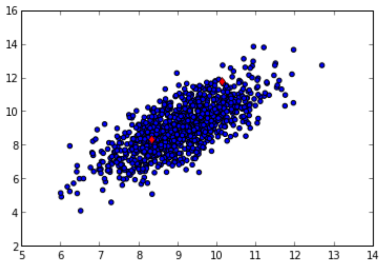
Temel Bileşen Analizi (Principal Component Analysis -PCA-)

PCA yontemi boyut azaltan yontemlerden biri, takip edilmeden (unsupervised) isleyebilir. Ana fikir veri noktalarinin izdusumunun yapilacagi yonler bulmaktir ki bu yonler baglaminda (izdusum sonrasi) noktalarin arasindaki sayisal varyans (empirical variance) en fazla olsun, yani noktalar grafik baglaminda dusunursek en "yayilmis" sekilde bulunsunlar. Boylece birbirinden daha uzaklasan noktalarin mesela daha rahat kumelenebilecegini umabiliriz. Bir diger amac, hangi degiskenlerin varyansinin daha fazla oldugunun gorulmesi uzerine, o degiskenlerin daha onemli olabileceginin anlasilmasi. Ornek olarak alttaki grafige bakalim,

```
from pandas import *
data = read_csv("testSet.txt",sep="\t",header=None)
print data[:10]
```

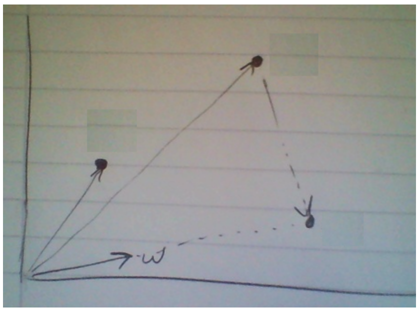
```
           0          1
0  10.235186  11.321997
1  10.122339  11.810993
2   9.190236   8.904943
3   9.306371   9.847394
4   8.330131   8.340352
5  10.152785  10.123532
6  10.408540  10.821986
7   9.003615  10.039206
8   9.534872  10.096991
9   9.498181  10.825446
```

```
plt.scatter(data.ix[:,0],data.ix[:,1])
plt.plot(data.ix[1,0],data.ix[1,1],'rd')
plt.plot(data.ix[4,0],data.ix[4,1],'rd')
plt.savefig('pca_1.png')
```



PCA ile yapmaya calistigimiz oyle bir yon bulmak ki, x veri noktalarinin tamaminin o yone izdusumu yapilinca sonuc olacak, "izdusumu yapilmis" z'nin varyansi en buyuk olsun. Bu bir maksimizasyon problemidir. Fakat ondan once x nedir, z nedir bunlara yakindan bakalim.

Veri x ile tum veri noktalari kastedilir, fakat PCA probleminde genellikle bir "vektorun digeri uzerine" yapilan izdusumu, "daha optimal bir w yonu bulma", ve "o yone dogru izdusum yapmak" kelimeleri kullanilir. Demek ki veri noktalarini bir vektor olarak gormeliyiz. Eger ustte kirmizi ile isaretlenen iki noktayi alirsak (bu noktalar verideki 1. ve 4. siradaki noktalar),



gibi bir goruntuden bahsediyoruz. Hayali bir w kullandik, ve noktalardan biri veri noktasi, w uzerine izdusum yapilarak yeni bir vektoru / noktayi ortaya cikartiliyor. Genel olarak ifade edersek, bir nokta icin

$$z_i = x_i^T w = x_i \cdot w$$

Yapmaya calistigimiz sayisal varyansi maksimize etmek demistik. Bu arada verinin hangi dagilimdan geldigini soylemedik, "her veri noktasi birbirinden ayri, bagimsiz ama ayni bir dagilimdandir" bile demedik, x bir rasgele degiskendir beyani yapmadik (x veri noktalarini tutan bir sey sadece). Sadece sayisal varyans ile is yapacagiz. Sayisal varyans,

$$\frac{1}{n}\sum_i (x_i \cdot w)^2$$

Toplama islemi yerine soyle dusunelim, tum x_i noktalarini istifleyip bir x matrisi haline getirelim, o zaman xw ile bir yansitma yapabiliriz, bu yansitma sonucu bir vektordur. Bu tek vektorun karesini almak demek onun devrigini alip kendisi ile carpmak demektir, yani

$$= \frac{1}{n}(xw)^T(xw) = \frac{1}{n}w^T x^T x w$$

$$= w^T \frac{x^T x}{n} w$$

$x^T x / n$ sayisal kovaryanstir (empirical covariance). Ona Σ diyelim.

$$= w^T \Sigma w$$

Ustteki sonuclarin boyutlari $1 \times N \cdot N \times N \cdot N \times 1 = 1 \times 1$.

Yani tek boyutlu skalar degerler elde ettik. Yani w yonundeki izdusum bize tek boyutlu bir cizgi verecektir. Bu sonuc aslinda cok sasirtici olmasa gerek, tum veri noktalarini alip, baslangici basnokta 0,0 (origin) noktasinda olan vektorlere cevirip ayni yone isaret edecek sekilde duzenliyoruz, bu vektorleri tekrar nokta olarak dusunursek, tabii ki ayni yonu gosteriyorlar, bilahere ayni cizgi uzerindeki noktalara donusuyorlar. Ayni cizgi uzerinde olmak ne demek? Tek boyuta inmis olmak demek.

Ufak bir sorun $w^T \Sigma w$'i surekli daha buyuk w_1'lerle sonsuz kadar buyutebilirsiniz. Bize ek bir kisitlama sarti daha lazim, bu sart $||w|| = 1$ olabilir, yani w'nin norm'u 1'den daha buyuk olmasin. Boylece optimizasyon w'yi surekli buyute buyute maksimizasyon yapmayacak, sadece yon bulmak ile ilgilenecek, iyi, zaten biz w'nin yonu ile ilgileniyoruz. Aradigimiz ifadeyi yazalim, ve ek siniri Lagrange ifadesi olarak ekleyelim, ve yeni bir L ortaya cikartalim,

$$L(w, \lambda) = w^T \Sigma w - \lambda(w^T w - 1)$$

Niye eksiden sonraki terim o sekilde eklendi? O terim oyle sekilde secildi ki, $\partial L/\partial\lambda = 0$ alininca $w^T w = 1$ geri gelsin / ortaya ciksin [2, sf 340].

Bu Lagrange'in dahice bulusu. Bu kontrol edilebilir, λ 'ya gore turev alirken w_1 sabit olarak yokolur, parantez icindeki ifadeler kalir ve sifira esitlenince orijinal kisitlama ifadesi geri gelir. Simdi

$$\max_w L(w, \lambda)$$

icin turevi w'e gore alirsak, ve sifira esitlersek,

$$\frac{\partial L}{\partial w} = 2w\Sigma - 2\lambda w_{=}0$$

$$2w\Sigma = 2\lambda w$$

$$\Sigma w = \lambda w$$

Ustteki ifade ozdeger, ozvektor ana formulune benzemiyor mu? Evet. Eger w, Σ'nin ozvektoru ise ve esitligin sagindaki λ ona tekabul eden ozdeger ise, bu esitlik dogru olacaktir.

Peki hangi ozdeger / ozvektor maksimal degeri verir? Unutmayalim, maksimize etmeye calistigimiz sey $w^T \Sigma w$ idi

Eger $\Sigma w = \lambda w$ yerine koyarsak

$$w^T \lambda w = \lambda w^T w = \lambda$$

Cunku $w_1^T w$'nin 1 olacagi sartini koymustuk. Neyse, maksimize etmeye calistigimiz deger λ cikti, o zaman en buyuk λ kullanirsak, en maksimal varyansi elde ederiz, bu da en buyuk ozdegerin ta kendisidir. Demek ki izdusum yapilacak "yon" kovaryans Σ'nin en buyuk ozdegerine tekabul eden ozvektor olarak secilirse, temel bilesenlerden en onemlisini hemen bulmus olacagiz. Ikinci, ucuncu en buyuk ozdegerin ozvektorleri ise diger daha az onemli yonleri bulacaklar.

Σ matrisi $n \times n$ boyutunda bir matris, bu sebeple n tane ozvektoru olacak. Her kovaryans matrisi simetriktir, o zaman lineer cebir bize der ki ozvektorler birbirine dikgen (orthogonal) olmali. Yine Σ bri kovaryans matrisi oldugu icin bir pozitif matris olmali, yani herhangi bir x icin $x\Sigma x \geqslant 0$. Bu bize tum ozvektorlerin $\geqslant 0$ olmasi gerektigini soyluyor.

Ustteki ozvektorler verinin temel bilesenleridir (principal components).

Ornek

Simdi tum bunlari bir ornek uzerinde gorelim. Iki boyutlu ornek veriyi ustte yuklemistik. Simdi veriyi "sifirda ortalayacagiz" yani her kolon icin o kolonun ortalama degerini tum kolondan cikartacagiz. PCA ile islem yaparken tum degerlerin sifir merkezli olmasi gerekiyor.

Daha sonra ozdegerlerini, vektorlerini hesaplayabilmek icin verinin kovaryansini hesaplayacagiz.

```
import numpy.linalg as lin
from pandas import *
data = read_csv("testSet.txt",sep="\t",header=None)
print data.shape
print data[:10]

means = data.mean()
meanless_data = data - means
cov_mat = np.cov(meanless_data, rowvar=0)
print cov_mat.shape
eigs,eigv = lin.eig(cov_mat)
eig_ind = np.argsort(eigs)
print eig_ind
```

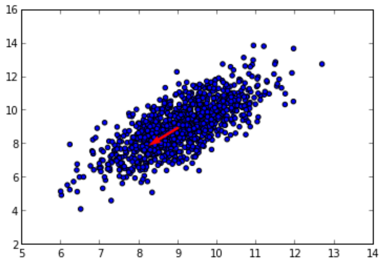
```
(1000, 2)
           0          1
0  10.235186  11.321997
1  10.122339  11.810993
2   9.190236   8.904943
3   9.306371   9.847394
4   8.330131   8.340352
5  10.152785  10.123532
6  10.408540  10.821986
7   9.003615  10.039206
8   9.534872  10.096991
9   9.498181  10.825446
(2, 2)
[0 1]
```

```
print eigs[1],eigv[:,1].T
print eigs[0],eigv[:,0].T
```

```
2.89713495618 [-0.52045195 -0.85389096]
0.366513708669 [-0.85389096  0.52045195]
```

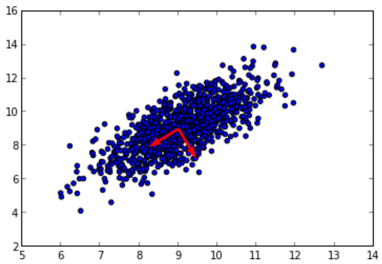
En buyuk olan yonu quiver komutunu kullanarak orijinal veri seti uzerinde gosterelim,

```
plt.scatter(data.ix[:,0],data.ix[:,1])
# merkez 9,9, tahminen secildi
plt.quiver(9,9,eigv[1,1],eigv[0,1],scale=10,color='r')
plt.savefig('pca_2.png')
```



Goruldugu gibi bu yon hakikaten dagilimin, veri noktalarinin en cok yayilmis oldugu yon. Demek ki PCA yontemi dogru sonucu buldu. Her iki yonu de cizersek,

```
plt.scatter(data.ix[:,0],data.ix[:,1])
plt.quiver(9,9,eigv[1,0],eigv[0,0],scale=10,color='r')
plt.quiver(9,9,eigv[1,1],eigv[0,1],scale=10,color='r')
plt.savefig('pca_3.png')
```



Bu ikinci yon birinciye dik olmaliydi, ve o da bulundu. Aslinda iki boyut olunca baska secenek kalmiyor, 1. yon sonrasi ikincisi baska bir sey olamazdi, fakat cok daha yuksek boyutlarda en cok yayilimin oldugu ikinci yon de dogru sekilde geri getirilecekti.

SVD ile PCA Hesaplamak

PCA bolumunde anlatilan yontem temel bilesenlerin hesabinda ozdegerler ve ozvektorler kullandi. Alternatif bir yontem Tekil Deger Ayristirma (Singular Value Decomposition -SVD-) uzerinden bu hesabi yapmaktir. SVD icin Lineer Cebir Ders 29'a bakabilirsiniz. Peki ne zaman klasik PCA ne zaman SVD uzerinden

PCA kullanmali? Bir cevap belki mevcut kutuphanelerde SVD kodlamasinin daha iyi olmasi, ayristirmanin ozvektor / deger hesabindan daha hizli isleyebilmesi [6].

Ayrica birazdan gorecegimiz gibi SVD, kovaryans matrisi uzerinde degil, A'nin kendisi uzerinde isletilir, bu hem kovaryans hesaplama asamasini atlamamizi, hem de kovaryans hesabi sirasinda ortaya cikabilecek numerik puruzlerden korunmamizi saglar (cok ufak degerlerin kovaryans hesabini bozabilecegi literaturde bahsedilmektedir).

PCA ve SVD baglantisina gelelim:

Biliyoruz ki SVD bir matrisi su sekilde ayristirir

$$A = USV^T$$

U matrisi $n \times n$ dikgen (orthogonal), V ise $m \times m$ dikgen. S'in sadece kosegeni uzerinde degerler var ve bu σ_j degerleri A'nin tekil degerleri (singular values) olarak biliniyor.

Simdi A yerine AA^T koyalim, yani A'nin kovaryans matrisinin SVD ayristirmasini yapalim, acaba elimize ne gececek?

$$AA^T = (USV^T)(USV^T)^T$$

$$= (USV^T)(VS^TU^T)$$

$$= USS^TU^T$$

S bir kosegen matrisi, o zaman SS^T matrisi de kosegen, tek farkla kosegen uzerinde artik σ_j^2 degerleri var. Bu normal.

SS^T yerine Λ sembolunu kullanalim, ve denklemi iki taraftan (ve sagdan) U ile carparsak (unutmayalim U ortanormal bir matris ve $U^TU = I$),

$$AA^TU = U\Lambda U^TU$$

$$AA^TU = U\Lambda$$

Son ifadeye yakindan bakalim, U'nun tek bir kolonuna, u_k diyelim, odaklanacak olursak, ustteki ifadeden bu sadece kolona yonelik nasil bir esitlik cikartabilirdik? Soyle cikartabilirdik,

$$(AA^T)u_k = \sigma^2 u_k$$

Bu ifade tanidik geliyor mu? Ozdeger / ozvektor klasik yapisina eristik. Ustteki esitlik sadece ve sadece eger u_k, AA^T'nin ozvektoru ve σ^2 onun ozdegeri ise gecerlidir. Bu esitligi tum U kolonlari icin uygulayabilecegimize gore demek ki U'nun kolonlarinda AA^T'nin ozvektorleri vardir, ve AA^T'nin ozdegerleri A'nin tekil degerlerinin karesidir.

Bu muthis bir bulus. Demek ki AA^T'nin ozektorlerini hesaplamak icin A uzerinde SVD uygulayarak U'yu bulmak ise yarar, kovaryans matrisini hesaplamak gerekli degil (bir hesap yerine otekini koymus olduk A^TA carpimi yerine yerine AA^T hesabi). AA^T ozdegerleri uzerinde buyukluk karsilastirmasi icin ise A'nin tekil degerlerine bakmak yeterli!

Ornek

Ilk bolumdeki ornege donelim, ve ozvektorleri SVD uzerinden hesaplatalim.

```
U,s,Vt = svd(meanless_data.T,full_matrices=False)
print U
```

```
[[-0.52045195 -0.85389096]
 [-0.85389096  0.52045195]]
```

```
print np.dot(U.T,U)
```

```
[[  1.00000000e+00   3.70255042e-17]
 [  3.70255042e-17   1.00000000e+00]]
```

Goruldugu gibi ayni ozvektorleri bulduk.

New York Times Yazıları Analizi

Simdi daha ilginc bir ornege bakalim. Bir arastirmaci belli yillar arasindaki NY Times makalelerinde her yazida hangi kelimenin kac kere ciktiginin verisini toplamis [1,2,3], bu veri 4000 kusur kelime, her satir (yazi) icin bir boyut (kolon) olarak kaydedilmis. Bu veri nytimes.csv uzerinde ek bir normalize isleminden sonra, onun uzerinde boyut indirgeme yapabiliriz.

Veri setinde her yazi ayrica ek olarak sanat (arts) ve muzik (music) olarak etiketlenmis, ama biz PCA kullanarak bu etiketlere hic bakmadan, verinin boyutlarini azaltarak acaba verinin "ayrilabilir" hale indirgenip indirgenemedigine bakacagiz. Sonra etiketleri veri ustune koyup sonucun dogrulugunu kontrol edecegiz.

Bakmak derken veriyi (en onemli) iki boyuta indirgeyip sonucu grafikleyecegiz. Illa 2 olmasi gerekmez tabii, 10 boyuta indirgeyip (ki 4000 kusur boyuttan sonra bu hala muthis bir kazanim) geri kalanlar uzerinde mesela bir kumeleme algoritmasi kullanabilirdik.

Ana veriyi yukleyip birkac satirini ve kolonlarini gosterelim.

```
from pandas import *
import numpy.linalg as lin
nyt = read_csv ("nytimes.csv")
labels = nyt['class.labels']
print nyt.ix[:8,102:107]
```

```
   after  afternoon  afterward  again  against
0      1          0          0      0        0
1      1          1          0      0        0
2      1          0          0      1        2
3      3          0          0      0        0
4      0          1          0      0        0
5      0          0          0      1        2
6      7          0          0      0        1
7      0          0          0      0        0
8      0          0          0      0        0
```

Yuklemeyi yapip sadece etiketleri aldik ve onlari bir kenara koyduk. Simdi onemli bir normalizasyon islemi gerekiyor - ki bu isleme ters dokuman-frekans agirliklandirmasi (inverse document-frequency weighting -IDF-) ismi veriliyor - her dokumanda aşırı fazla ortaya cikan kelimelerin onemi ozellikle azaltiliyor, ki diger kelimelerin etkisi artabilsin.

IDF kodlamasi alttaki gibidir. Once `class.labels` kolonunu atariz. Sonra "herhangi bir deger iceren" her hucrenin 1 digerlerinin 0 olmasi icin kullanilan DataFrame uzerinde `astype(bools)` isletme numarasini kullaniriz, boylece asiri buyuk degerler bile sadece 1 olacaktir. Bazi diger islemler sonrasi her satiri kendi icinde tekrar normalize etmek icin o satirdaki tum degerlerin karesinin toplaminin karekokunu aliriz ve satirdaki tum degerler bu karekok ile bolunur. Buna oklitsel (euclidian) normalizasyon denebilir.

Not: Oklitsel norm alirken toplamin hemen ardindan cok ufak bir 1e-16 degeri eklememize dikkat cekelim, bunu toplamin sifir olma durumu icin yapiyoruz, ki sonra sifirla bolerken NaN sonucundan kacinalim.

```
nyt2 = nyt.drop('class.labels',axis=1)
freq = nyt2.astype(bool).sum(axis=0)
freq = freq.replace(0,1)
w = np.log(float(nyt2.shape[0])/freq)
nyt2 = nyt2.apply(lambda x: x*w,axis=1)
nyt2 = nyt2.apply(lambda x: x / np.sqrt(np.sum(np.square(x))+1e-16), axis=1)
#nyt2 = nyt2.div(nyt2.sum(axis=0), axis=1)
nyt2=nyt2.ix[:,1:] # ilk kolonu atladik
print nyt2.ix[:8,102:107]
```

```
   afterward     again   against       age  agent
0          0  0.000000  0.000000  0.051085      0
1          0  0.000000  0.000000  0.000000      0
2          0  0.021393  0.045869  0.000000      0
3          0  0.000000  0.000000  0.000000      0
4          0  0.000000  0.000000  0.000000      0
5          0  0.024476  0.052480  0.000000      0
6          0  0.000000  0.008536  0.000000      0
7          0  0.000000  0.000000  0.000000      0
```

```
8           0  0.000000  0.000000  0.000000      0
```

Not: Bir diger normalize metotu

```
import pandas as pd

df = pd.DataFrame([[1.,1.,np.nan],
                   [1.,2.,0.],
                   [1.,3.,np.nan]])
print df
print df.div(df.sum(axis=0), axis=1)
```

```
   0  1    2
0  1  1  NaN
1  1  2    0
2  1  3  NaN
          0         1    2
0  0.333333  0.166667  NaN
1  0.333333  0.333333  NaN
2  0.333333  0.500000  NaN
```

SVD yapalim

```
nyt3 = nyt2 - nyt2.mean(0)
u,s,v = lin.svd(nyt3.T,full_matrices=False)
print s[:10]
```

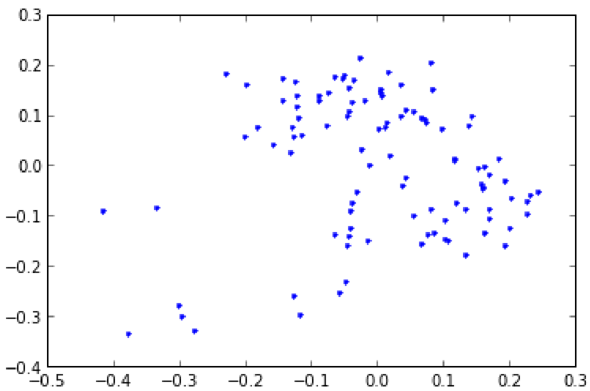
```
[ 1.41676764  1.37161893  1.31840061  1.24567955  1.20596873  1.18624932
  1.15118771  1.13820504  1.1138296   1.10424634]
```

```
print u.shape
```

```
(4430, 102)
```

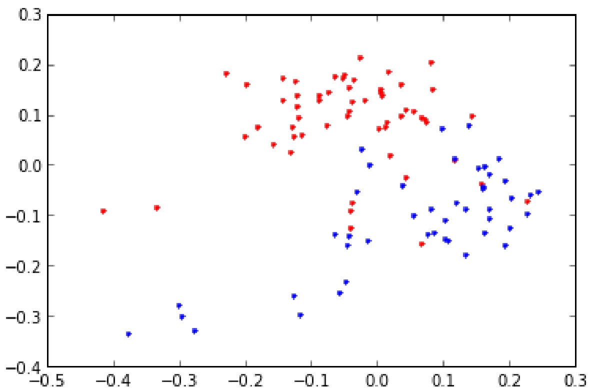
SVD'nin verdigi u icinden iki ozvektoru seciyoruz (en bastakiler, cunku Numpy SVD kodu bu ozvektorleri zaten siralanmis halde dondurur), ve veriyi bu yeni kordinata izdusumluyoruz.

```
proj = np.dot(nyt, u[:,:2])
proj.shape
plt.plot(proj[:,0],proj[:,1],'.')
plt.savefig('pca_4.png')
```



Simdi ayni veriyi bir de etiket bilgisini devreye sokarak cizdirelim. Sanat kirmizi muzik mavi olacak.

```
arts =proj[labels == 'art']
music =proj[labels == 'music']
plt.plot(arts[:,0],arts[:,1],'r.')
plt.plot(music[:,0],music[:,1],'b.')
plt.savefig('pca_5.png')
```



Goruldugu gibi veride ortaya cikan / ozvektorlerin kesfettigi dogal ayirim, hakikaten dogruymus.

Metotun ne yaptigina dikkat, bir suru boyutu bir kenara atmamiza ragmen geri kalan en onemli 2 boyut uzerinden net bir ayirim ortaya cikartabiliyoruz. Bu PCA yonteminin iyi bir is becerdigini gosteriyor, ve kelime sayilarinin makalelerin icerigi hakkinda ipucu icerdigini ispatliyor.

Kaynaklar

[1] Alpaydin, E., Introduction to Machine Learning, 2nd Edition

[2] Strang, G., Linear Algebra and Its Applications, 4th Edition

[3] `http://www.stat.columbia.edu/~fwood/Teaching/w4315/Spring2010/PCA/slides.pdf`

[4] Cosma Rohilla Shalizi, *Advanced Data Analysis from an Elementary Point of View*

[5] `http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2008T19`

[6] `http://www.stat.cmu.edu/~cshalizi/490/pca`

[7] `http://www.math.nyu.edu/faculty/goodman/teaching/RPME/notes/Section3.pdf`

[8] Lineer Cebir notlarimizda SVD turetilmesine bakinca ozdeger/vektor mantigina atif yapildigini gorebiliriz ve akla su gelebilir; "ozdeger / vektor rutini isletmekten kurtulalim dedik, SVD yapiyoruz, ama onun icinde de ozdeger/vektor

hesabi var". Fakat sunu belirtmek gerekir ki SVD numerik hesabini yapmanin tek yontemi ozdeger/vektor yontemi degildir. Mesela Numpy Linalg kutuphanesi icindeki SVD, LAPACK dgesdd rutinini kullanir ve bu rutin ic kodlamasinda QR, ve bir tur bol / istila et (divide and conquer) algoritmasi isletmektedir.