of convergence of the Ritz Values to the eigenvalues of our matrix A and we will plot the global error, defined as followed:

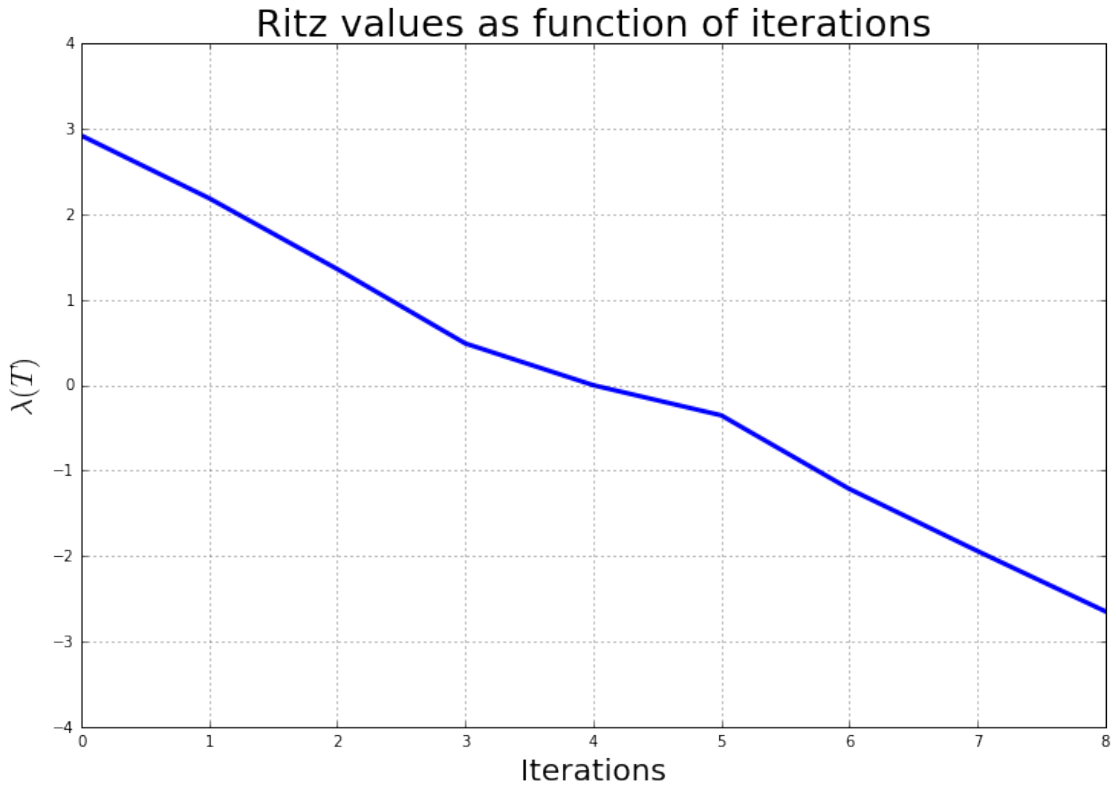
$$\frac{|\lambda_i(T) - \lambda_i(A)|}{|\lambda_i(A)|}$$

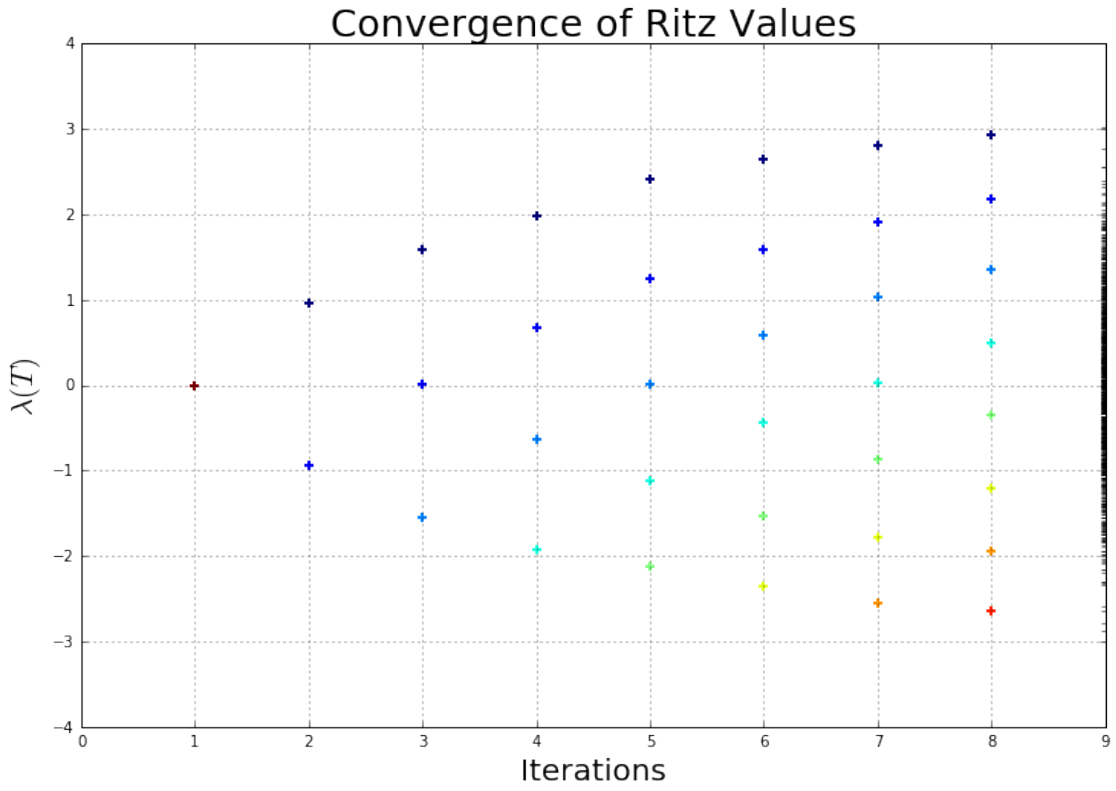
We divide by $|\lambda_i(A)|$ in order to normalize the error between 1 (no accuracy) and 10^{-16} (machine precision), which is where we would like to go obviously.

```
In [26]: m1 = 9
         Lanczos_demmel(A,v1,m=m1,n=n)
```

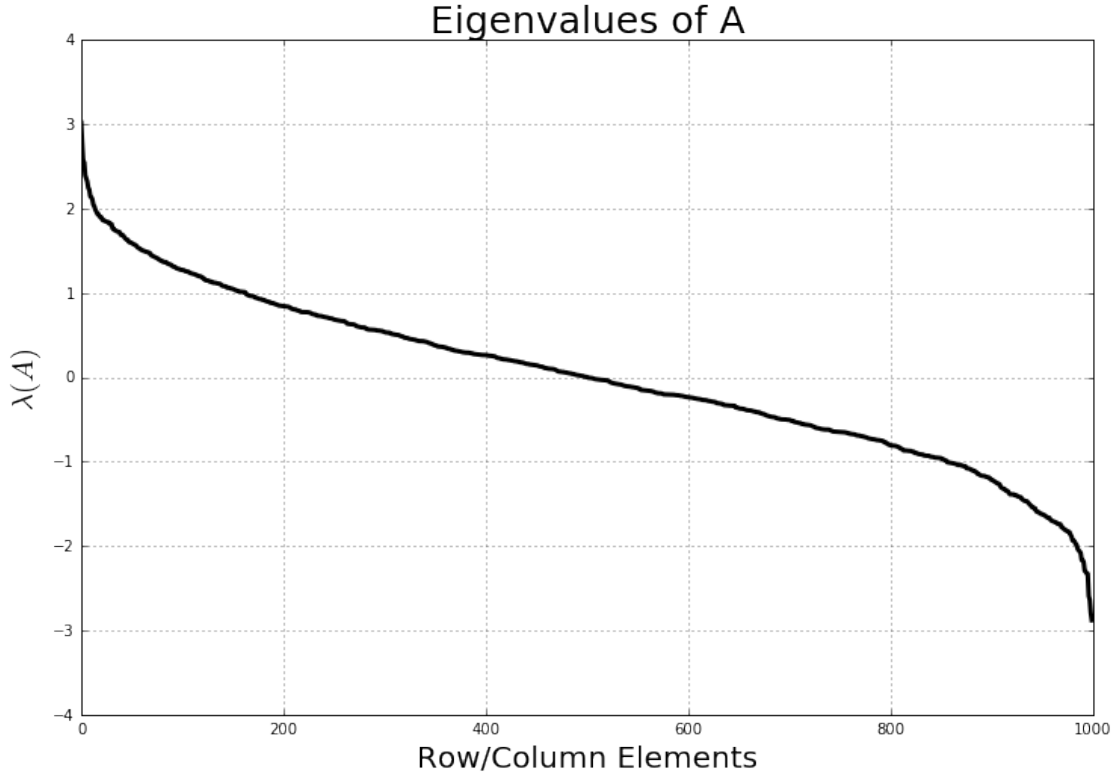
```
Number of iterations: 9
Eigenvalues of T: [ 2.92306485  2.18815857  1.3596351   0.48886923  0.         -0.35310949
 -1.21411453 -1.9406475  -2.64734486]
Greatest 4 Eigenvalues of T: [ 2.92306485  2.18815857  1.3596351   0.48886923]
Greatest 4 Eigenvalues of A: [ 3.01866708  2.768157    2.5516242   2.54973605]
Global Error for the greatest 4 Eigenvalues: [ 0.03167035  0.20952512  0.46714916  0.80826673]
Smallest 4 Eigenvalues of T: [-0.35310949 -1.21411453 -1.9406475  -2.64734486]
Smallest 4 Eigenvalues of A: [-2.59278381 -2.63721861 -2.79108077 -2.87975182]
Global Error for the smallest 4 Eigenvalues: [ 0.86381067  0.5396231   0.30469676  0.08070381]
```

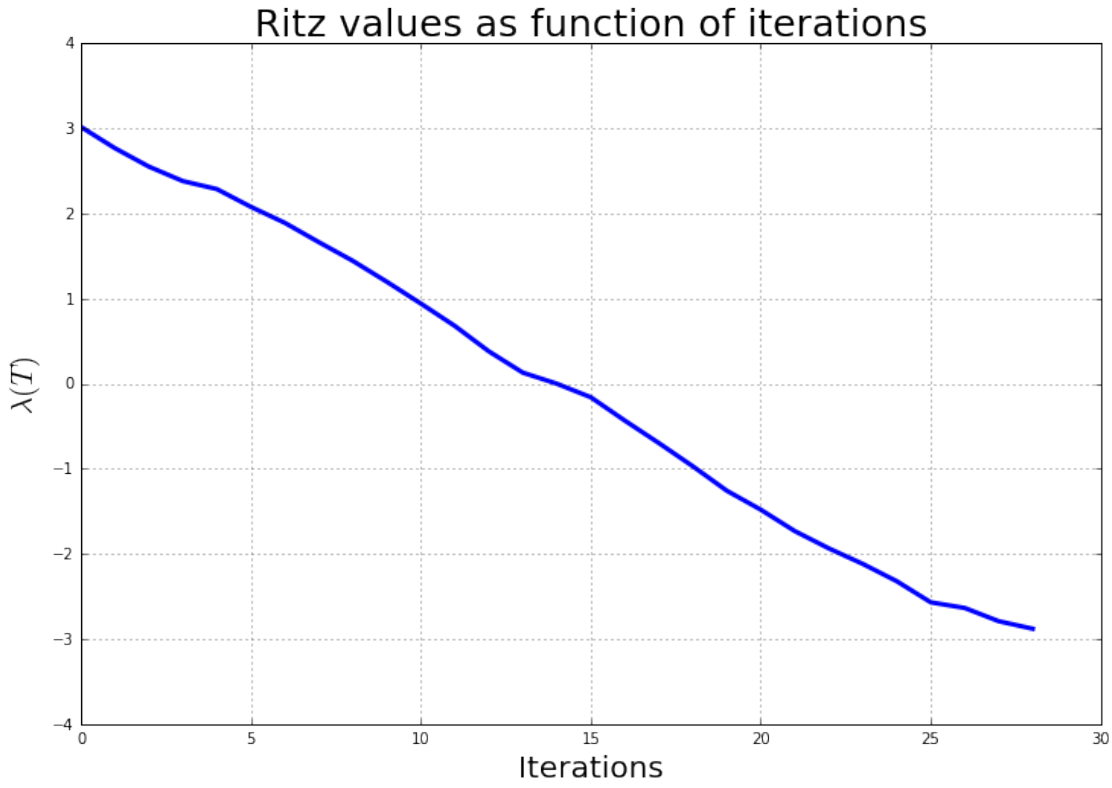


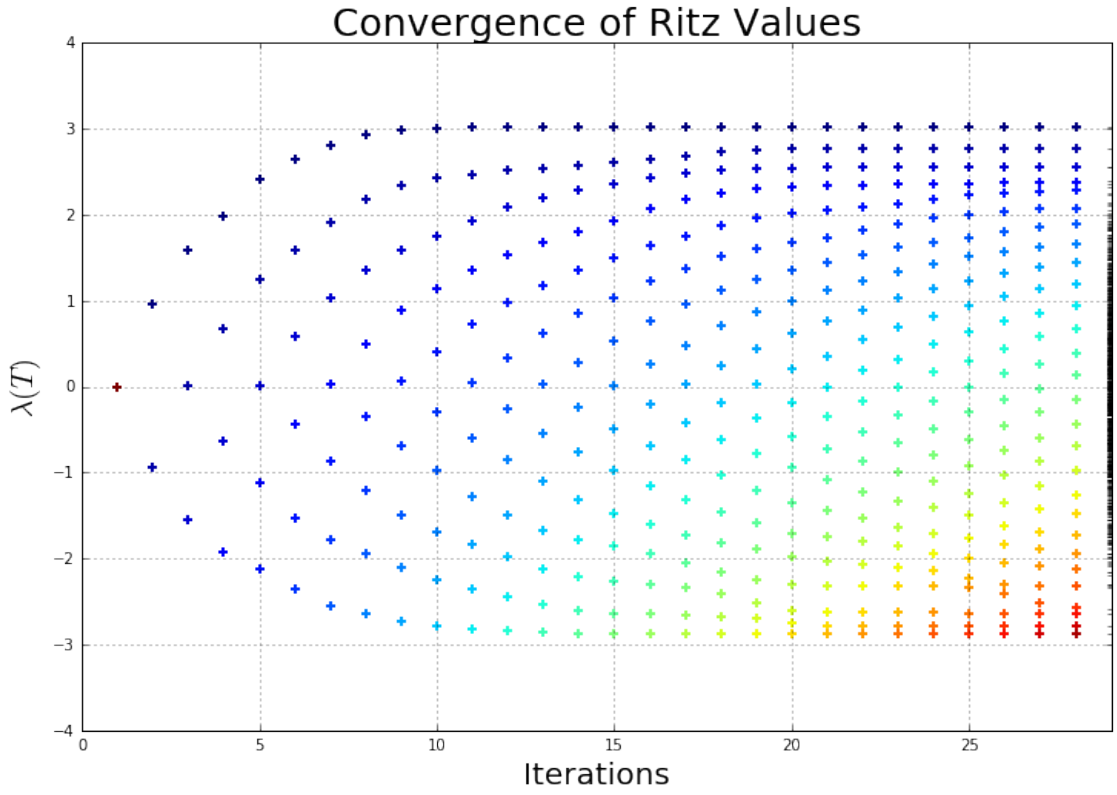

Ritz values as function of iterations
$\lambda(T)$
Iterations


Convergence of Ritz Values
$\lambda(T)$
Iterations

```
In [27]: A = A
         m2 = 29
         Lanczos_demmel(A,v1,m=m2,n=n)

Number of iterations: 29
Eigenvalues of T: [ 3.01866708  2.76815582  2.55154847  2.38129681  2.28715204  2.07890319
  1.89041997  1.66356588  1.44380643  1.1978529   0.94240415  0.67907122
  0.38065608  0.13020098  0.         -0.15899983 -0.43310982 -0.69688863
 -0.96994013 -1.25727171 -1.48038657 -1.73160292 -1.93486109 -2.11741474
 -2.3209559  -2.56883703 -2.63569655 -2.79099211 -2.87975073]
Greatest 4 Eigenvalues of T: [ 3.01866708  2.76815582  2.55154847  2.38129681]
Greatest 4 Eigenvalues of A: [ 3.01866708  2.768157    2.5516242   2.54973605]
Global Error for the greatest 4 Eigenvalues: [  1.47377673e-11   4.27063510e-07   2.96804819e-0
Smallest 4 Eigenvalues of T: [-2.56883703 -2.63569655 -2.79099211 -2.87975073]
Smallest 4 Eigenvalues of A: [-2.59278381 -2.63721861 -2.79108077 -2.87975182]
Global Error for the smallest 4 Eigenvalues: [  9.23593193e-03   5.77145125e-04   3.17640180e-0
```

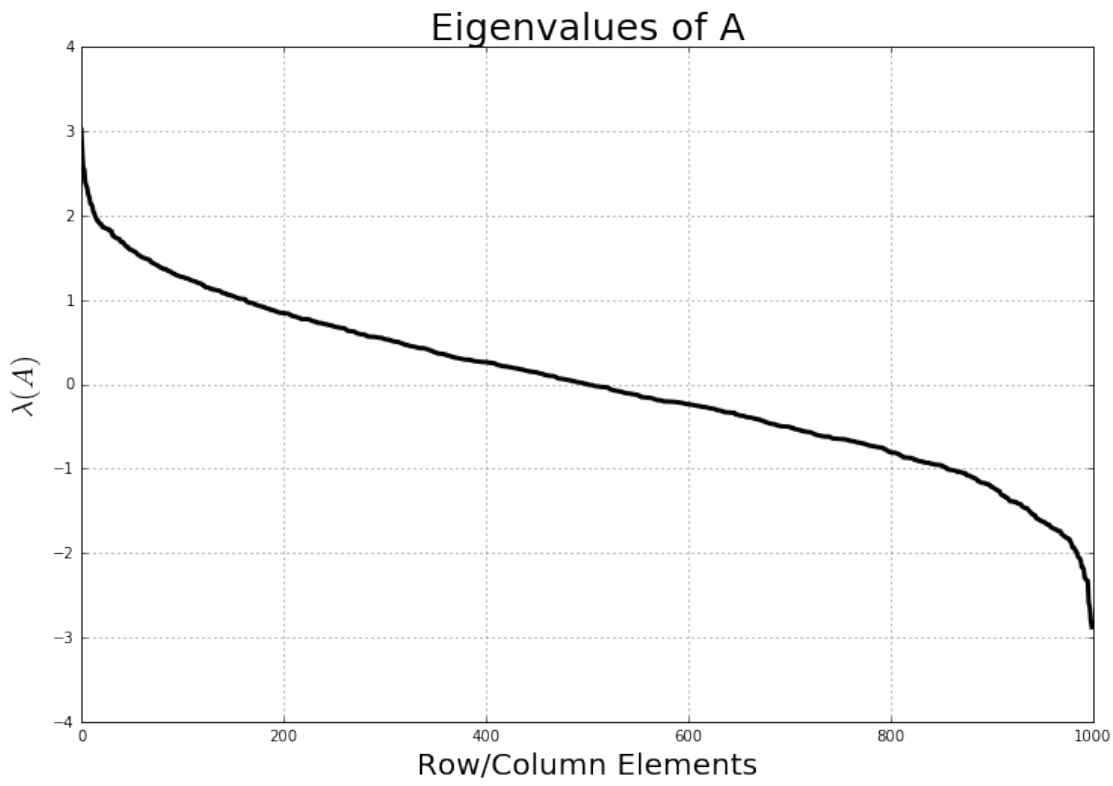


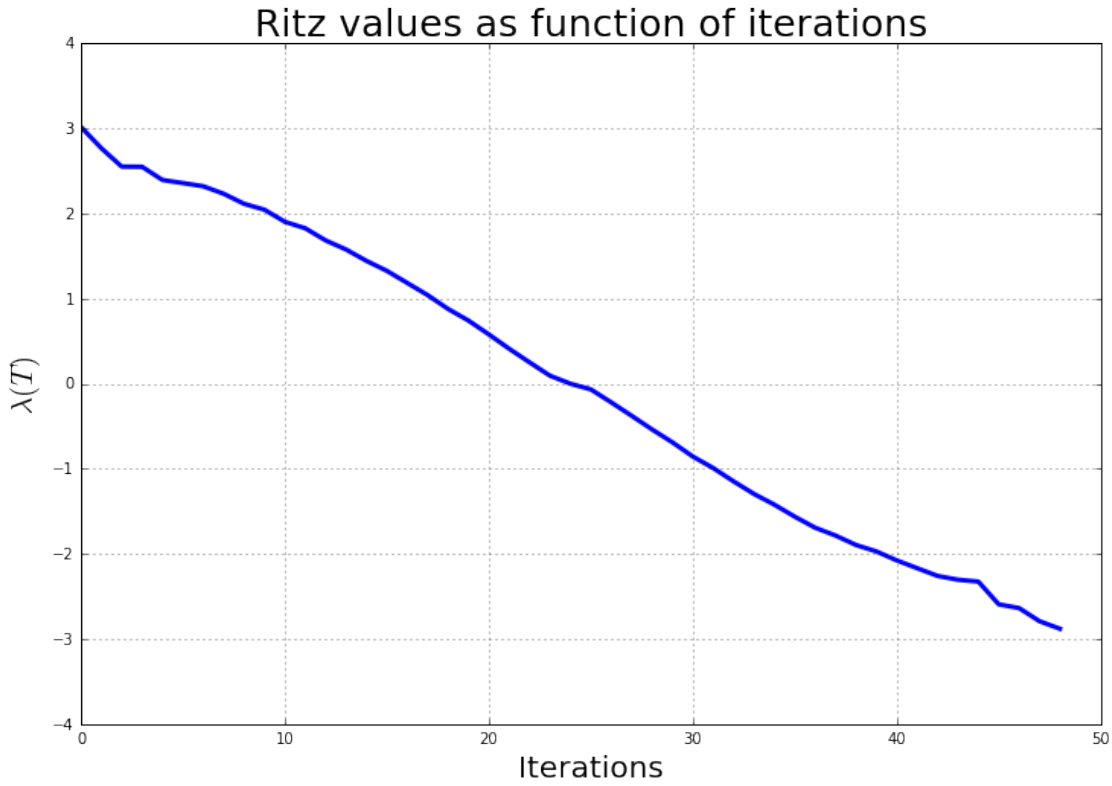

Ritz values as function of iterations
$\lambda(T)$
Iterations
4
3
2
1
0
−1
−2
−3
−4
0
5
10
15
20
25
30

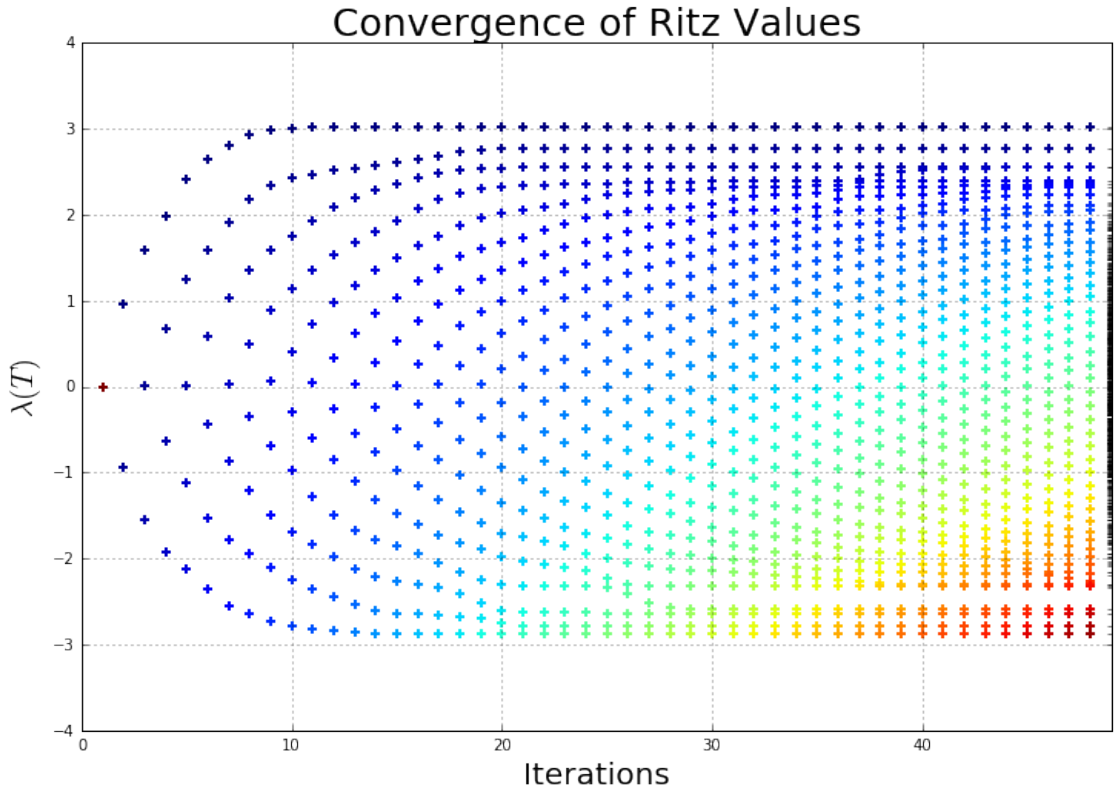

Convergence of Ritz Values
$\lambda(T)$
Iterations
4
3
2
1
0
−1
−2
−3
−4
0
5
10
15
20
25

```
In [28]: A = A
         m3 = 49
         Lanczos_demmel(A,v1,m=m3,n=n)
```

```
Number of iterations: 49
Eigenvalues of T: [ 3.01866708  2.768157    2.5516238   2.5497154   2.39418134  2.35804856
  2.32061749  2.23233123  2.11442088  2.04490125  1.90272682  1.82685468
  1.68410855  1.57718196  1.44368953  1.32648508  1.18414085  1.04144156
  0.87894304  0.74217501  0.57978553  0.41032495  0.2506854   0.09401502
  0.         -0.06656599 -0.21883795 -0.37647587 -0.53681951 -0.68874965
 -0.85538556 -0.99095496 -1.14776699 -1.29526946 -1.42150496 -1.56261544
 -1.69231778 -1.78449209 -1.89522848 -1.97098373 -2.07638733 -2.16783834
 -2.25863401 -2.30361198 -2.32558644 -2.59278381 -2.63721861 -2.79108077
 -2.87975182]
Greatest 4 Eigenvalues of T: [ 3.01866708  2.768157    2.5516238   2.5497154 ]
Greatest 4 Eigenvalues of A: [ 3.01866708  2.768157    2.5516242   2.54973605]
Global Error for the greatest 4 Eigenvalues: [  7.35571690e-16   1.60427754e-16   1.55609883e-
Smallest 4 Eigenvalues of T: [-2.59278381 -2.63721861 -2.79108077 -2.87975182]
Smallest 4 Eigenvalues of A: [-2.59278381 -2.63721861 -2.79108077 -2.87975182]
Global Error for the smallest 4 Eigenvalues: [  4.96755113e-11   1.14187308e-12   4.93241388e-
```




Ritz values as function of iterations
$\lambda(T)$
Iterations
4
3
2
1
0
−1
−2
−3
−4
0
10
20
30
40
50


Convergence of Ritz Values
$\lambda(T)$
Iterations
4
3
2
1
0
−1
−2
−3
−4
0
10
20
30
40

Principal Component Analysis (PCA) is a linear transformation technique whose main goal to find patterns in the data, detecting the correlation among variables. In a nutshell, we want to find the directions (principal components, i.e. eigenvectors) that maximize the variance of high-dimensional data and project it onto a smaller dimensional dimensional subspace, while retaining the most valuable information. Steps to follow:

- Standardize data
- Compute eigenvalues and eigenvectors of covariance matrix
- Sort eigenvalues in descending order and choose the k largest eigenvalues
- Construct Projection Matrix
- Build new lower-dimensional dataset

Making use of Python's *pandas* library, we work with the "Breast Cancer Wisconsin (Diagnostic) Database" provided by the University of California, Irvine *here*

```
In [2]: from sklearn.datasets import load_breast_cancer
        cancer = load_breast_cancer()
        from pandas import DataFrame, read_csv
        import pandas as pd

In [3]: print cancer.DESCR

Breast Cancer Wisconsin (Diagnostic) Database
=============================================

Notes
-----
Data Set Characteristics:
    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the contour)
        - symmetry
        - fractal dimension ("coastline approximation" - 1)
```

The mean, standard error, and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For instance, field 3 is Mean Radius, field 13 is Radius SE, field 23 is Worst Radius.

- class:
  - WDBC-Malignant
  - WDBC-Benign

:Summary Statistics:

	Min	Max
radius (mean):	6.981	28.11
texture (mean):	9.71	39.28
perimeter (mean):	43.79	188.5
area (mean):	143.5	2501.0
smoothness (mean):	0.053	0.163
compactness (mean):	0.019	0.345
concavity (mean):	0.0	0.427
concave points (mean):	0.0	0.201
symmetry (mean):	0.106	0.304
fractal dimension (mean):	0.05	0.097
radius (standard error):	0.112	2.873
texture (standard error):	0.36	4.885
perimeter (standard error):	0.757	21.98
area (standard error):	6.802	542.2
smoothness (standard error):	0.002	0.031
compactness (standard error):	0.002	0.135
concavity (standard error):	0.0	0.396
concave points (standard error):	0.0	0.053
symmetry (standard error):	0.008	0.079
fractal dimension (standard error):	0.001	0.03
radius (worst):	7.93	36.04
texture (worst):	12.02	49.54
perimeter (worst):	50.41	251.2
area (worst):	185.2	4254.0
smoothness (worst):	0.071	0.223
compactness (worst):	0.027	1.058
concavity (worst):	0.0	1.252
concave points (worst):	0.0	0.291
symmetry (worst):	0.156	0.664
fractal dimension (worst):	0.055	0.208

:Missing Attribute Values: None

:Class Distribution: 212 - Malignant, 357 - Benign

:Creator: Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian

:Donor: Nick Street

:Date: November, 1995

This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.
https://goo.gl/U2Uwz2

Features are computed from a digitized image of a fine needle
aspirate (FNA) of a breast mass. They describe
characteristics of the cell nuclei present in the image.

Separating plane described above was obtained using
Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree
Construction Via Linear Programming." Proceedings of the 4th
Midwest Artificial Intelligence and Cognitive Science Society,
pp. 97-101, 1992], a classification method which uses linear
programming to construct a decision tree. Relevant features
were selected using an exhaustive search in the space of 1-4
features and 1-3 separating planes.

The actual linear program used to obtain the separating plane
in the 3-dimensional space is that described in:
[K. P. Bennett and O. L. Mangasarian: "Robust Linear
Programming Discrimination of Two Linearly Inseparable Sets",
Optimization Methods and Software 1, 1992, 23-34].

This database is also available through the UW CS ftp server:

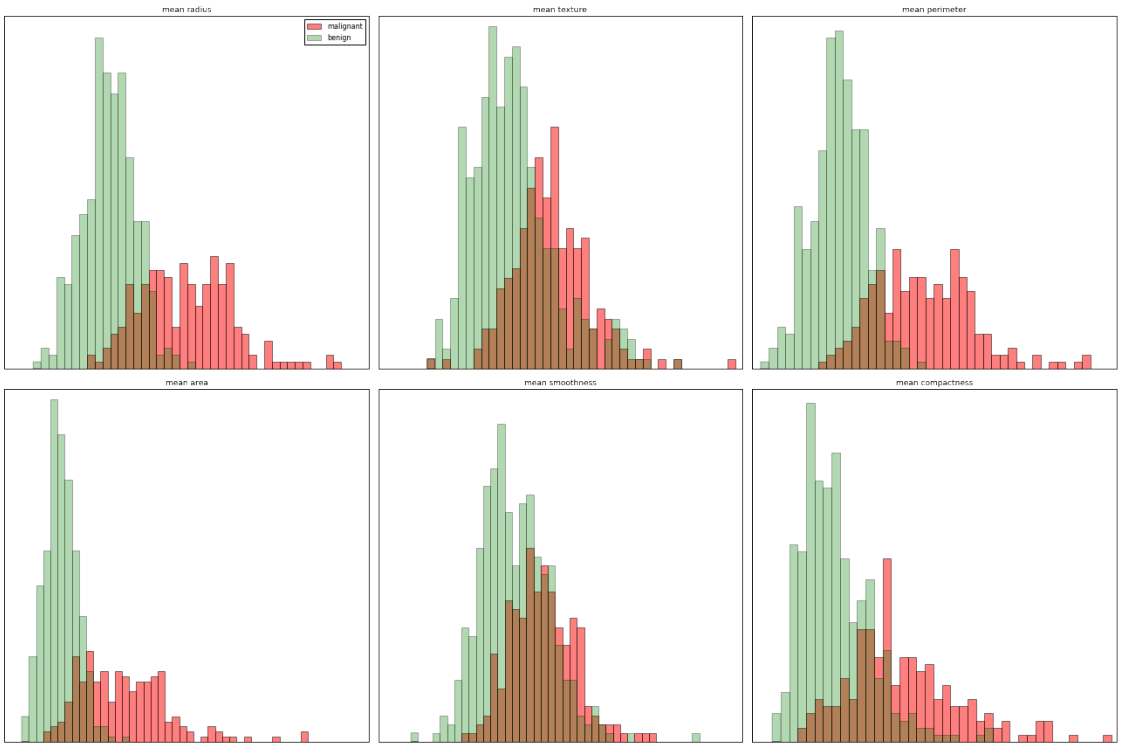
ftp ftp.cs.wisc.edu
cd math-prog/cpo-dataset/machine-learn/WDBC/

References

- W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on Electronic Imaging: Science and Technology, volume 1905, pages 861-870, San Jose, CA, 1993.
- O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and prognosis via linear programming. Operations Research, 43(4), pages 570-577, July-August 1995.
- W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994) 163-171.

To see what features play the bigger role in descerning from the benign to the malignant classes, we plot some histograms. If two histograms are separated based on features, then we can say it is a feature that helps us discerning the two classes.

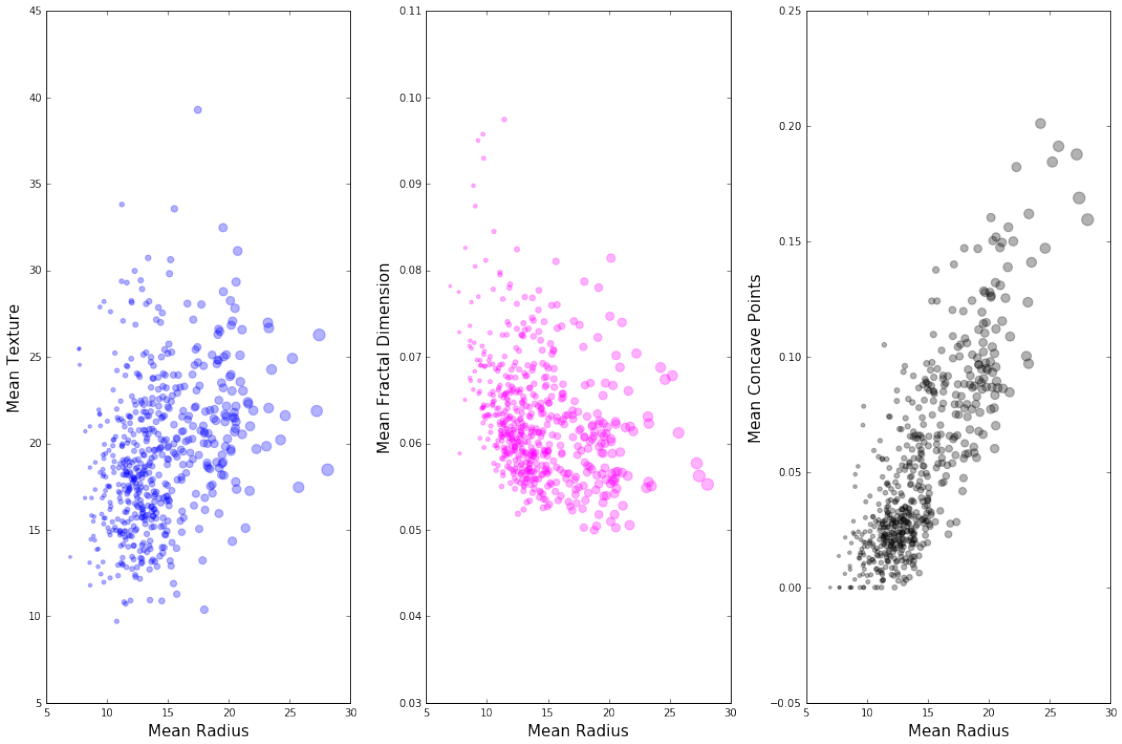
```
In [11]: fig,axes =plt.subplots(2,3, figsize=(18, 12)) ##code based upon //towardsdatascience.
         malignant=cancer.data[cancer.target==0]
         benign=cancer.data[cancer.target==1]
         ax=axes.ravel()
         for i in range(6):
             _,bins=np.histogram(cancer.data[:,i],bins=40) #every class has 40 bins
             ax[i].hist(malignant[:,i],bins=bins,color='r',alpha=.5)# red color for malignant
             ax[i].hist(benign[:,i],bins=bins,color='g',alpha=0.3)
             ax[i].set_title(cancer.feature_names[i],fontsize=9)
             ax[i].axes.get_xaxis().set_visible(False)
             ax[i].set_yticks(())
         ax[0].legend(['malignant','benign'],loc='best',fontsize=8)
         plt.tight_layout()
         plt.show()
```



Ok, we can try to see if there is any correlation among the variables

```
In [3]: cancer_df=pd.DataFrame(cancer.data,columns=cancer.feature_names)
        X = cancer_df
```

```
figure(figsize=(15,10))
plt.subplot(1,3,1)
plt.scatter(X['mean radius'], X['mean texture'], s=X['mean area']*0.05, color='blue', l
plt.xlabel('Mean Radius',fontsize=15)
plt.ylabel('Mean Texture',fontsize=15)
plt.subplot(1,3,2)
plt.scatter(X['mean radius'], X['mean fractal dimension'], s=X['mean area']*0.05, colo
plt.xlabel('Mean Radius',fontsize=15)
plt.ylabel('Mean Fractal Dimension',fontsize=15)
plt.subplot(1,3,3)
plt.scatter(X['mean radius'], X['mean concave points'], s=X['mean area']*0.05, color='
plt.xlabel('Mean Radius',fontsize=15)
plt.ylabel('Mean Concave Points',fontsize=15)
plt.tight_layout()
plt.show()
```



Now, we standardize the data using Python's *StandardScaler* and $fit - transform$

```
In [4]: from sklearn.preprocessing import StandardScaler
        X_std = StandardScaler().fit_transform(X)
        X_std.shape
```

```
Out[4]: (569L, 30L)
```

Now our data has been transformed onto unit scale (i.e. mean=0 and standard deviation=1).

So, we can get to the core of PCA: computing eigenvalues and eigenvectors of the covariance matrix. The covariance matrix is a $d \times d$ matrix (d being the number of columns of the original dataset), where each elements represents the covariance between two features. The eigenvectors will give the directions of maximum variance, while the eigenvalues will give us the magnitude. First things first, we compute the covariance matrix:

$$\Sigma = \frac{1}{n-1}(\mathbf{X} - \bar{\mathbf{x}})^T(\mathbf{X} - \bar{\mathbf{x}}),$$

where $\bar{\mathbf{x}} = \sum_{i=1}^{n} x_i$ is the mean vector, which is a d-dimensional vector whose values represents the sample mean of a featured column of the dataset. Thankfully, in Python, we can sweep all of this under the big rug of the buit-in numpy function $cov()$ applied to the transpose of our standardized dataset:

```
In [5]: cov_mat = np.cov(X_std.T); cov_mat.shape

Out[5]: (30L, 30L)
```

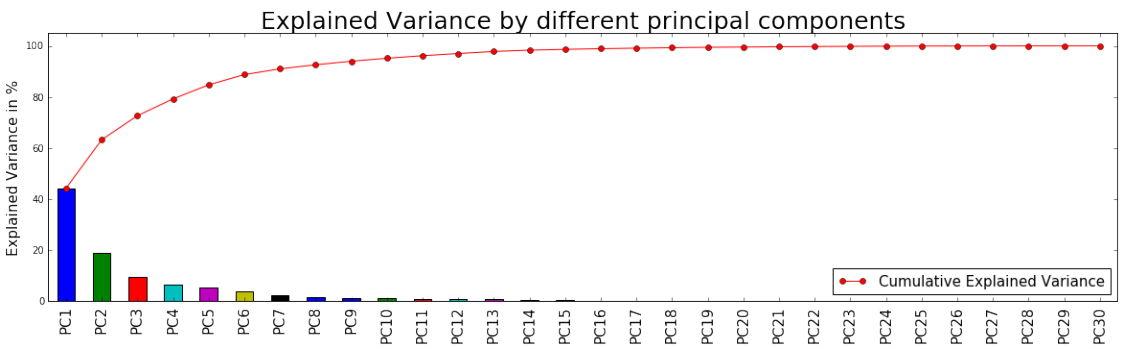
Finally, we can perform the eigendecomposition of our covariance matrix to get its eigenvalues and eigenvectors (i.e. directions of maximum variance and their magnitudes). In fact, the eigenvalues of our covariance matrix will tell us which feature of the original space bears the most information about the distribution of the data, while the eigenvalues with the lower magnitude bear little information. Thus, we compute the eigenpairs and sort them in descending order:

```
In [6]: eig_vals, eig_vecs = np.LA.eig(cov_mat)
        # Make a list w/ (eigenvalue, eigenvector)
        eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in range(len(eig_vals))]

        # Sort the (eigenvalue, eigenvector) in descending order
        eig_pairs.sort() #sort in ascending order
        eig_pairs.reverse() #reverse previous sorting

        #Check for descending order
        print('Eigenvalues in descending order:')
        for i in eig_pairs:
            print(i[0])

Eigenvalues in descending order:
13.3049907944
5.70137460373
2.82291015501
1.98412751773
1.65163324233
1.2094822398
0.676408881701
0.47745625469
0.417628782108
0.351310874882
0.294433153491
0.261621161366
```

```
0.241782421328
0.157286149218
0.0943006956011
0.0800034044774
0.0595036135304
0.0527114222101
0.049564700213
0.0312142605531
0.0300256630904
0.0274877113389
0.0243836913546
0.0180867939843
0.0155085271344
0.00819203711761
0.00691261257918
0.0015921360012
0.000750121412719
0.000133279056664
```

From here, to determine which principal components (eigenvectors) we are going to pick for our new featured subspace, we compute the so-called "explained variance", which tell us how much variance can be attributed to a given eigenvector.

```
In [9]: tot = sum(eig_vals)
        var_exp = [(i / tot)*100 for i in sorted(eig_vals, reverse=True)]
        var_exp_list = []
        cum_var_exp = np.cumsum(var_exp)
        figure(num=1, figsize=(20, 5));
        x=[i for i in range(1,30)]
        var_series = pd.Series(var_exp)
        x_labels = ['PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6','PC7', 'PC8', 'PC9', 'PC10',
                    'PC11', 'PC12', 'PC13','PC14','PC15','PC16','PC17','PC18','PC19','PC20',
                    'PC21','PC22','PC23','PC24','PC25','PC26','PC27','PC28','PC29','PC30']

        ax = var_series.plot(kind='bar');
        ax.set_title('Explained Variance by different principal components',fontsize=25);
        ax.set_ylabel('Explained Variance in %',fontsize=15);
        ax.set_xticklabels(x_labels,fontsize=15);
        ax.plot(cum_var_exp,'ro-');
        ax.set_ylim([0,105]);
        ax.legend(['Cumulative Explained Variance'],loc=0,fontsize=15);
```

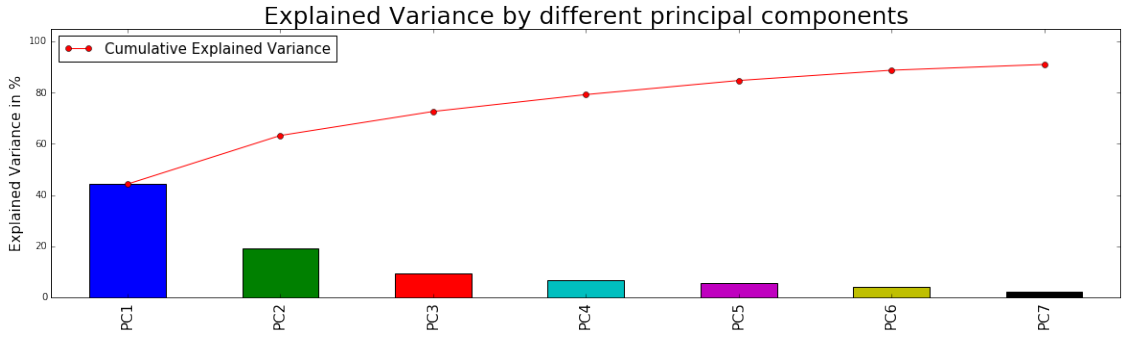


We now print the list containing our explained variance data:

```
In [10]: var_exp_list.append(var_exp)
         var_exp_list[0]

Out[10]: [44.272025607526352,
          18.971182044033092,
          9.3931632574313895,
          6.6021349154701339,
          5.4957684923462651,
          4.0245220398833546,
          2.2507337129825089,
          1.588723800021324,
          1.3896493745591108,
          1.168978189413151,
          0.97971898759801546,
          0.87053790073788329,
          0.80452498719673327,
          0.52336574549263604,
          0.31378321676273907,
          0.26620933651523304,
          0.19799679253242877,
          0.1753959450226367,
          0.16492530592251614,
          0.1038646748338712,
          0.099909646370025348,
          0.091464675105434559,
          0.081136125889910246,
          0.06018335666716744,
          0.051604237916519108,
          0.027258799547749436,
          0.023001546250596423,
          0.0052977929038085124,
          0.0024960103246888677,
          0.00044348274273714813]
```

```
In [12]: New_var = [i for i in var_exp_list[0] if i >= 2.]
         Svar = np.sum(New_var)
         L = len(New_var)
         print("New Variance Data: " + np.str(New_var))
         print("Number of elements = " + np.str(L))
         print("Sum of New Variance = " + np.str(Svar))

New Variance Data: [44.272025607526352, 18.971182044033092, 9.3931632574313895, 6.602134915470
Number of elements = 7
Sum of New Variance = 91.0095300697
```

```
In [13]: cum_newvar_exp = np.cumsum(New_var)
         figure(num=2, figsize=(20, 5));
         x=[i for i in range(1,7)]
         var_series = pd.Series(New_var)
         x_labels = ['PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6','PC7']

         ax = var_series.plot(kind='bar');
         ax.set_title('Explained Variance by different principal components',fontsize=25);
         ax.set_ylabel('Explained Variance in %',fontsize=15);
         ax.set_xticklabels(x_labels,fontsize=15);
         ax.plot(cum_newvar_exp,'ro-');
         ax.set_ylim([0,105]);
         ax.legend(['Cumulative Explained Variance'],loc=2,fontsize=15);
```



Major drawback: after dimensionality reduction, there usually isn't a particular meaning assigned to each principal component. The new components are just the main dimensions of variation. So it's difficult to interpret the new axes as they are some complex mixture of the original features. That being said, we see that the first 7 sorted PCs retain more than 91% of the information of the original subspace. So, we can build a projection matrix to be used to transform the original dataset into our new lower-dimensional subspace. Our matrix is going to be 30×7.

```
In [14]: matrix_w = np.hstack((eig_pairs[0][1].reshape(30,1),
                               eig_pairs[1][1].reshape(30,1),
                               eig_pairs[2][1].reshape(30,1),
```

```
                                    eig_pairs[3][1].reshape(30,1),
                                    eig_pairs[4][1].reshape(30,1),
                                    eig_pairs[5][1].reshape(30,1),
                                    eig_pairs[6][1].reshape(30,1)))

        matrix_w.shape
```

```
Out[14]: (30L, 7L)
```

Now, we can finally redure our original dataset into the new lower-dimensional dataset, which will be 569×7:

$$\mathbf{Y} = \mathbf{X} \times \mathbf{W}$$

```
In [15]: Y = X_std.dot(matrix_w)
         Y.shape
```

```
Out[15]: (569L, 7L)
```

From the previous analysis, we know that this subspace retains more than 91% of the information of the original subspace. So we have succesfully reduced our original space, without any significant loss of information.

Thank you for listening!