Procesamiento y Análisis de Datos Astronómicos

10.- Modelado de datos y determinación de parámetros III

R. Gil-Hutton

Marzo 2020

Actividades:

Práctica 8:

- Para dos de las variables de su archivo de datos estime la precisión en los parámetros para el tamaño de las muestras, ajuste modelos creados con generalizaciones multiparamétricas de su elección, grafique los errores y estime con una prueba de hipótesis si son aleatorios o no.

Práctica 9:

- Para los dos ajustes realizados en el práctico anterior calcule el intervalo de confianza proyectado y la varianza de cada uno de los parámetros del modelo.

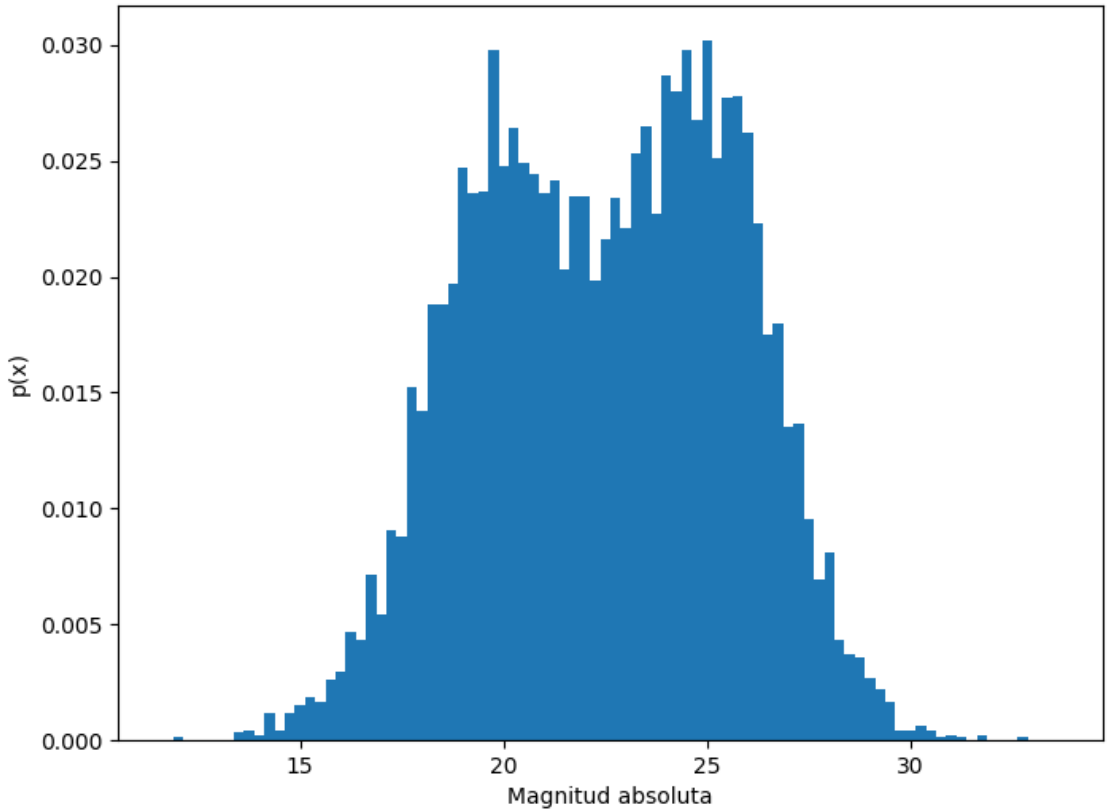
Práctica 8:

Vamos a tratar de ajustar una distribución bimodal usando una combinación de distribuciones normales a la muestra de magnitudes absolutas de objetos Apollo del archivo **apollo-aeih.dat**, utilizando un modelo no lineal. La muestra cuenta con 9239 elementos lo que me permite una precisión relativa en los estimadores del 1 %.

```
In [2]: apo=np.loadtxt('apollo-aeih.dat')

In [3]: np.min(apo[:,3]),np.max(apo[:,3])
Out[3]: (12.4, 33.24)

In [4]: his,lim=np.histogram(apo[:,3],bins=88,range=(12,34))

In [5]: his=his/np.sum(his)

In [6]: xx=lim[1:]-0.125
```

Práctica 8:

La distribución de magnitudes para los objetos Apollo es la siguiente:



Práctica 8:

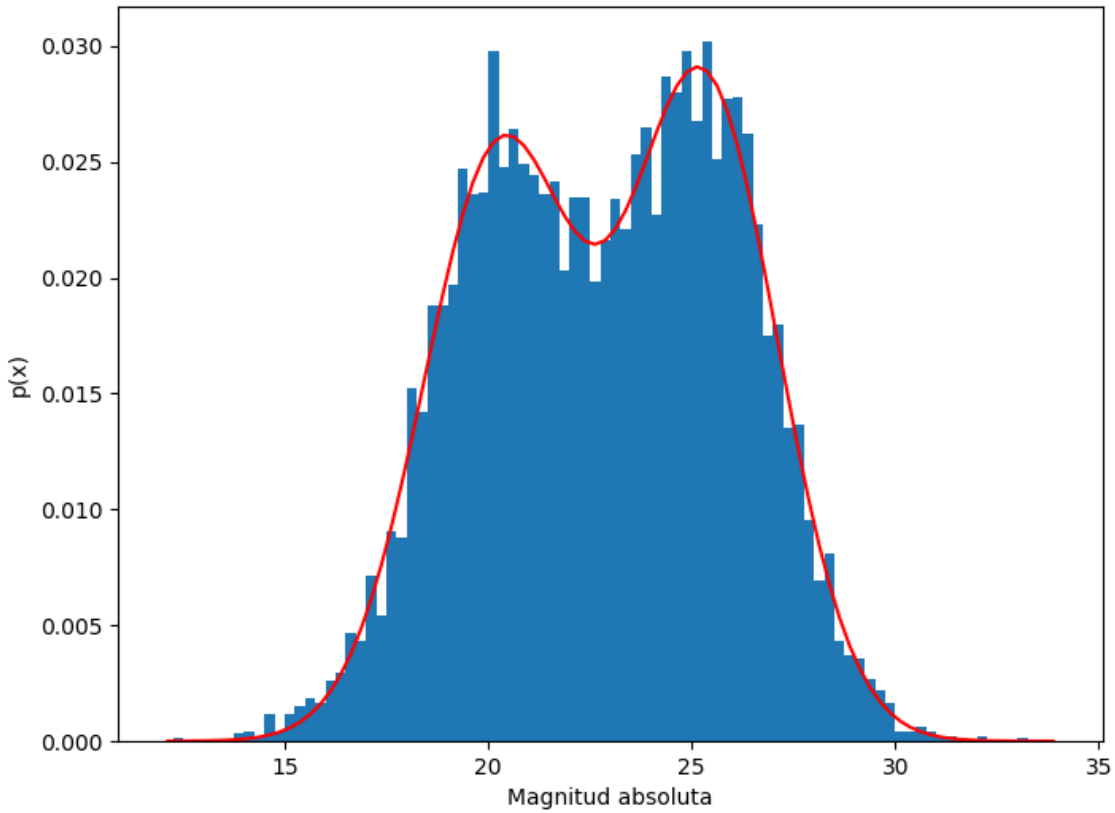
La función (arbitraria) usada para el ajuste es:

$$p(x) = \frac{k_1}{\sqrt{2\pi}\sigma_1} \exp\left(-\frac{(x-\mu_1)^2}{2\sigma_1^2}\right) + \frac{k_2}{\sqrt{2\pi}\sigma_2} \exp\left(-\frac{(x-\mu_2)^2}{2\sigma_2^2}\right)$$

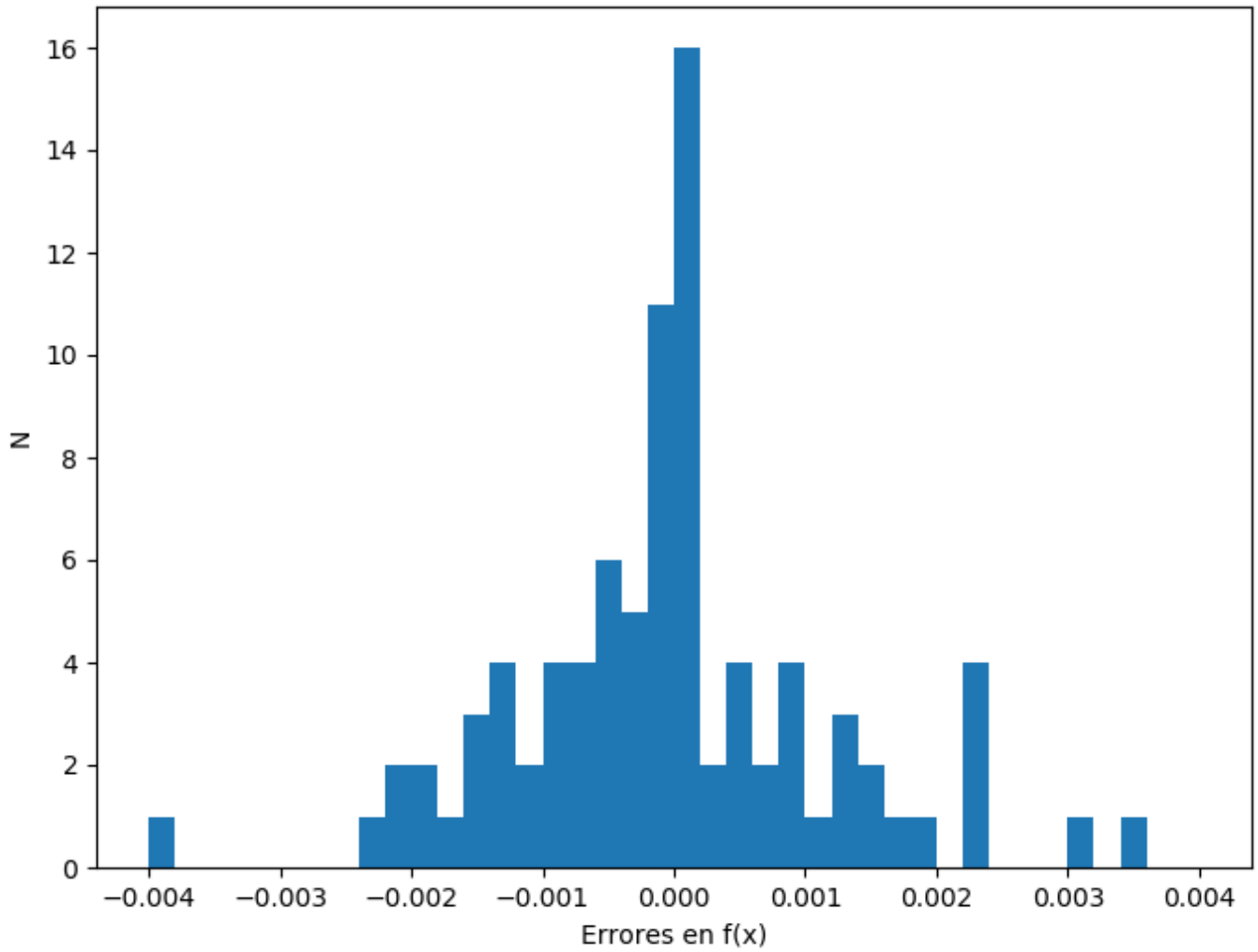
```
In [102]: def funcajus(par,xx,yy):
     ...:     """
     ...:     El array par contiene mu1,var1,mu2,var2,k1 y k2.
     ...:     Los arrays xx e yy tienen los valores centrales de los bines y su
valor en la distribucion, respectivamente.
     ...:
     ...:     La función devuelve las diferencias entre el ajuste y el valor obs
ervado en la distribución para cada bin.
     ...:     """
     ...:     zz=par[4]/np.sqrt(2.*np.pi)/np.sqrt(par[1])*np.e**(-(xx-par[0])**2
/2/par[1])+par[5]/np.sqrt(2.*np.pi)/np.sqrt(par[3])*np.e**(-(xx-par[2]
)**2/2/par[3])-yy
     ...:     return zz
     ...:

In [103]: import scipy.optimize as opt

In [104]: sol=opt.least_squares(funcajus,np.array([20,2,25,2,1.,1.]),method='lm'
     ...: ,ftol=1.e-12, gtol=1.e-12,xtol=1.e-12,args=(xx,his))
```

Práctica 8:

```
In [10]: par0=sol['x']

In [11]: par0
Out[11]:
array([20.28165865,  3.5040597 , 25.28577549,  3.3455703 ,  0.11921901,
        0.12987199])
```



Práctica 8:


16
14
12
10
8
6
4
2
0
N
−0.004
−0.003
−0.002
−0.001
0.000
0.001
0.002
0.003
0.004
Errores en f(x)

Práctica 8:

```
In [19]: inx=np.where(er > 0.)

In [20]: ma=len(er[inx])

In [21]: inx=np.where(er < 0.)

In [22]: md=len(er[inx])

In [23]: len(er)
Out[23]: 88

In [24]: mu=2.*ma*md/(ma+md)+1

In [25]: sig=np.sqrt(2.*ma*md*(2*ma*md-88)/88**2/87)

In [26]: import scipy.stats as sts

In [27]: vs=sts.norm.ppf(0.975)

In [28]: vi=sts.norm.ppf(0.025)

In [29]: vs*sig/np.sqrt(88)+mu
Out[29]: 45.881388048841075

In [30]: vi*sig/np.sqrt(88)+mu
Out[30]: 43.93679376934074

In [31]: r=45
```

Práctica 9:

Uso bootstrap para generar muestras sintéticas y obtener parámetros sinteticos del ajuste:

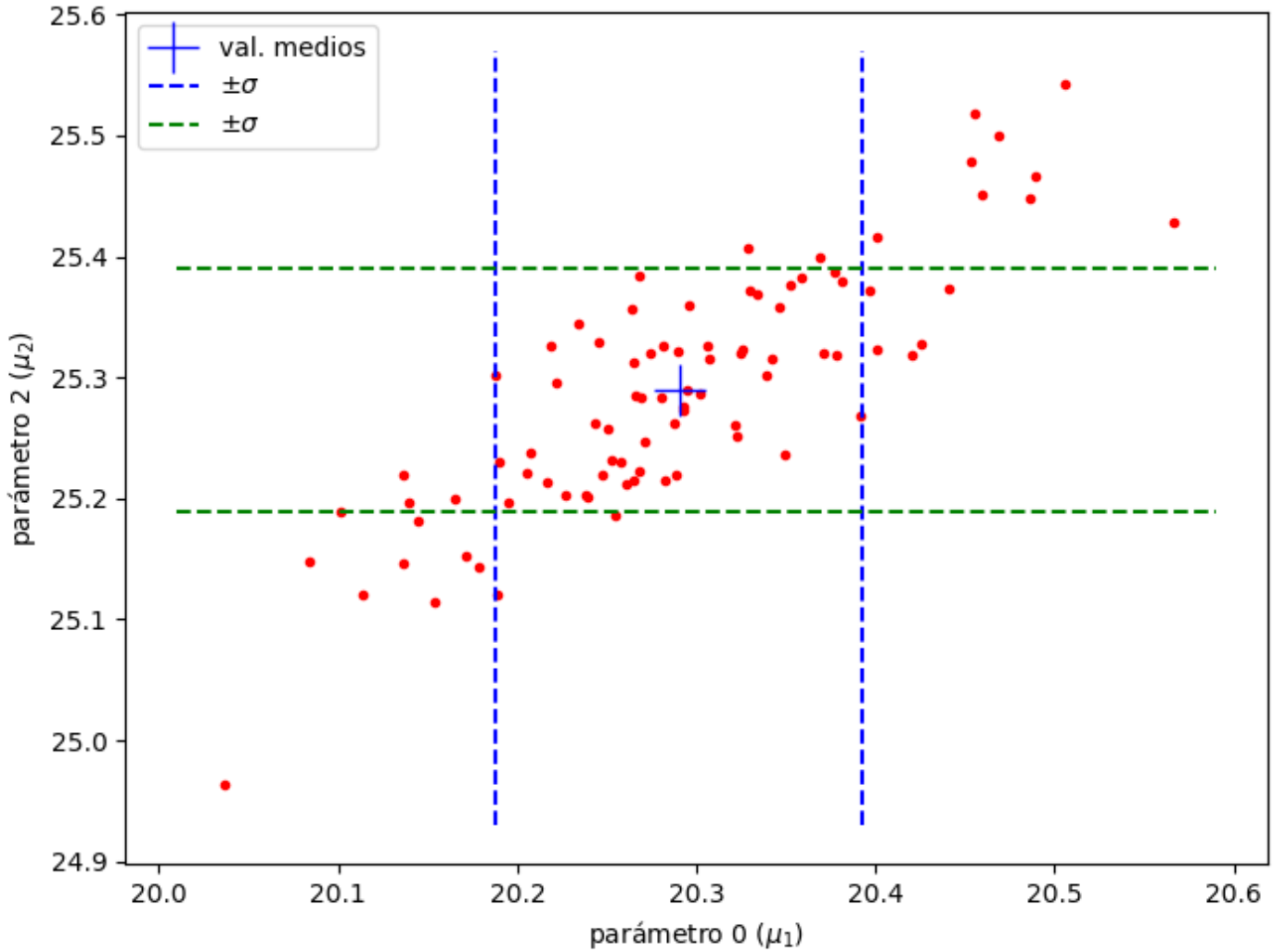
```
In [40]: fit=[]

In [41]: for ii in range(88):
    ...:     inx=np.random.randint(0,88,88)
    ...:     mx=xx[inx]
    ...:     mh=his[inx]
    ...:     sol=opt.least_squares(funcajus,par0,method='lm',ftol=1.e-12,gtol=1.e-12,xtol=1.e-12,arg
    ...: s=(mx,mh))
    ...:     fit.append(sol['x'])
    ...:

In [42]: fit=np.array(fit)

In [43]: vmean=np.mean(fit,axis=0)

In [44]: vstd=np.std(fit,axis=0,ddof=1)

In [45]: vmean
Out[45]:
array([20.29057207,  3.53409682, 25.28965866,  3.31783036,  0.1200152 ,
        0.12924526])

In [46]: vstd
Out[46]:
array([0.10232547, 0.32488471, 0.09991482, 0.23390043, 0.00605945,
       0.00575531])
```

Práctica 9:

Calculo los valores extremos para un intervalo de confianza de 68,3 % para los parámetros simulados (se muestra solo el cálculo de μ_1):

```
In [121]: ss=np.sort(fit[:,0])

In [122]: ic=np.cumsum(ss)

In [123]: ic=ic/ic[-1]

In [124]: linf=(1.-0.683)/2.

In [125]: lsup=1.-linf

In [126]: ilow=np.where(ic > linf)

In [127]: isup=np.where(ic < lsup)

In [128]: ilow[0][0],isup[0][-1]
Out[128]: (14, 73)

In [129]: ss[14],ss[73]
Out[129]: (20.1897778365604, 20.391118505736298)
```

Práctica 9:


val. medios
±σ
±σ
25.6
25.5
25.4
25.3
25.2
25.1
25.0
24.9
parámetro 2 (μ_2)
20.0
20.1
20.2
20.3
20.4
20.5
20.6
parámetro 0 (μ_1)

Práctica 9:

Calculo la matriz de covarianza para obtener las varianzas de los parámetros:

```
In [145]: mcov
Out[145]:
array([[ 1.04705008e-02,  2.47107423e-02,  8.67918042e-03,
        -1.83700466e-02,  4.75440183e-04, -4.96316224e-04],
       [ 2.47107423e-02,  1.05550074e-01,  2.65411289e-02,
        -4.56044812e-02,  1.75706163e-03, -1.52704730e-03],
       [ 8.67918042e-03,  2.65411289e-02,  9.98297113e-03,
        -1.84678390e-02,  5.32595518e-04, -4.94594047e-04],
       [-1.83700466e-02, -4.56044812e-02, -1.84678390e-02,
         5.47094130e-02, -1.02593818e-03,  1.03675687e-03],
       [ 4.75440183e-04,  1.75706163e-03,  5.32595518e-04,
        -1.02593818e-03,  3.67169306e-05, -2.97445877e-05],
       [-4.96316224e-04, -1.52704730e-03, -4.94594047e-04,
         1.03675687e-03, -2.97445877e-05,  3.31236310e-05]])

In [146]: mcov[0,0],mcov[1,1],mcov[2,2],mcov[3,3],mcov[4,4],mcov[5,5]
Out[146]:
(0.010470500813954297,
 0.10555007359066819,
 0.009982971130350552,
 0.05470941298330095,
 3.671693061322836e-05,
 3.312363101801747e-05)
```

Correlación y asociación:

El objetivo es comparar ciertos resultados con otros o con un modelo:

- verificar que las observaciones de otro son rasonables.
- verificar que nuestras observaciones son rasonables.
- para verificar una hipótesis.
- en ausencia de otra suposición, verificar si existe relación entre variables aleatorias para definir una nueva ley.

En el caso de existir alguna relación entre las variables se dice que están correlacionadas o asociadas.

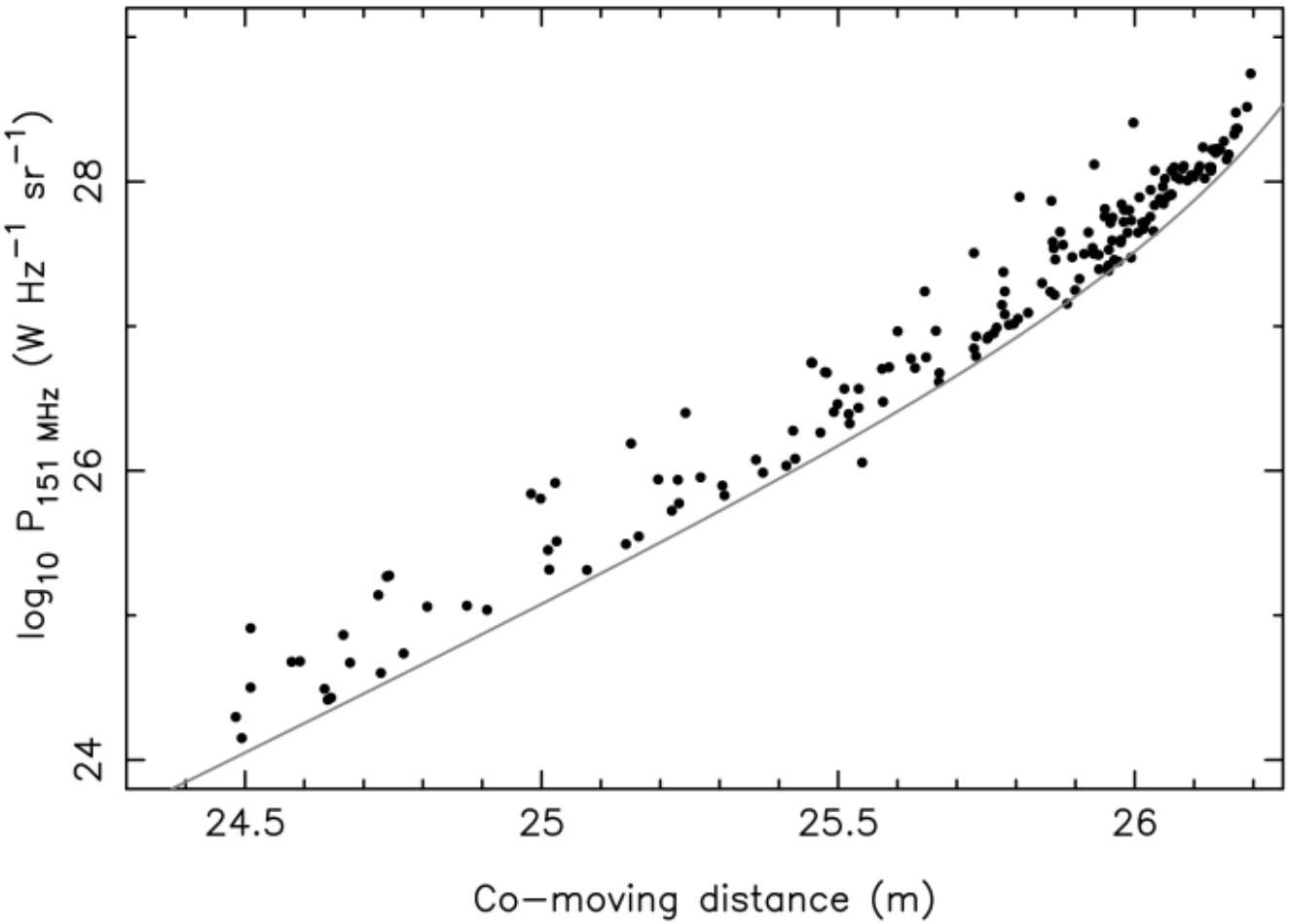
Correlación y asociación:

Para realizar un análisis de correlación hay que tener algunos cuidados:

- la correlación entre dos cantidades se observa gráficamente?
- la correlación observada puede deberse a efectos de selección?
- si se encuentra una correlación, tiene sentido físico?
- hay grupos de puntos agrupados ($\sim 10\,\%$) que si se remueven destruyen la correlación encontrada?.
- recordar que si aparece una correlación entre dos variables es posible que sea una consecuencia de la dependencia de ambas respecto de una tercera.

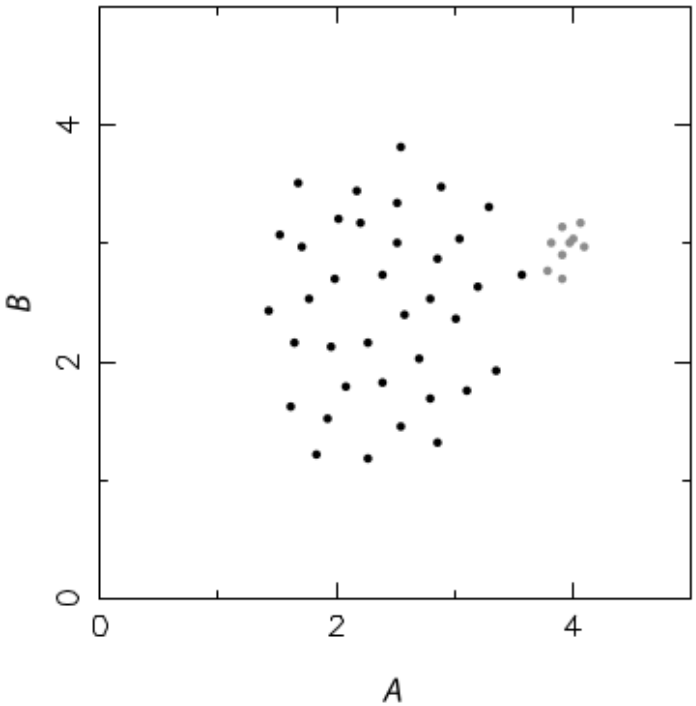
Correlación y asociación:

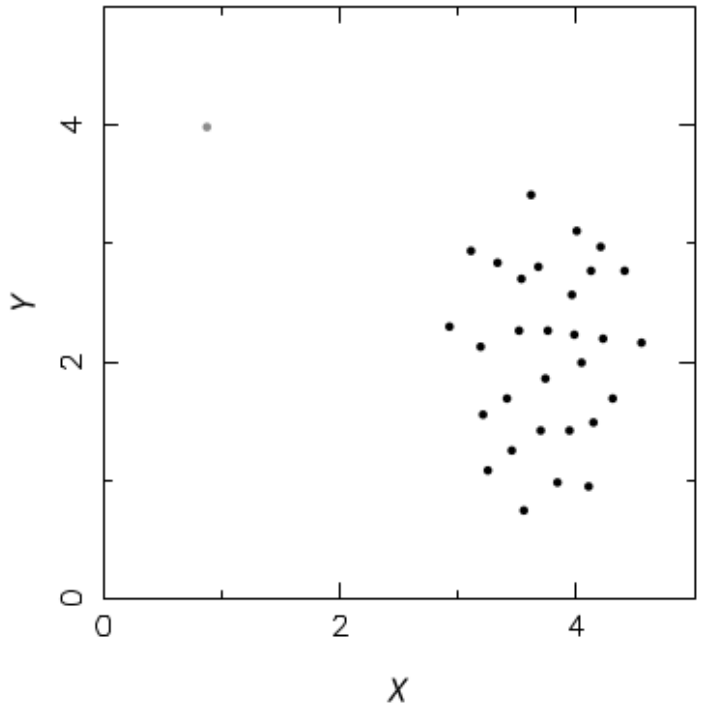
Ejemplo de efectos de selección en luminosidades en radio de radio-fuentes contra módulo de distancia:



Correlación y asociación:

Casos típicos de una falsa correlación:





Correlación y asociación:

Si un modelo tiene parámetros $a_1, a_2, \cdots, a_M$, la matriz de covarianza $\mathcal{C}$ es una matriz cuadrada $M \times M$ cuyos elementos son:

$$C_{i,j} = Cov(a_i, a_j)$$

Esta matriz es simétrica ya que $Cov(a_i, a_j) = Cov(a_j, a_i)$ y en su diagonal están las varianzas de cada parámetro $Cov(a_i, a_i)$.

Si $Cov(a_i, a_j) = Cov(a_j, a_i) = 0$ se dice que los parámetros a_i y a_j son independientes entre sí. Si $Cov(a_i, a_j) = Cov(a_j, a_i) < 0$ los parámetros a_i y a_j están anti correlacionados y en el caso de $Cov(a_i, a_j) = Cov(a_j, a_i) > 0$ se dice que los parámetros a_i y a_j están correlacionados.

Correlación y asociación:

- Para medir correlación hay que definir formalmente como se relacionan dos variables. Entre dos cantidades a y b se utiliza el coeficiente de correlación ρ el cual puede ser calculado a partir de la matriz de covarianza:

$$\rho = \frac{Cov(a, b)}{\sigma_a \sigma_b}$$

donde $-1 < \rho < 1$, siendo $\rho = 0$ prueba de independencia.

Correlación y asociación:

- Un estimador para el coeficiente de correlación es el coeficiente de Pearson:

$$r = \frac{\sum_{i=1}^{N}(a_i - \bar{a})(b_i - \bar{b})}{\sum_{i=1}^{N}(a_i - \bar{a})^2 \sum_{i=1}^{N}(b_i - \bar{b})^2}$$

cuya varianza es:

$$\sigma_r^2 = \frac{(1 - r^2)^2}{N - 1}$$

Correlación y asociación:

- Una prueba paramétrica tradicional es intentar rechazar la hipótesis $\rho = 0$ calculando r y probando para un valor no nulo:

$$t = \frac{r\sqrt{N-2}}{\sqrt{1-r^2}}$$

que tiene una distribución T de Student con $N-2$ grados de libertad. Si el valor de t excede el correspondiente valor crítico la hipótesis de que las variables no tienen correlación se puede rechazar al nivel de significancia elegido.

Correlación y asociación:

- Una prueba no paramétrica bien conocida es el coeficiente de Spearman, donde para un conjunto de pares (x_i, y_i) se obtienen mediante ordenación sus rangos X_i e Y_i y se calcula:

$$r_s = 1 - 6\frac{\sum_{i=1}^{N}(X_i - Y_i)^2}{N^3 - N}$$

donde $0 < r_s < 1$ y un valor alto significa correlación. Si $N \geq 30$, la variable:

$$t_r = r_s\sqrt{\frac{N-2}{1-r_s^2}}$$

también se distribuye como T de Student con $N - 2$ grados de libertad.

Correlación y asociación:

En el módulo scipy.stats tenemos disponibles funciones para varias pruebas de correlación, tanto paramétricas como no paramétricas:

- scipy.stats.pearsonr()
- scipy.stats.spearmanr()
- scipy.stats.pointbiserialr()
- scipy.stats.kendalltau()
- scipy.stats.weightedtau()

PCA:

- Para buscar correlaciones entre varias variables necesitamos aplicar un procedimiento denominado Análisis en Componentes Principales (PCA), el cual siempre se puede describir de forma geométrica o algebraica.
- Consideremos M variables representadas por una nube de puntos en un espacio M-dimensional. Si dos de estas variables están correlacionadas, la nube de puntos aparecerá alargada según cierta dirección en ese espacio. PCA permite identificar estas direcciones, M en total, y las usa como un sistema de coordenadas ortogonal.

PCA:

Primero vamos a hacer una interpretación geométrica. Suponganos que tenemos el siguiente conjunto de N valores de dos variables X e Y:

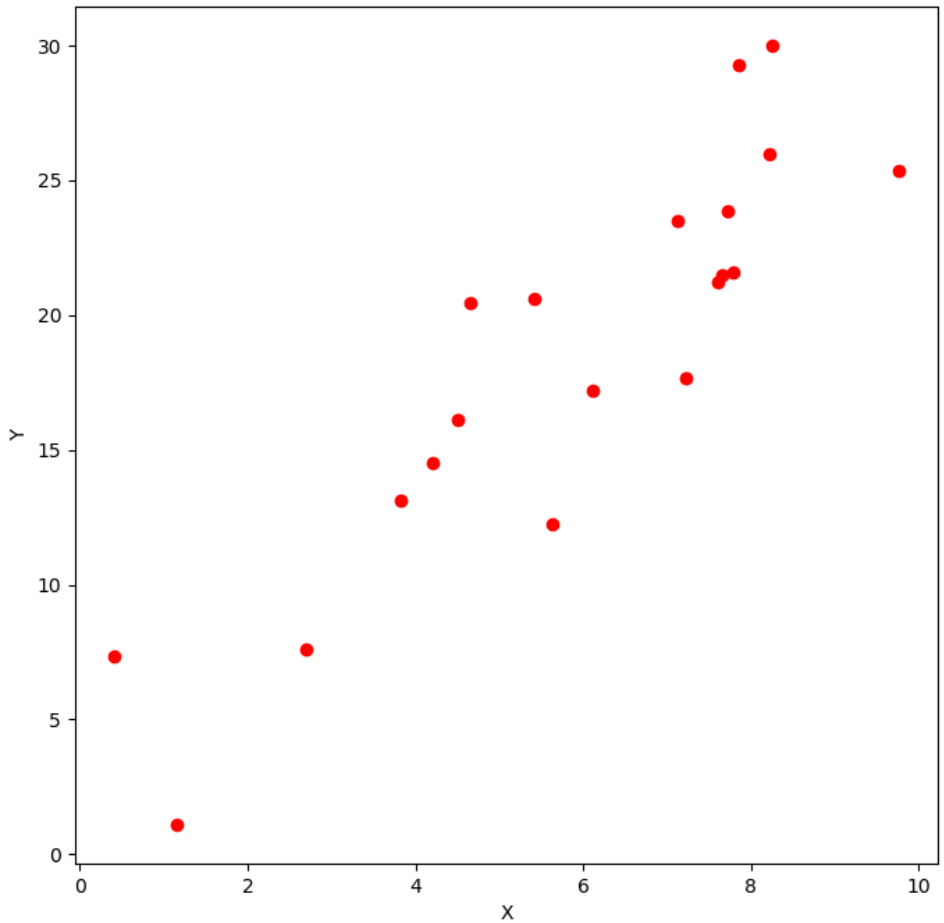
xx=[0.41, 1.15, 2.7 , 3.82, 4.2 , 4.5 , 4.65, 5.41, 5.64, 6.11, 7.12, 7.23, 7.6 , 7.65, 7.73, 7.79, 7.85, 8.22, 8.26, 9.76]

yy=[7.33, 1.10, 7.62, 13.12, 14.54, 16.10, 20.43, 20.59, 12.27, 17.19, 23.52, 17.65, 21.25, 21.51, 23.85, 21.57, 29.28, 26.00, 30.00, 25.34]

Al calcular la matriz de covarianza se verifica que existe una fuerte correlación entre ambas variables al obtener $Cov(x, y) = Cov(y, x) > 0$:

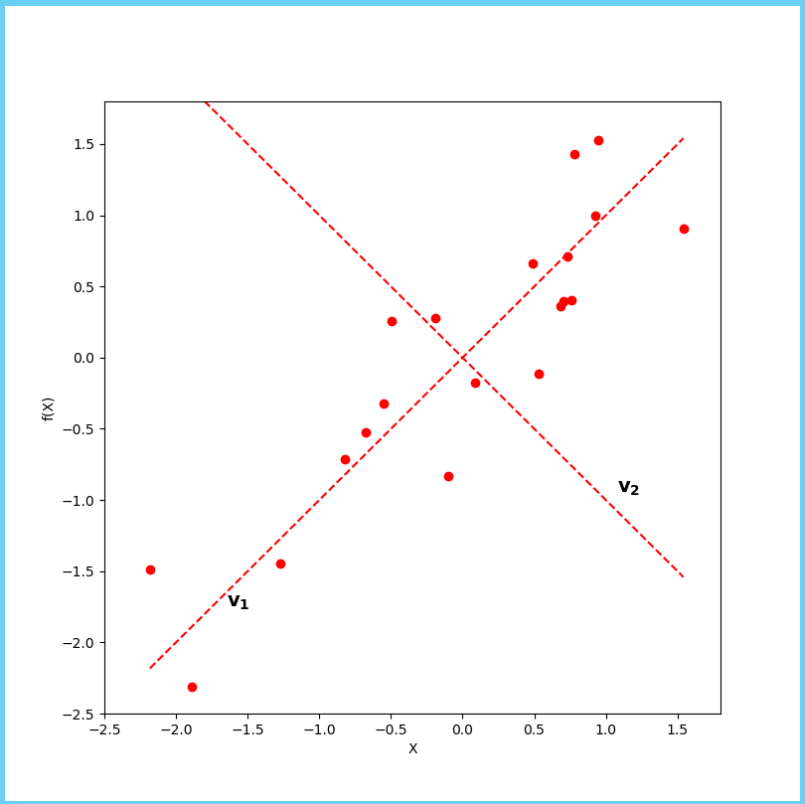
```
In [129]: np.cov([xx,yy],ddof=1)
Out[129]:
array([[ 6.30611579, 16.86710526],
       [16.86710526, 56.68393789]])
```

PCA:



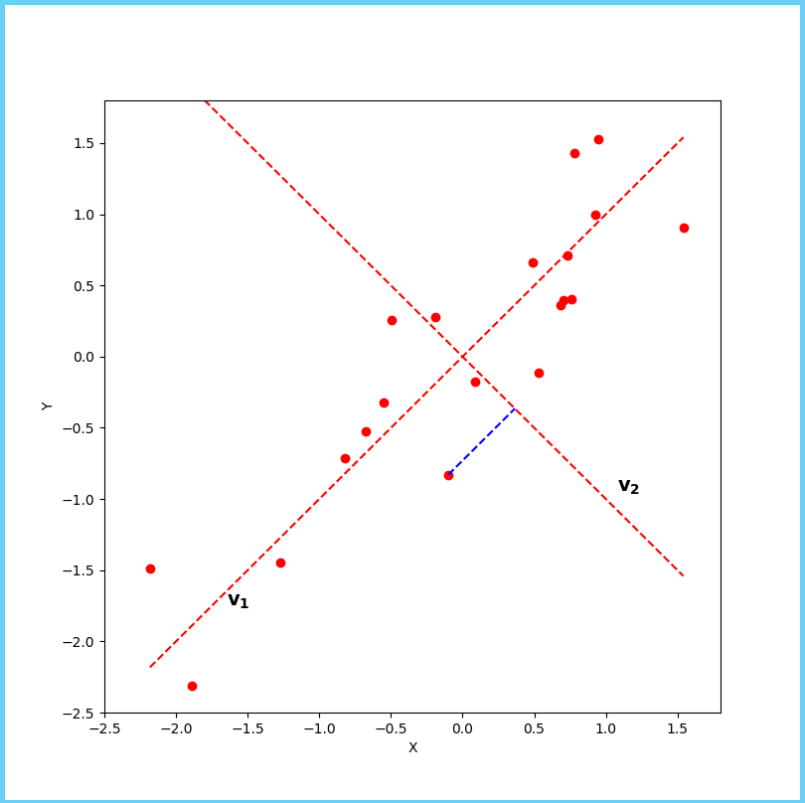
PCA:

- debido a las diferentes escalas y unidades es conveniente renormalizar los datos con su valor medio y desviación standard. Si tenemos una muestra $x'_1, x'_2, \cdots, x'_N$ se renormaliza con $x_i = \frac{x'_i - \bar{x}'}{s}$.



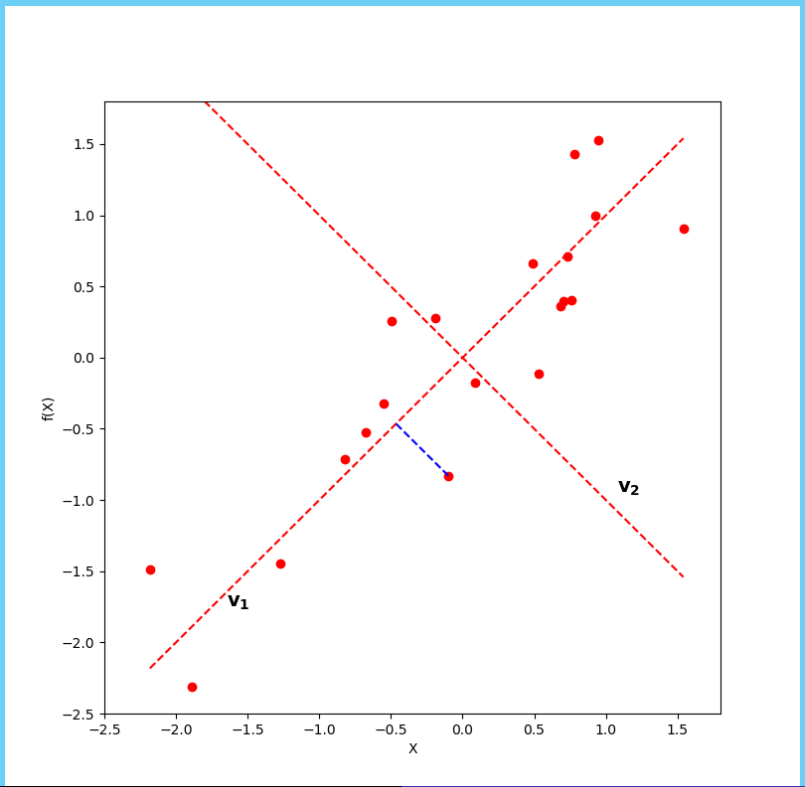
PCA:

- Se identifica la dirección de máxima elongación al minimizar la suma cuadrática de las distancias a este eje. La distancia de cada punto desde el origen sobre este eje corresponde a la Primera Componente Principal (PC1).



PCA:

- La dirección perpendicular a la de máxima elongación corresponde al segundo eje y se define siguiendo el mismo procedimiento. La distancia de cada punto desde el origen sobre este eje corresponde a la Segunda Componente Principal (PC2).



PCA:

- PCA permite identificar las M direcciones posibles al obtener $Cov(x, y) = Cov(y, x) = 0$ en la matriz de covarianza.
- En el caso plano la orientación final de los ejes de referencia se resuelve simplemente con una rotación pero en un sistema multidimensional no es tan sencillo y es necesario definir un nuevo sistema de coordenadas ortogonal.
- Para realizar esta tarea tenemos que considerar antes algunas propiedades matriciales con el objeto de diagonalizar la matriz de covarianza.

Matrices:

- Una matriz $\mathbf{A}$ se dice que es simétrica si es igual a su transpuesta: $\mathbf{A} = \mathbf{A}^\mathbf{T}$.
- Una matriz $\mathbf{A}$ se dice que es hermitiana si es igual a la conjugada compleja de su transpuesta: $\mathbf{A} = (\mathbf{A}^\mathbf{T})^*$.
- Una matriz $\mathbf{A}$ se dice que es ortogonal si su transpuesta es igual a su inversa: $\mathbf{A}^\mathbf{T} * \mathbf{A} = \mathbf{A} * \mathbf{A}^\mathbf{T} = \mathbf{I}$.
- Una matriz $\mathbf{A}$ se dice que es unitaria si su conjugada es igual a su inversa: $\mathbf{A}^* * \mathbf{A} = \mathbf{A} * \mathbf{A}^* = \mathbf{I}$.
- Una matriz $\mathbf{A}$ se dice que es normal si conmuta con su hermitiana conjugada: $(\mathbf{A}^\mathbf{T})^* * \mathbf{A} = \mathbf{A} * (\mathbf{A}^\mathbf{T})^*$.
- Para matrices reales ser hermitiana es igual a simétrica, unitaria es lo mismo que ortogonal y ambas clases son normales.

Autovectores y autovalores:

- Una matriz $\mathbf{A}$ de dimensiones $M \times M$ se dice que tiene un autovector $\mathbf{v}$ y el correspondiente autovalor λ si verifica que:

$$\mathbf{A} * \mathbf{v} = \lambda \mathbf{v}$$

- la ecuación anterior se puede escribir como:

$$(\mathbf{A} - \lambda \mathbf{I}) * \mathbf{v} = \mathbf{0}$$

que solo es posible si $\det(\mathbf{A} - \lambda \mathbf{I}) = 0$, lo que corresponde a un polinomio de grado M en λ cuyas raices son los autovalores buscados.

- en inglés se denominan eigenvector y eigenvalue ("eigen" es una palabra alemana para "propio" o "típico").

Autovectores y autovalores:

- $\det(\mathbf{A} - \lambda\mathbf{I}) = 0$ prueba que siempre hay M autovalores, no necesariamente distintos (en ese caso, autovalores degenerados).
- Además, a cada uno de los autovalores le corresponde un autovector.
- Los autovectores de una matriz normal con autovalores no degenerados son completos y ortogonales cubriendo el espacio vectorial M-dimensional.
- Si se suma a cada lado de la ecuación $\mathbf{A} * \mathbf{v} = \lambda\mathbf{v}$ el término $\tau\mathbf{v}$, los autovalores cambian en una cantidad aditiva τ mientras que los autovectores no cambian.

Autovectores y autovalores:

- Si la matriz $\mathbf{V}$ es una matriz $M \times M$ cuyas columnas son los autovectores de una matriz real y simétrica $\mathbf{A}$, al ser sus autovectores ortonormales tenemos:

$$\mathbf{A} * \mathbf{V} = \mathbf{V} * \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_M \end{pmatrix}$$

- Multiplicando ambos lados por $\mathbf{V^T}$ se logra diagonalizar la matriz:

$$\mathbf{V^{-1}} * \mathbf{A} * \mathbf{V} = \mathbf{V^T} * \mathbf{A} * \mathbf{V} = \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_M \end{pmatrix}$$

PCA:

Ahora vamos a hacer una interpretación algebraica.
Suponganos que tenemos el mismo conjunto de N valores de dos variables X e Y:

xx=[0.41, 1.15, 2.7 , 3.82, 4.2 , 4.5 , 4.65, 5.41, 5.64, 6.11, 7.12, 7.23, 7.6 , 7.65, 7.73, 7.79, 7.85, 8.22, 8.26, 9.76]

yy=[7.33, 1.10, 7.62, 13.12, 14.54, 16.10, 20.43, 20.59, 12.27, 17.19, 23.52, 17.65, 21.25, 21.51, 23.85, 21.57, 29.28, 26.00, 30.00, 25.34]

- Se renormalizan los datos con su valor medio y desviación standard, y se calcula la matriz de covarianza:

```
In [197]: mcov=np.cov([xx0,yy0],ddof=1)

In [198]: mcov
Out[198]:
array([[1.        , 0.89213295],
       [0.89213295, 1.        ]])
```

PCA:

- Calcular los autovalores y autovectores de la matriz de covarianza:

```
In [199]: import numpy.linalg as alg

In [200]: w,V=alg.eig(mcov)

In [201]: w
Out[201]: array([1.89213295, 0.10786705])

In [202]: V
Out[202]:
array([[ 0.70710678, -0.70710678],
       [ 0.70710678,  0.70710678]])
```

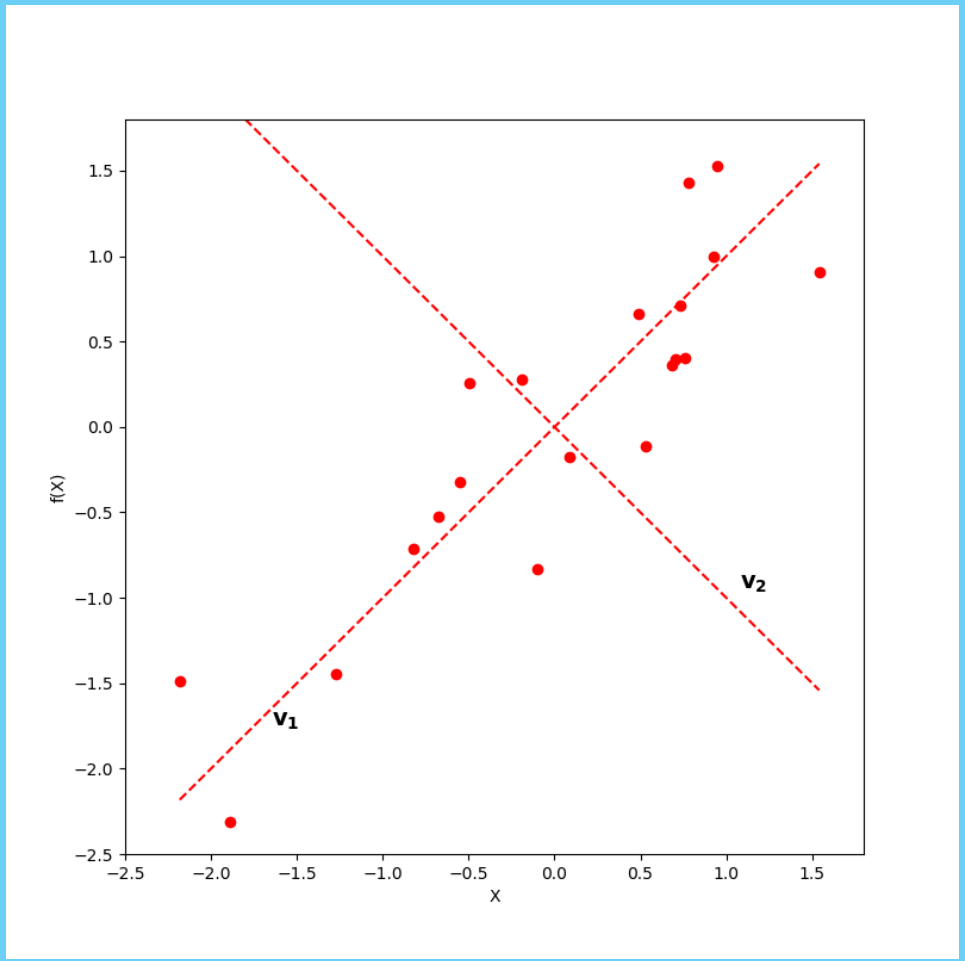
- Los autovalores encontrados en el vector **w** dan el peso de cada eje de correlación e indican el orden de los mismos (PC1 es el de mayor valor, PC2 el siguiente, etc.).
- La columnas de la matriz **V** corresponden a cada autovector en el mismo orden que se dan los autovalores en **w**.

PCA:

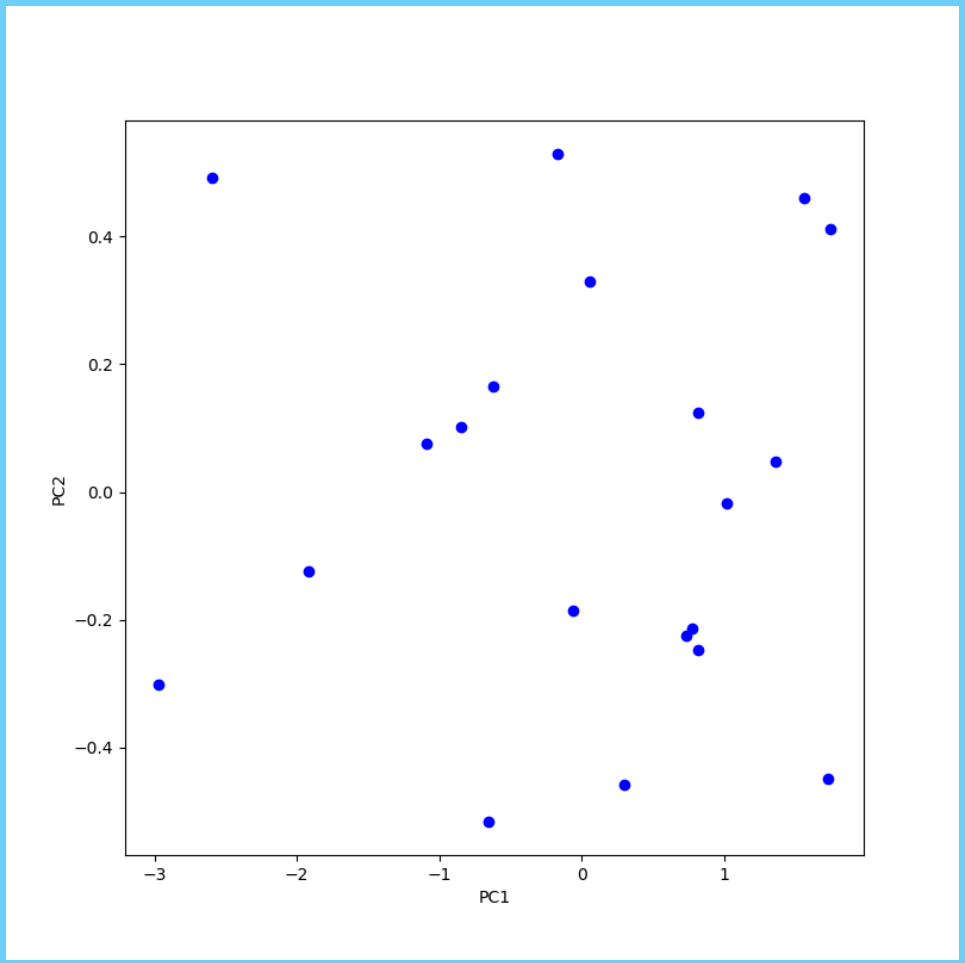
- Construir una matriz con los autovectores en las filas y según el orden indicado por los autovalores, pudiendo descartar los que tengan autovalores muy bajos.
- Construir una matriz con los vectores de datos para cada variable en las filas en el mismo orden que se utilizó para calcular la matriz de covarianza.
- El producto de estas matrices dan los valores de PC1 y PC2 para cada punto:

```
In [203]: VT=np.array([V[:,0],V[:,1]])

In [204]: val=np.array([xx0,yy0])

In [205]: pc=np.dot(VT,val)
```

Datos escalados originales



Transformación en PC



PCA:

Ejemplo: Utilizando la cuarta versión del Moving Object Catalogue del Sloan Digital Sky Survey (SDSS-MOV) se hace un análisis de componentes principales de los flujos observados para asteroides numerados con el objeto de clasificarlos taxonómicamente.

Se van a extraer las magnitudes y sus errores, se encontrarán los colores corregidos por la contribución del Sol, y se obtendrán los flujos normalizados al flujo en el filtro r'.

De las más de 200000 observaciones se seleccionan aquellas cuyos errores en flujo sean menores al 10 % y que los flujos en diferentes filtros cumplan con una serie de requisitos.

Un paper con un trabajo similar aunque algo diferente en el procedimiento es Icarus **183**, 411-419, 2006.

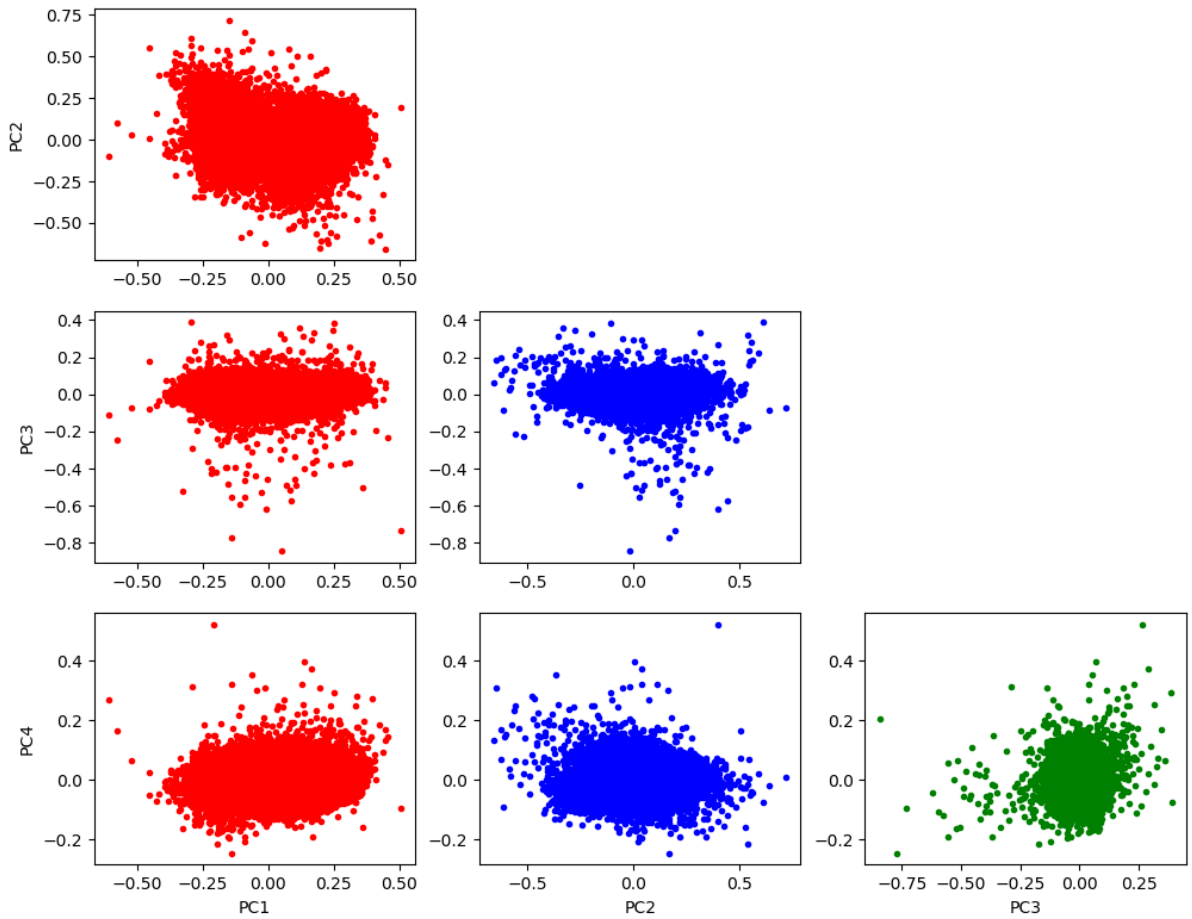
PCA:

```
In [2]: f=open('ADR4.dat','r')

In [3]: flx=[]

In [4]: lin=f.readline()

In [5]: while(lin != ''):
   ...:     if(int(lin[242:244])==1):
   ...:         mu=float(lin[163:169])
   ...:         mg=float(lin[174:180])
   ...:         mr=float(lin[185:191])
   ...:         mi=float(lin[196:202])
   ...:         mz=float(lin[207:213])
   ...:         du=float(lin[169:174])
   ...:         dg=float(lin[180:185])
   ...:         dr=float(lin[191:196])
   ...:         di=float(lin[202:207])
   ...:         dz=float(lin[213:218])
   ...:         du=np.sqrt(du**2+dr**2)
   ...:         dg=np.sqrt(dg**2+dr**2)
   ...:         di=np.sqrt(di**2+dr**2)
   ...:         dz=np.sqrt(dz**2+dr**2)
   ...:         dr=np.sqrt(2.)*dr
   ...:         du=0.921*du*(1.+0.4605*du)
   ...:         dg=0.921*dg*(1.+0.4605*dg)
   ...:         dr=0.921*dr*(1.+0.4605*dr)
   ...:         di=0.921*di*(1.+0.4605*di)
   ...:         dz=0.921*dz*(1.+0.4605*dz)
   ...:         if((du<0.1)and(dg<0.1)and(dr<0.1)and(di<0.1)and(dz<0.1)):
   ...:             fu=10.**(-0.4*(mu-mr-1.77))
   ...:             fg=10.**(-0.4*(mg-mr-0.45))
   ...:             fi=10.**(0.4*(mr-mi-0.10))
   ...:             fz=10.**(0.4*(mr-mz-0.14))
   ...:             if((fu<1.0)and(fg<1.3)and(fi<1.5)and(fz<1.7)and(fg>0.6)and(fg>fu)):
   ...:                 flx.append([fu,fg,fi,fz])
   ...:     lin=f.readline()
   ...:

In [6]: f.close()
```

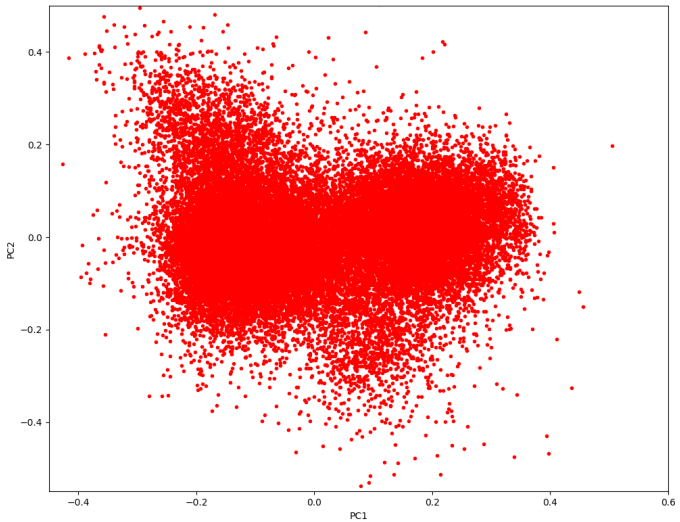
PCA:

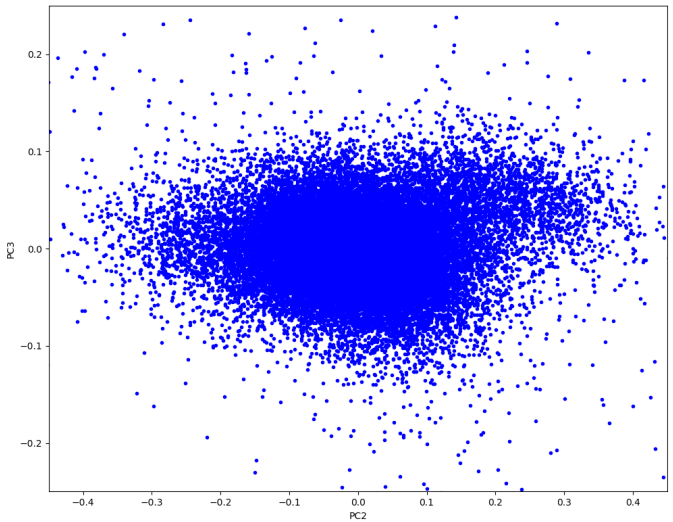
```
In [7]: len(flx)
Out[7]: 34858

In [8]: flx=np.array(flx).T

In [9]: vmed=np.mean(flx,axis=1)

In [10]: back=np.reshape(np.tile(vmed,34858),(34858,4)).T

In [11]: flx=flx-back

In [12]: mcov=np.cov(flx,ddof=1)

In [13]: mcov
Out[13]:
array([[ 0.0186115 ,  0.00956369, -0.00403608,  0.00173029],
       [ 0.00956369,  0.00718289, -0.00232327,  0.00112185],
       [-0.00403608, -0.00232327,  0.00427747,  0.00280536],
       [ 0.00173029,  0.00112185,  0.00280536,  0.01082033]])

In [14]: import numpy.linalg as alg

In [15]: w,v=alg.eig(mcov)

In [16]: w
Out[16]: array([0.0252203 , 0.01185301, 0.00174086, 0.00207801])

In [17]: v
Out[17]:
array([[ 0.84679196,  0.0230805 , -0.44263172,  0.29408812],
       [ 0.48135711,  0.00132529,  0.87362516, -0.07122258],
       [-0.20324255, -0.35832538,  0.18526254,  0.89217329],
       [ 0.09965524, -0.93331048, -0.08083337, -0.33536002]])

In [18]: vt=np.vstack((v[:,0],v[:,1],v[:,3],v[:,2]))

In [19]: pca=np.dot(vt,flx)
```

PCA:



PCA:


PC2
PC1
0.4
0.2
0.0
−0.2
−0.4
−0.4
−0.2
0.0
0.2
0.4
0.6


PC3
PC2
0.2
0.1
0.0
−0.1
−0.2
−0.4
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4

Actividades:

Práctica 10:

- Hacer un Análisis de Componentes Principales con todas las variables del archivo de datos que está usando para los ejercicios del curso.

Entrega

Para la próxima clase

Por consultas: ricardo.gil-hutton@conicet.gov.ar
Grupo de Ciencias Planetarias - CUIM 2