==========

PCA: automatic optimal dimensionality reduction

==========

The aim of the present notebook is numerically-testing the **Geometric-Complexity algorithm** (GC_alg) in [Mera&al2021] for automatic choice of the optimal dimensionality for Principal Component Analysis (PCA).

- *The first part is dedicated to data-preprocessing*.

After a brief introduction to notation and convention used for PCA, we start by defining a few function for pre-processing the data, assumed to be obtained by a multivariate source. In particular, we introduce an euristic way of determining the precision-parameter `s` that is going to play an essential role in the correction brought by the GC_alg with respect to pre-existing algorithms.

- *In the second part is dedicated to algorithm-implementation*.

We first introduce the *scree-plot* for PCA-visualisation, then the main two pre-existing algorithms for dimensionality-reduction, Minka's algorithm and Tavory's algorithm, and finally new Geometric-Complexity algorithm.

- *The third part is dedicated to data-analysis/algorithm testing*.

In the last part, after introducing some extra function for algorithm testing with respect to the relevant parameters -- data-precision `s`, data-length `N` and number of features `d` -- we proceed with the tests on real datasets.

- *Finally, we draw some conclusions.*

Table of content

In [1]:
```
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

In [2]:
```
print(__doc__)
```

```
 2
 3 # Authors: Gael Varoquaux
 4 #          Jaques Grobler
 5 #          Kevin Hughes
 6 # License: BSD 3 clause
 7
 8 from sklearn.decomposition import PCA
 9 from sklearn.covariance import EmpiricalCovariance
10 from sklearn.preprocessing import scale
11 from math import gamma
12
13
14 from mpl_toolkits.mplot3d import Axes3D
15 import numpy as np
16 import matplotlib.pyplot as plt
17 from scipy import stats
18 from scipy.special import loggamma as lgm
19 from numpy import genfromtxt
20 from tabulate import tabulate
21
22 import kneed
23
```

```
Automatically created module for IPython interactive environment
```

Notation ◇

We assume the dataset $\mathbb{X}$ to be in a matrix-form whose rows and columns store respectively the instances and the different features associated with the experiment. In particular $\mathbb{X}$ is a Nxd matrix where:

- $d \geq 1$ is the integer indicating the number of parameters;
- $N \gg d$ is the integer representing the number of instances.

Storing the data in `ndarrays` , we have `shape(data)=(N,d)` .

The covariance matrix is the $d \times d$ positive-semidefinite matrix s.t. $\Sigma = \mathbb{X}^T\mathbb{X}$.

Data pre-processing ◇

Zero mean data △

In [3]:
```
1 def zeromean(data):
2     col=np.size(data,1)
3     newdata=data
4     for i in range(col):
5         mean=np.mean(data[:,i])
6         newdata[:,i]=data[:,i]-mean
7     return newdata
```

Data description length: the s parameter △

The Geometric-Complexity algorithm strongly depends on the empirical data-precision, encoded in the parameter s, defined in [Mera&al.2020] as *the smallest integer that satisfies*

$$\mathbb{1}_d \leq \Sigma \leq 2^{2s}\ \mathbb{1}_d.$$

The latter is an operatorial inequality where $\mathbb{1}_d$ indicates the identical operator $d \times d$. The

first part requires the covariance matrix to be rescaled s.t. its eigenvalues are multiple of a fundamental precision. We rescale the data matrix considering as fundamental precision 10^{-n} where n is the number of significative digits for every feature. Indicating with Σ_{int} the integers-matrix obtained from the rescaled data, naturally $\Sigma_{\text{int}} \geq \mathbb{1}_d$, i.e. $(\Sigma_{\text{int}} - \mathbb{1}_d)$ is positive semidefinite.

The second part of the inequality defines s as the smallest integer s.t. the matrix $(\Sigma_{\text{int}} - 2^{2s}\,\mathbb{1}_d)$ is positive semidefinite. Indicating with σ_1 the largest eigenvalue of Σ_{int}, we get:

$$s \geq \frac{1}{2}\left(\log_2 \sigma_1\right) \quad \Rightarrow \quad s = \lceil \frac{1}{2} \log_2 \sigma_1 \rceil .$$

We can obtain a scalar inequality also by considering the trace of the operator $d \leq \Sigma \leq d\,2^{2s}$, which results in an averaged s:

$$s \geq \frac{1}{2}\left(\log_2 \operatorname{Tr}\Sigma_{\text{int}} - \log_2 d\right) \quad \Rightarrow \quad s = \lceil \frac{1}{2}\left(\log_2 \operatorname{Tr}\Sigma_{\text{int}} - \log_2 d\right) \rceil .$$

The Geometric-Complexity algorithm gives an higher weight to the model description with respect to the other algorithms considered. Maybe considering an averaged s can avoid dimensionality underestimation when the dataset is small.

The functions defined below respectively:

- `data_int()` rescales the dataset so that the entries are integers multiple of some fundamental precision;
- `precision()` compute the parameter s from the data-matrix -- no need to rescale it before -- when its features have same precision.

In [4]:
```
print(np.finfo(numpy.float))
```

```
Machine parameters for float64
---------------------------------------------------------------
precision =  15   resolution = 1.0000000000000001e-15
machep =    -52   eps =        2.2204460492503131e-16
negep =     -53   epsneg =     1.1102230246251565e-16
minexp =  -1022   tiny =       2.2250738585072014e-308
maxexp =   1024   max =        1.7976931348623157e+308
nexp =       11   min =        -max
---------------------------------------------------------------
```

In [5]:
```
def data_int(data,order):

    '''INPUT: np.array data-matrix Nxd, integer data-order (if dat
    OUTPUT: np.array of integers, equal to data-matrix*order.'''

    data1=data[:]*order
    for i in range(shape(data1)[0]):
        for j in range(shape(data1)[1]):
            data1[i][j]=int(data1[i][j])
    return data1
```

In [6]:
```
def precision(data,sign_digits,b=0):

    '''INPUT: data-matrix Nxd and an integer=max[number of data-en
    OUTPUT: integer equal to the minimum number of bits for the co
    Options: the function computes the output as the number of bit
    covariance matrix -- b=0 (default). Setting b=1, the function
    for storing tha largest eigenvalue of the rescaled covariance

    data1=data[:]*(10**sign_digits)
    for i in range(shape(data1)[0]):
        for j in range(shape(data1)[1]):
            data1[i][j]=int(data1[i][j])
    cov=data1.T@data1
    if b==1:
        eig=np.linalg.eig(cov)[0]
        eig.sort
```

```
        A=eig[0]
    else:
        d=shape(data)[1]
        A=np.trace(cov)/d
    return int((np.log(A))/(2*np.log(2)))+1 ## 1/2 of the length o
```

Test dataset △

The function below creates ad-hoc datasets from a given one for algorithm-testing: given a data-matrix, the function keeps the first *k* columns/features and replaces the last *d-k* ones with linear combinations of the first *k* ones with an additional noise $\mathcal{N}(0, \tau)$. So the right choice for dimensionality reduction on the resulting dataset is exactly *k*. Code adapted from [A.Tavory2019].

$$\texttt{data\_Tavory (data, k, tau) = data[:k]} + \sum_{i=1}^{k} \texttt{tau}_i * \texttt{data[i]} \text{ , with tau}_i \text{ sampled from } \mathcal{N}(0, \tau)$$

In [7]:
```
def data_Tavory(data,k,tau):

    '''INPUT data-matrix Nxd, an integer k<d and a noise term tau,
    OUTPUT data-matrix whose first k columns are the original one,
    the last d-k are a random linear combination of the first k wi

    assert k<shape(data)[1]
    data=zeromean(data)
    ldata=data[:,:k]
    m = data.shape[1]
    rdata = np.dot(ldata,np.random.randn(k, m - k)) + tau * np.ran
    return np.concatenate([ldata.T, rdata.T]).T
```

Data analysis: PCA ◇

- Minka and Scree-plot knee
- Tavory NML
- Geometric complexity

In Python's library Sklearn, Principal Component analysis is implemented by the function `PCA()` (https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html).

In multivariate statistics, data'a **singular values** -- the eigenvalues of factors or principal components -- are used to be visualized with a line plot called ***scree plot***.

In [8]:
```
def scree_plot(data):

    '''Line plot of data's Singular Values'''

    data=zeromean(data)
    data_pca=PCA().fit(X=data)
    plot(data_pca.explained_variance_ratio_, '-.', label='Var_rati
    legend();
    xlabel('Number of factors').set_color('black');
    ylabel('Explained variance ratio').set_color('black');
```

- **Finding knee in the Scree plot** In the Scree Plots above, there seems to be a ''bend", that can indicate the optimal number of components. Using the Kneedle algorithm (see [Satopaa2011]) for finding "bends" in plots, we get a possible estimation of optimal number of components, as a function of the dataset lengtht. We use the Python repository kneed (https://pypi.org/project/kneed/) for implementing the algorithm.

As A.Tavory underlines, this method is known for its tendency to find a lower number of

In [9]:
```
def find_knee(pca):
    '''Returns the knee of the scree plot of the explained_varianc
    yv = pca.explained_variance_ratio_
    xv = np.linspace(1, len(yv), len(yv))
    return kneed.KneeLocator(xv, yv, S=1.0, curve='convex', direct
```

- **Minka's optimal dimensionality** When selecting `PCA(mle)` Minka's algorithm described in [Minka2000] is used for selecting the optimal dimensionality reduction. By interpreting PCA as density estimation, Minka'a algorithm estimates the true dimensionality of the data using Bayesian model selection -- provided enough data. The algorithm assumes a Gaussian prior.

In [10]:
```
def Minka_optk(data):
    data=zeromean(data)
    pca_mle=PCA('mle').fit(data)
    print('Minka Complexity',pca_mle.components_.shape[0])
```

NML - Tavory △

In [Tavory2019] the author applies the MDL (Minimum Description Length) principle -- in its modern definition NML (normalized) -- to the PCA problem of computing the optimal data dimensionality. No prior probability distribution is required by the criteria, so Tavory-NML algorithm should provide the optimal dimensionality reduction also for non Gaussian distributed data.

The `nml_optk()` function implements Tavory-NML algorithm. The code has been adapted from the numerics published in the same paper.

In [11]:
```
def nml_optk(data):

    '''INPUT: data matrix NXd
    OUTPUT: optimal dimensionality reduction -- integer k<=d -- vi

    data=zeromean(data)
    n, m = shape(data)[0], shape(data)[1]
    optdM=-1;
    optscoreM=np.Infinity
    p = PCA().fit(X=data)
    log_sigma=np.log(np.dot(data.T, data).sum())

    for k in range(1,m-1):
        tau=sum(p.explained_variance_[k:])
        s=(n * m - k * m) * np.log(tau)
        s=s+k*m*log_sigma
        s=s+(m * n - k * n - 1) * np.log( m * n / (m * n - k * m))
        minus=(m * k + 1) * np.log(m * k)
        s=s-minus
        smax=s+m * k * np.log(m / np.sqrt(np.e)) + (k - 1) * np.lo
        if(smax<optscoreM):
            optdM=k;
            optscoreM=smax

    return optdM
```

Geometric Complexity △

Data matrix rotation

The covariance matrix Σ is diagonal in the new basis. Setting U as the orthogonal matrix of basis change:

- $\Sigma = \frac{1}{d-1} X^T X$ -- empirical covariance matrix;
- $\Lambda = U \Sigma U^T = W^T W = \text{diag}(\sigma_1, \sigma_2, \ldots)$ with $W = XU^T$ the rotated dataset.

`zmdata_pca.components_` is the ndarray of shape (n_components, n_features) corresponding to the basis change matrix U. Then the rotated dataframe: `rotdata=mult(zmdata,zmdata_pca.components_.T)` .

In [12]:
```
def changebasis(data, components):
    return np.matmul(data, components.T)
```

Optimal Log-likelihood

From Eq.16 [Mera&al2020], the code-length associated with a set of i.i.d. data $\{x \in \mathbb{K}^d\}_{i=1}^N$ is:

$$\mathcal{L}(x^*) = -\log\ p(x^N|\hat{Q}) + \underbrace{\frac{m(m+1)}{4} \log\left(\frac{N}{2\pi}\right)}_{a} + \underbrace{\frac{(m-1)m(m+2)}{24\,N}}_{b} + \log \text{vol}_g M($$

where the logarithm are in natural base. $Q := \Sigma^{-1}$ is the *precision matrix*, m ranges over the number of features $[0, d]$ and represents the optimal dimensionality of the data.

- **Log-Gaussian distribution**:

$$\log\ p(x^N|\hat{Q}) = \frac{N}{2} \log\det\left(\frac{1}{2\pi}\hat{Q}\right) - \frac{1}{2}\sum_x x^T \hat{Q} x$$

where Q is the precision matrix i.e. $Q = \Sigma^{-1}$. Then we have:

$$\log\det\left(\frac{1}{2\pi}\hat{Q}\right) = -d\log(2\pi) - \sum_{i=1}^{d} \log \text{eigen}_i(\Sigma^{-1}) \approx -d\ \log(2\pi) - \sum_{i=1}^{m} \log \sigma_m^{-1} - (d$$

and

$$\sum_x x^T \hat{\Sigma}^{-1} x = \sum_x x^T U U^T \hat{\Sigma}^{-1} U U^T x = \sum_x x^T U\, \text{diag}(\sigma_1^{-1}, \ldots, \sigma_d^{-1}) U^T x$$
$$= \text{tr}\left[\text{diag}(\sigma_1^{-1}, \ldots, \sigma_d^{-1}) \sum_w w w^T\right] \approx \text{tr}\left[\sigma_m \sum_w w w^T\right]$$

where Λ_m is the diagonal covariance matrix whose last $d - m$ eigenvalues have been approximated by their average $\bar{\sigma}$, i.e. $\Lambda_m = \text{diag}(\sigma_1^{-1}, \ldots, \sigma_m^{-1}, (\bar{\sigma})^{-1}, \ldots (\bar{\sigma})^{-1})$.

Finally:

$$\log\ p(x^N|\hat{Q}) \approx -\frac{Nd}{2}\log(2\pi) - \frac{N}{2}\sum_{i=1}^{m} \log\sigma_m - \frac{N(d-m)}{2}\log\bar{\sigma} - \frac{1}{2}\text{tr}\left[\sigma_m \sum_w w w^T\right.$$

where $\bar{\sigma} = (d-m)^{-1} \sum_{i=m+1}^{d} \sigma_i$.

- **Log-vol** term: From Eq.15:

$$\log \text{vol}_g(M(s)) = \underbrace{-\frac{m}{2}\log 2 + \frac{m(m+1)}{4}\log\pi - \log m!}_{c} - \sum_{j=1}^{m} \log\Gamma\left(\frac{j}{2}\right) + \log\ I(s)$$

Recall $\Gamma(n+1) = n!$, we write $\log m! = \log\Gamma(m+1)$. Then, using the upper bound for $I(s)$:

$$\log\ I(s) \leq m\log(s\log(2)) + (2s+1)\frac{m(m-1)}{4}\log 2$$

Observe that $\log(m!) = \sum_{a=2}^{m}\log(a)$.

The function `GC_optk()` optimizes the log-code-length described above with respect to m, setting by default the precision-parameter to the machine limit for storaging floats,

In [32]:

```
def GC_optk(data,s=64):

    def sum_logf(length):
        '''Return a list of given length whose entries are:
           list[i-1]=sum_a=1^i(np.log(a)+np.log(gamma(a/2)))'''
        sumlog_vec=[]
        sumlog1=0
        for a in range(1,length):
            sumlog1=sumlog1+np.log(a)+lgm(a/2)
            sumlog_vec.append(sumlog1)
        return sumlog_vec

    def log_gauss(N,d,k,lambdas,Sw):

        'lambdas=pca.explained_variance_'
        #k=k-1
        if k==d:
            lambda_bar=1
        else:
            lambda_bar=sum(lambdas[k:])/(d-k)
        #print('k:',k,'lambda_bar',lambda_bar)
        Q=np.diag([1/eig for eig in lambdas[:k]]+[1/lambda_bar for
        r=d*np.log(2*np.pi)
        r=r+sum(np.log(lambdas[:k])) #kth-eigen included
        r=r+(d-k)*np.log(lambda_bar)
        r=r+np.trace(Q@Sw)
        return r*N/2

    zm_data=zeromean(data)
    pca = PCA().fit(zm_data)
    dimrange=pca.components_.shape[0] #number of features
    N=zm_data.shape[0] #number of instances
    if (dimrange != zm_data.shape[1]):
            print("attributes have linear dependencies, ther is a
            print("Solution: remove dependent variables eg: averag
            return "error";
    rotdata=changebasis(zm_data,pca.components_)
    optd=-1;
    optscore=np.Infinity
    S=np.matmul(rotdata.T,rotdata)
    sumlog_vec=sum_logf(dimrange+1)
    for m in range(1,dimrange+1):
        r=log_gauss(N,dimrange,m, pca.explained_variance_,S)
        r=r+m*(m+1)/4*np.log(N/(2*np.pi)) #a
        r=r+m*(m+2)*(m-1)/(24*N) #b
        r=r-m/2*np.log(2)+m*(m+1)/4*np.log(np.pi) #c
        sumlog=sumlog_vec[m-1]
        r=r-sumlog
        r=r+m*np.log(s*np.log(2))+(2*s+1)*m*(m-1)/4*np.log(2)
        if(r<optscore):
            optd=m;
            optscore=r

    return optd
```

Tests ◇

Test-functions ◇

For a given dataset:

- `GC_s()` plots the optimal reduced dimension via *Geometric Complexity algorithm* in function of the precision parameter $s \in [0, 64]$.
- `GC_table()` is an adjustment of the function `GC_optk()` : other than computing the optimal dimensionality reduction, returns a table with the values of the terms of the Geometric Complexity with the increasing of the dimensionality;
- `PCA_test()` compares optimal reduced dimensionality obtained from *scree-plot knee*, *Minka's algorithm*, *Tavory's algorithm*, *Geometric-Complexity-algorithm*, showing the results on the scree-plot;
- `PCA_datalen()` optimal reduced dimensionality in function of dataset-length;
- `PCA_datafeat()` optimal reduced dimensionality in function of the number of features in the data;

- **GC_s()**

In [18]:
```
def GC_s(data,tick=0):

    '''INPUT: dataset;
    OUTPUT: plot of the optimal-dimensionality-reduction -- chosen
    in function of the number of bits s\in[1,64] used for describi
    Options: tick=1 set label's locations.'''

    v=[]
    gc_vec=[]
    for i in range(1,65):
        v.append(i)
        gc=GC_optk(data,i)
        gc_vec.append(gc)
    plot(v,gc_vec,'x',markersize=2,color="blue")
    if tick==1:
        yticks(np.arange(gc_vec[-1]-2,gc_vec[1]+2,step=1))  # Set
        xticks(np.arange(0,65,step=4))  # Set label locations
        grid()
    xlabel('Precision').set_color('black');
    ylabel('Optimal dimension').set_color('black');
```

- **PCA_test()**

In [19]:
```
def PCA_test(data, knee=1, rounds=10, minka=1, tavory=1, GC=1, s=6

    '''INPUT: data matrix Nxd
    OUTPUT: scree plot and knee, optimal dimensionality reduction
    Options: by default all the algorithm above are runned -- knee
    You can choose to exclude any one by setting knee/minka/tavory

    data=zeromean(data)
    data_pca=PCA().fit(X=data)

    eigen=data_pca.explained_variance_ratio_
    plot(eigen, '.', markersize=1,label='Var_ratio',color="black")
    #eigen1=data_pca.explained_variance_ratio_[0]

    if knee==1:
        data_kneedle=find_knee(data_pca)
        print('knee',round(data_kneedle, rounds))
        plot(data_kneedle,eigen[int(data_kneedle)],'2', markersize

    if minka==1:
        pca_mle=PCA('mle').fit(data)
        mc=pca_mle.components_.shape[0]
```

```
        print('Minka Complexity',mc)
        plot(mc,eigen[int(mc)],'2', markersize=20, label='Minka',c

    if tavory==1:
        nml=nml_optk(data)
        print('Tavory Complexity',nml)
        plot(nml,eigen[int(nml)],'2', markersize=20, label='Tavory

    if GC==1:
        n1=GC_optk(data,s)
        print('Geometric Complexity', n1)
        plot(n1,eigen[int(n1)],'2', markersize=20, label='Optimal

    legend();
    xlabel('Number of factors').set_color('black');
    ylabel('Explained variance ratio').set_color('black');
```

- **GC_table()**

In [20]:

```
def GC_table(data,s=64,table=False):

    '''INPUT:data-matrix Nxd
    OUTPUT: integer, optimal dimensionality reduction via Geometri
    Options: precision s=int, if table=True the function prints a

    def sum_logf(length):
        '''Return a list of given length whose entries are:
           list[i-1]=sum_a=1^i(np.log(a)+np.log(gamma(a/2)))'''
        sumlog_vec=[]
        sumlog1=0
        for a in range(1,length):
            sumlog1=sumlog1+np.log(a)+lgm(a/2)
            sumlog_vec.append(sumlog1)
        return sumlog_vec

    def log_gauss(N,d,k,lambdas,Sw):

        'lambdas=pca.explained_variance_'

        if k==d:
            lambda_bar=1
        else:
            lambda_bar=sum(lambdas[k:])/(d-k)

        Q=np.diag([1/eig for eig in lambdas[:k]]+[1/lambda_bar for
        r=d*np.log(2*np.pi)
        r=r+sum(np.log(lambdas[:k])) #kth-eigen included
        r=r+(d-k)*np.log(lambda_bar)
        r=r+np.trace(Q@Sw)
        return r*N/2

    zm_data=zeromean(data)
    pca = PCA().fit(zm_data)
    dimrange=pca.components_.shape[0] #number of features
    N=zm_data.shape[0] #number of instances
    if (dimrange != zm_data.shape[1]):
            print("attributes have linear dependencies, ther is a
            print("Solution: remove dependent variables eg: averag
            return "error";
    rotdata=changebasis(zm_data,pca.components_)
    optd=-1;
    optscore=np.Infinity
    S=np.matmul(rotdata.T,rotdata)
    sumlog_vec=sum_logf(dimrange+1)
    tab=[[]]
    for m in range(1,dimrange+1):
        r0=log_gauss(N,dimrange,m, pca.explained_variance_,S)
        r1=+m*(m+1)/4*np.log(N/(2*np.pi)) #a
        r1=r1+m*(m+2)*(m-1)/(24*N) #b
        r2=m/2*np.log(2)+m*(m+1)/4*np.log(np.pi) #c
```

```
        sumlog=sumlog_vec[m-1]
        r2=r2-sumlog
        rs=m*np.log(s*np.log(2))+(2*s+1)*m*(m-1)/4*np.log(2)
        r=r0+r1+r2+rs
        if table==True:
            vec=[m]
            vec.append(r0)#log(p)
            vec.append(r1)#model1
            vec.append(r2+rs)#log(vol)
            vec.append(rs)#log(I(s))
            vec.append(r)#geometric complexity
            tab=tab+[vec]
        if(r<optscore):
            optd=m;
            optscore=r
    if table==True:
        print(tabulate(tab[1:],headers=['log(p)','model1','log(vol
    return optd
```

- **PCA_datalen()**

In [34]:

```
def PCA_datalen(data, num=1, minka=1, tavory=1, GC=1, s=64, vec=0)

    '''INPUT data-matrix Nxd, integer num, integer s
    OUTPUT plot of the optimal dimensionality reduction obtained w
    Geometric Complexity -- in function of the dimension of the da
    Options The integer num indicate the interval between two diff
    for the Geometric Complexity algorithm.'''

    zmdata=zeromean(data)
    data_pca=PCA().fit(X=data)
    minka_k=[]
    tav_k=[]
    gc_k=[]
    for i in range(shape(zmdata)[1]+1,shape(zmdata)[0]+1,num):
        if minka==1:
            pca_mle=PCA('mle').fit(zmdata[:i])
            mc=pca_mle.components_.shape[0]
            minka_k.append(mc)
        if tavory==1:
            nml=nml_optk(zmdata[:i])
            tav_k.append(nml)
        if GC==1:
            gc=GC_optk(zmdata[:i],s)
            gc_k.append(gc)

    if minka==1:
        plot(minka_k, 'x', markersize=5,label='Minka',color="green
    if tavory==1:
        plot(tav_k,'2', markersize=5, label='Tavory nml',color="pu
    if GC==1:
        plot(gc_k, 'x', markersize=5,label='Geometric Complexity',
    legend();
    xlabel('Number of instances').set_color('black');
    ylabel('Reduced dimension').set_color('black');
    if vec==1:
        return minka_k,tav_k,gc_k
```

- **PCA_datafeat()**

In [22]:

```
def PCA_datafeat(data, num=1, minka=1, tavory=1, GC=1, s=64, vec=0

    '''INPUT data-matrix Nxd, integer num, integer s
    OUTPUT plot of the optimal dimensionality reduction obtained w
    Geometric Complexity -- in function of the number of features.
    Options The integer num indicate the interval between two diff
    for the Geometric Complexity algorithm.'''

```

```
    zmdata=zeromean(data)
    data_pca=PCA().fit(X=data)
    minka_k=[]
    tav_k=[]
    gc_k=[]
    for i in range(int(shape(zmdata)[1]/2),shape(zmdata)[1],num):
        if minka==1:
            pca_mle=PCA('mle').fit(zmdata[:,:i])
            mc=pca_mle.components_.shape[0]
            minka_k.append(mc)
        if tavory==1:
            nml=nml_optk(zmdata[:,:i])
            tav_k.append(nml)
        if GC==1:
            gc=GC_optk(zmdata[:,:i],s)
            gc_k.append(gc)

    if minka==1:
        plot(minka_k, 'x', markersize=5,label='Minka',color="green
    if tavory==1:
        plot(tav_k,'2', markersize=5, label='Tavory nml',color="pu
    if GC==1:
        plot(gc_k, 'x', markersize=5,label='Geometric Complexity',
    legend();
    xlabel('Number of instances').set_color('black');
    ylabel('Reduced dimension').set_color('black');
    if vec==1:
        return np.array(minka_k),np.array(tav_k),np.array(gc_k)
```

Test-data ◇

We consider datasets obtained from multivariate sources and s.t. $N > 3d$.

After uploadiing and pre-processing the data we proceed as it follows:

1. compute the optimal dimensionality reduction via Geometric Complexity algorithm `GC_optK` :
    - when the data-features display the same scale, we use the s-parameter obtained with the `precision()` function;
    - when the data features have different scales, we first obtain the int-covariance matrix with the `data_int()` function;
    - plot the GC_k in function of the s-parameter using `GC_s` ;
2. compare the obtained result with the ones obtained via Minka's and Tavory's algorithm using `PCA_test()` and/or `GC_table()` ;
3. plot the dimensionality reductions obtained with the three algorithms with respect to the dataset-length using `PCA_datalen()` ;
4. plot the dimensionality reductions obtained with the three algorithms with respect to the number of features in the data using `PCA_datafeat()` .

We start by creating ad-ok datasets using the `data_Tavory()` function from a given dataset and testing the three algorithms;

- Test-data from Sonar-data;

then, we proceed with real datasets:

- Sonar-data;
- Cnae data: highly sparse dataset;
- Drug-data: low-dim dataset;
- Music-data: features with different scales;
- Ceramic-composition-data;

Test-data from Sonar-data △

In [23]:
```
my_data_path='Data/sonar.all-data.csv'
my_data = genfromtxt('Data/sonar.all-data.csv', delimiter=',')

### removing the class
mdata=my_data[:,:-1]

print('mdata:',mdata.shape, mdata.dtype)
```

```
mdata: (208, 60) float64
```

Ad-ok datasets from my_data :

In [26]:
```
err=1e-6
zmdata_k5=data_Tavory(my_data,5,err)
zmdata_k10=data_Tavory(my_data,10,err)
zmdata_k30=data_Tavory(my_data,30,err)
```

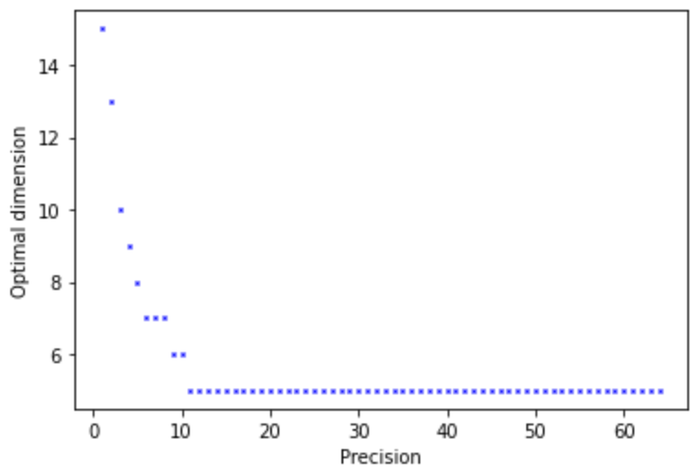
In [11]:
```
my_data_test_path='Data/Minkatest.csv'
my_data_test = genfromtxt('Data/Minkatest.csv', delimiter=',')
print('my_data_test:',my_data_test.shape, my_data_test.dtype)
```

```
my_data_test: (1000, 100) float64
```

1. GC optimal dimensionality reduction and precision

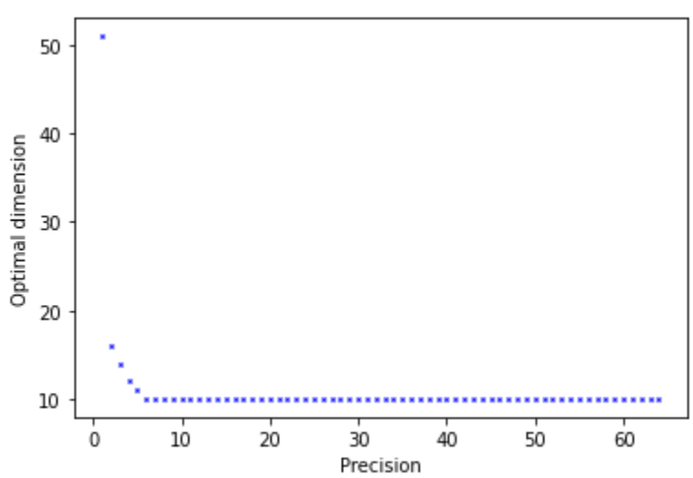
In [42]:
```
precision(zmdata_k5,4),GC_optk(zmdata_k5,precision(zmdata_k5,4)),G
```

Out[42]:
```
(14, 5, None)
```



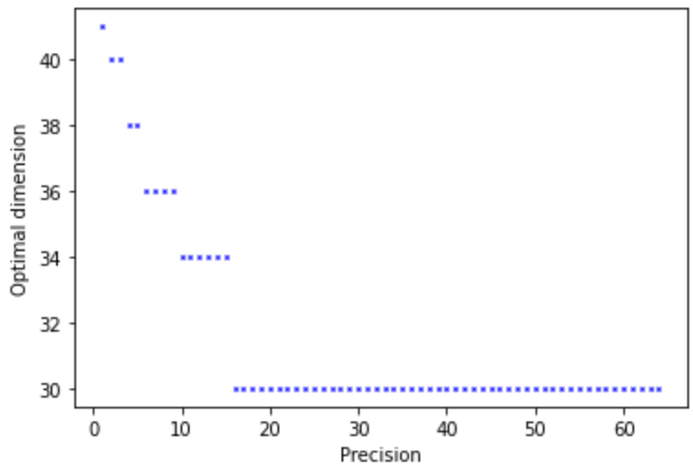
In [43]:
```
precision(zmdata_k10,4),GC_optk(zmdata_k10,precision(zmdata_k10,4)
```

Out[43]:
```
(15, 10, None)
```



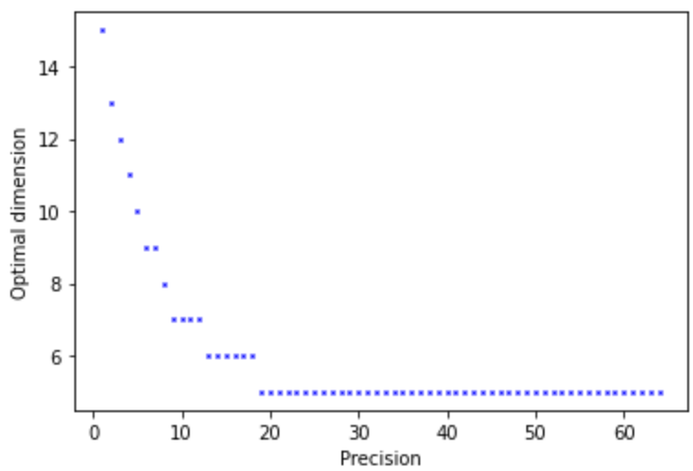
In [44]:
```
1 precision(zmdata_k30,4),GC_optk(zmdata_k30,precision(zmdata_k30,4)
```

Out[44]: (17, 30, None)



In [48]:
```
1 precision(my_data_test,4),GC_optk(my_data_test,precision(my_data_t
```

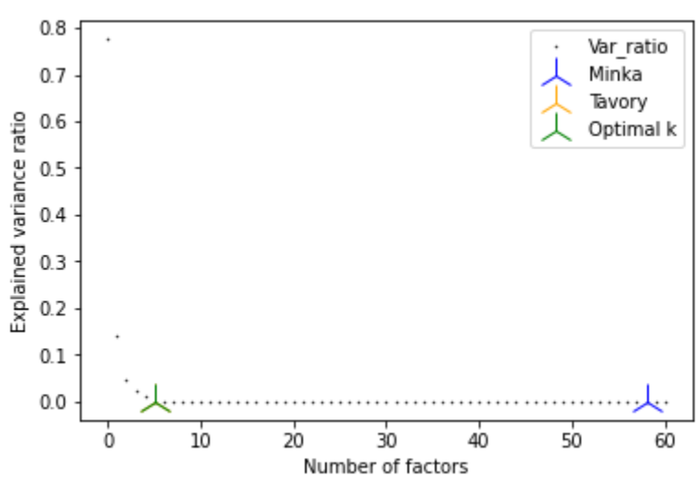
Out[48]: (18, 6, None)



2. Scree plot and algorithms-comparing

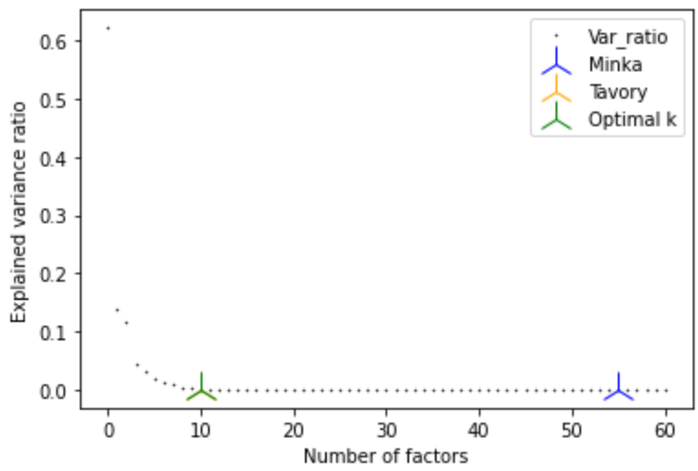
In [27]:
```
1 PCA_test(zmdata_k5,14)
```

```
Minka Complexity 58
Tavory Complexity 5
Geometric Complexity 5
```



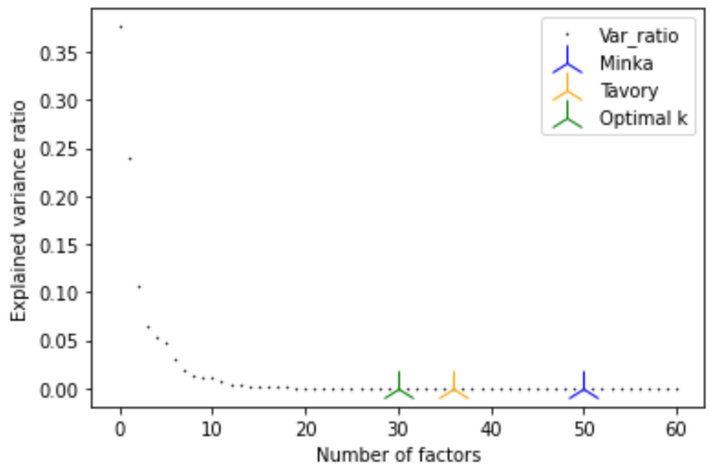
In [46]:
```
1 PCA_test(zmdata_k10,15)
```

```
Minka Complexity 55
Tavory Complexity 10
Geometric Complexity 10
```



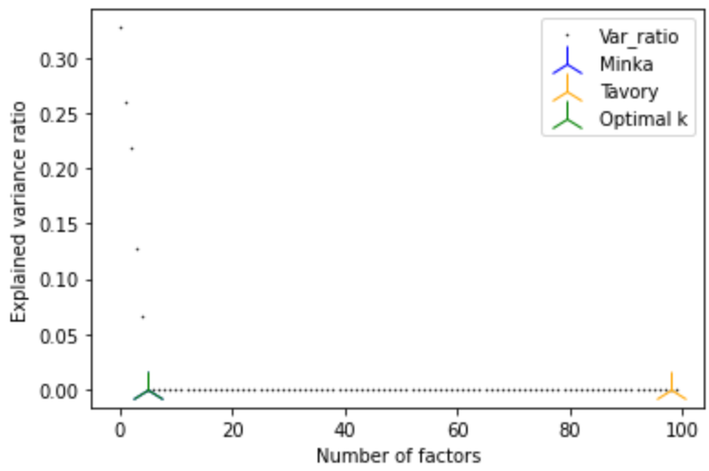
In [45]:
```
1 PCA_test(zmdata_k30,17)
```

```
Minka Complexity 50
Tavory Complexity 36
Geometric Complexity 30
```



In [49]:
```
1 PCA_test(my_data_test,18)
```

```
Minka Complexity 5
Tavory Complexity 98
Geometric Complexity 5
```



Sonar-data △

In [12]:
```
1 my_data_path='Data/sonar.all-data.csv'
```

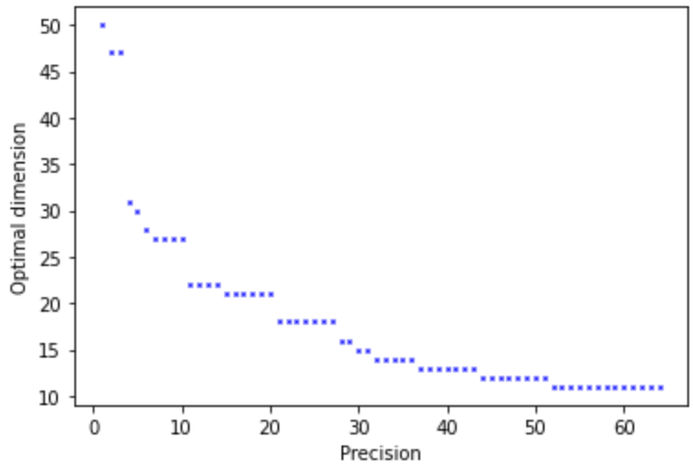
```
2 my_data = genfromtxt('Data/sonar.all-data.csv', delimiter=',')
3
4 #removing class
5 mdata=my_data[:,:-1]
6
7 print('mdata:',mdata.shape, mdata.dtype)
```

```
mdata: (208, 60) float64
```

1. GC optimal dimensionality reduction and precision

In [55]:
```
1 precision(my_data,4),GC_optk(my_data,precision(my_data,4)),GC_s(my
```

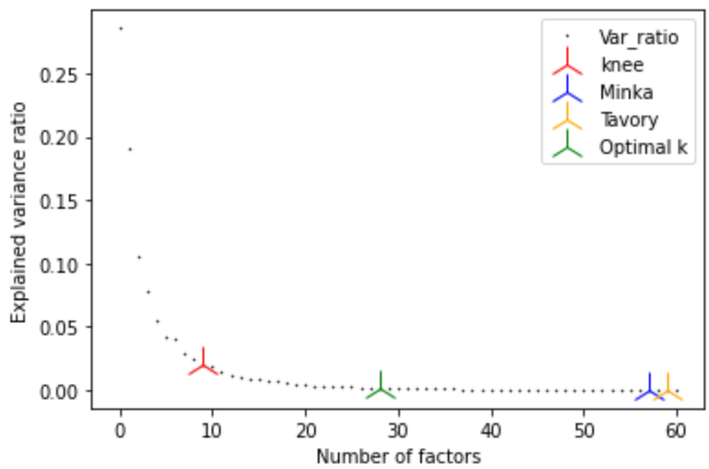
Out[55]:
```
(16, 21, None)
```



2. Scree plot and algorithms-comparing

In [138]:
```
1 PCA_test(my_data,s=6)
```

```
knee 9.0
Minka Complexity 57
Tavory Complexity 59
Geometric Complexity 28
```

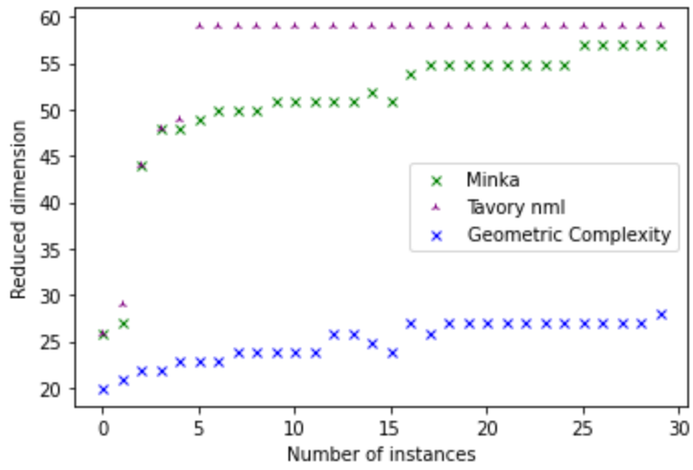


In [137]:
```
1 GC_table(my_data,s=6,table=True)
```

	log(p)	model1	log(vol(s))	log(I(s))	GC
1	1.3008e+06	1.74983	1.77182	1.42525	1.3008e+06
2	1.29929e+06	5.25109	8.50068	7.35595	1.2993e+06
3	1.2983e+06	10.505	20.0227	17.7921	1.29833e+06

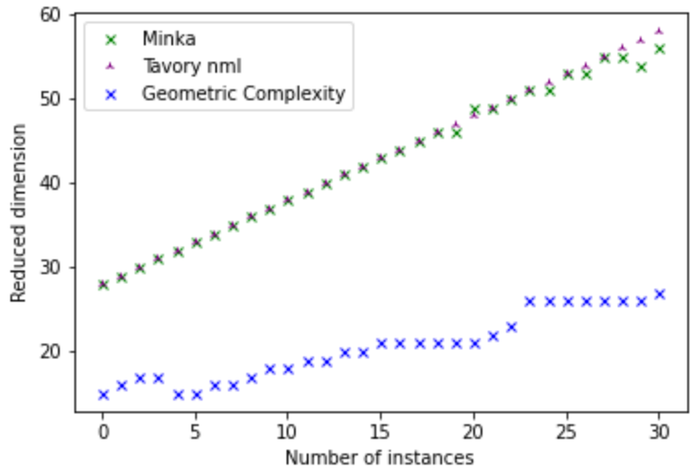
3. Optimal dimensionality in function of data-length N

In [85]:
```
1 PCA_datalen(my_data,num=5,s=6)
```



4. Optimal dimensionality in function of data-features d

In [87]:
```
1 PCA_datafeat(my_data,s=6)
```



Highly sparse data - source and description (https://archive.ics.uci.edu/ml/datasets/cnae-9) △

In [13]:
```
1 #loading data
2 cnae_path='Data/CNAE-9.data'
3 cnae_data = genfromtxt(cnae_path, delimiter=',')
4 cn = open(cnae_path, "r")
5 #print(cn.read())
6 cn.close()
```

In [60]:
```
1 cnae_data=cnae_data[:,[i for i in range(1,cnae_data.shape[1])]] #r
```

In [61]:
```
1 cnae_data.shape
```

Out[61]: (1080, 856)

1. GC optimal dimensionality reduction and precision

In [65]:
```
precision(cnae_data,0)
```

Out[65]: 1

In [66]:
```
GC_optk(cnae_data,1)
```

Out[66]: 622

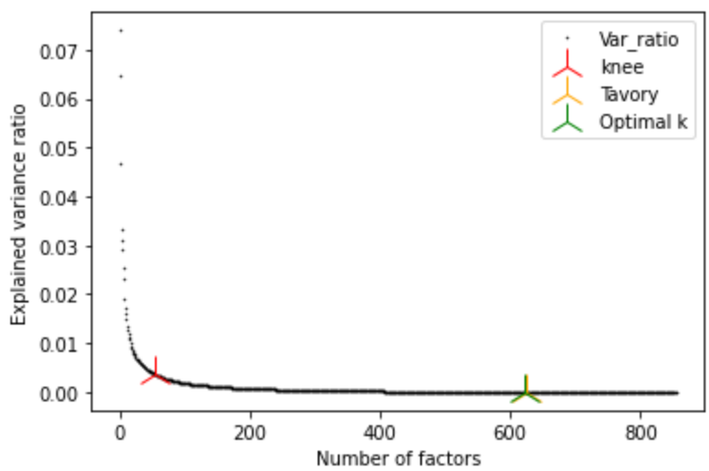
In [ ]:
```
#GC_s(cnae_data)
```

2. Scree plot and algorithms-comparing

Minka's built-in algorithm goes in overflow. I exclude it from the test.

In [67]:
```
PCA_test(cnae_data,minka=0,s=1)
```

```
knee 55.0
Tavory Complexity 624
Geometric Complexity 622
```

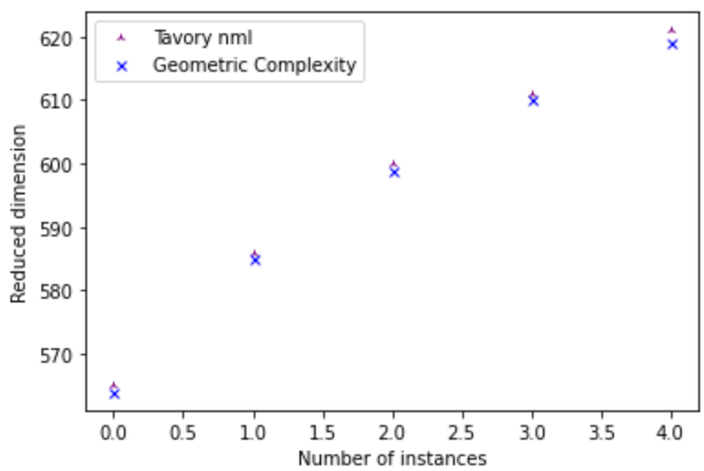


3. Optimal dimensionality in function of data-length N

In [69]:
```
shape(cnae_data)
```

Out[69]: (1080, 856)

In [133]:
```
PCA_datalen(cnae_data,num=50,s=1,minka=0)
```



4. Optimal dimensionality in function of data-features d

In [ ]:
```
PCA_datafeat(cnae_data,num=20,s=1,minka=0)
```

[Drug-data - source and description (https://archive.ics.uci.edu/ml/datasets/Drug+consumption+%28quantified%29) △](#)

Just the first 13 features are not empty.

In [14]:
```
#loading data
drug_path='Data/drug_consumption.data'
drug_data = genfromtxt(drug_path, delimiter=',')
dr=open(drug_path,'r')
print(dr.read())
dr.close()
```

```
1,0.49788,0.48246,-0.05921,0.96082,0.12600,0.31287,-0.57545,-0.5833
1,-0.91699,-0.00665,-0.21712,-1.18084,CL5,CL2,CL0,CL2,CL6,CL0,CL5,CL
0,CL0,CL0,CL0,CL0,CL0,CL0,CL0,CL0,CL2,CL0,CL0
2,-0.07854,-0.48246,1.98437,0.96082,-0.31685,-0.67825,1.93886,1.4353
3,0.76096,-0.14277,-0.71126,-0.21575,CL5,CL2,CL2,CL0,CL6,CL4,CL6,CL3,
CL0,CL4,CL0,CL2,CL0,CL2,CL3,CL0,CL4,CL0,CL0
3,0.49788,-0.48246,-0.05921,0.96082,-0.31685,-0.46725,0.80523,-0.8473
2,-1.62090,-1.01450,-1.37983,0.40148,CL6,CL0,CL0,CL0,CL6,CL3,CL4,CL0,
CL0,CL0,CL0,CL0,CL0,CL0,CL0,CL1,CL0,CL0,CL0
4,-0.95197,0.48246,1.16365,0.96082,-0.31685,-0.14882,-0.80615,-0.0192
8,0.59042,0.58489,-1.37983,-1.18084,CL4,CL0,CL0,CL3,CL5,CL2,CL4,CL2,C
L0,CL0,CL0,CL2,CL0,CL0,CL0,CL0,CL2,CL0,CL0
5,0.49788,0.48246,1.98437,0.96082,-0.31685,0.73545,-1.63340,-0.4517
4,-0.30172,1.30612,-0.21712,-0.21575,CL4,CL1,CL1,CL0,CL6,CL3,CL6,CL0,
CL0,CL1,CL0,CL0,CL1,CL0,CL0,CL2,CL2,CL0,CL0
6,2.59171,0.48246,-1.22751,0.24923,-0.31685,-0.67825,-0.30033,-1.5552
1,2.03972,1.63088,-1.37983,-1.54858,CL2,CL0,CL0,CL0,CL6,CL0,CL4,CL0,C
L0,CL0,CL0,CL0,CL0,CL0,CL0,CL0,CL0,CL6,CL0,CL0
7,1.09449,-0.48246,1.16365,-0.57009,-0.31685,-0.46725,-1.09207,-0.451
```

In [71]:
```
drug_data[:,1:14]
```

Out[71]:
```
array([[ 0.49788,  0.48246, -0.05921, ..., -0.21712, -1.18084,      n
an],
       [-0.07854, -0.48246,  1.98437, ..., -0.71126, -0.21575,      n
an],
       [ 0.49788, -0.48246, -0.05921, ..., -1.37983,  0.40148,      n
an],
       ...,
       [-0.07854,  0.48246,  0.45468, ...,  0.52975, -0.52593,      n
an],
       [-0.95197,  0.48246, -0.61113, ...,  1.29221,  1.2247 ,      n
an],
       [-0.95197, -0.48246, -0.61113, ...,  0.88113,  1.2247 ,      n
an]])
```

In [72]:
```
drug_data=drug_data[:,1:13] #cutting class and missing entries
```

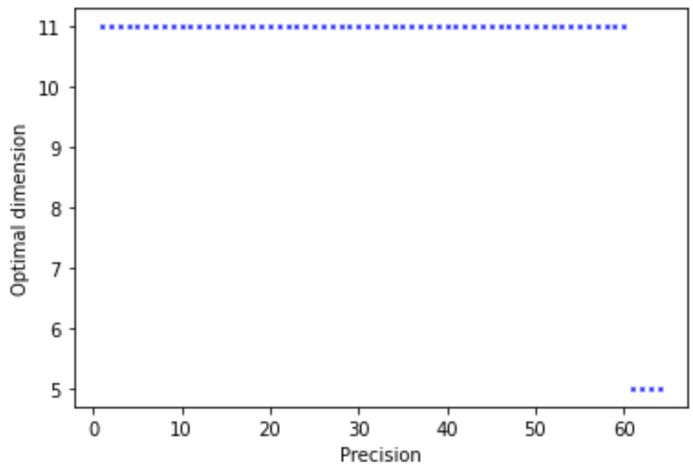
In [62]:
```
shape(drug_data)
```

Out[62]:
```
(1885, 12)
```

1. GC optimal dimensionality reduction and precision

In [73]:
```
1 precision(drug_data,5),GC_optk(drug_data,precision(drug_data,5)),G
```

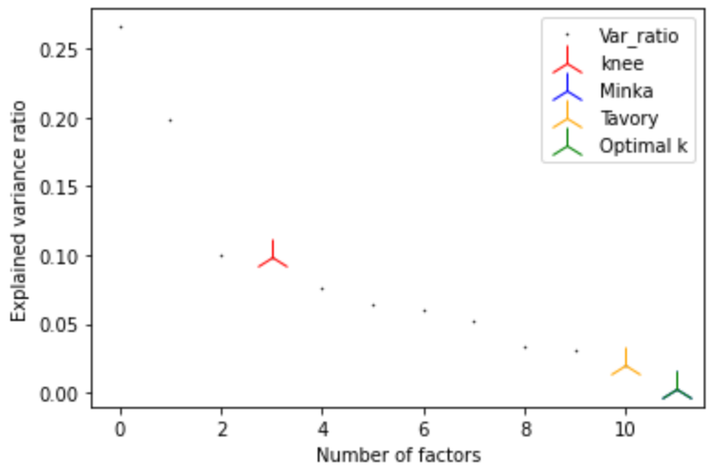
Out[73]: (22, 11, None)



2. Scree plot and algorithms-comparing

In [77]:
```
1 PCA_test(drug_data,s=22)
```

```
knee 3.0
Minka Complexity 11
Tavory Complexity 10
Geometric Complexity 11
```

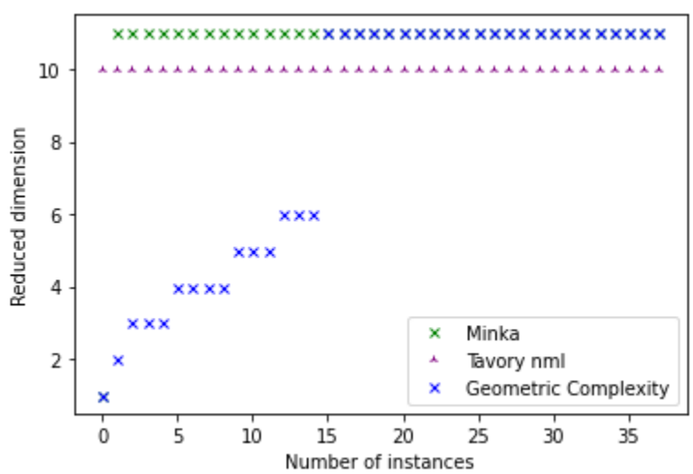


3. Optimal dimensionality in function of data-length N

In [78]:
```
1 shape(drug_data)
```

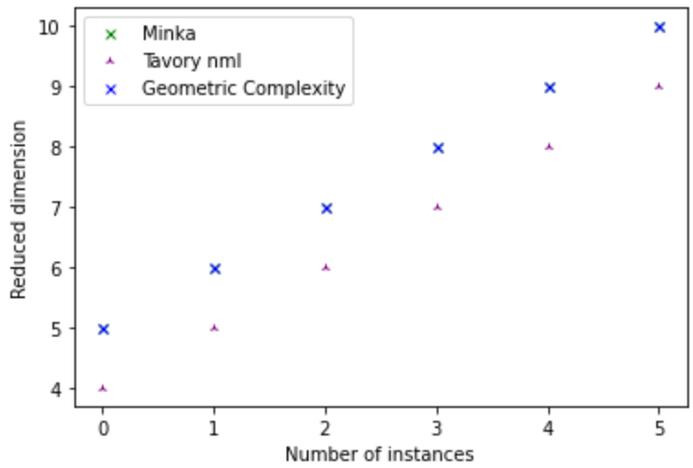
Out[78]: (1885, 12)

In [81]:
```
1 PCA_datalen(drug_data,num=50,s=22)
```



4. Optimal dimensionality in function of data-features d

In [84]:
```
1 PCA_datafeat(drug_data,s=22)
```



Music data - source and description (https://archive.ics.uci.edu/ml/datasets/Geographical+Original+of+Music) △

In [88]:
```
1 #loading data
2 music_path='Data/default_features_1059_tracks.txt'
3 music_data = genfromtxt(music_path, delimiter=',')
4 ms=open(music_path,'r')
5 print(ms.read())
6 ms.close()
```

```
7.161286,7.835325,2.911583,0.984049,-1.499546,-2.094097,0.576,-1.2056
71,1.849122,-0.425598,-0.105672,1.728885,1.788986,0.849798,-1.109353,
0.537904,-0.115368,5.069512,6.00771,0.820869,0.89619,0.131699,0.85928
6,2.059065,0.266773,1.192932,-1.421091,2.128661,-1.288109,1.458738,-
0.734508,-0.092678,-0.571314,-0.142634,2.748619,3.099077,0.31727,-0.1
3058,2.048282,-0.173489,0.324616,-0.300817,0.471089,-0.538577,-0.9791
24,-0.679165,0.135963,-1.094049,-0.072197,-0.752002,-0.660715,1.31972
9,1.094839,-0.937659,-0.895371,-0.734962,0.441859,0.389178,-0.94458
4,-0.04361,-1.504263,0.351267,-1.018726,-0.174878,-1.089543,-0.6688
4,-0.914772,-0.83625,-15.75,-47.95
0.225763,-0.094169,-0.603646,0.497745,0.874036,0.29028,-0.077659,-0.8
87385,0.432062,-0.093963,0.029105,0.407297,-0.034418,-6.07E-4,-1.5877
12,-0.134767,0.67905,0.867759,0.549205,-0.357172,-0.578459,0.293603,-
0.369997,-0.360397,-0.088276,-0.68448,-0.420736,0.263,0.074617,0.2779
73,0.468588,0.978996,0.586847,0.760345,1.400111,0.943587,-0.402494,0.
058298,-0.221967,-0.302481,-0.539966,0.179847,-0.634147,-0.252916,-0.
441251,-0.342925,0.628843,0.212837,-0.038171,-0.44029,-0.157062,1.627
259,1.989545,-0.357803,-0.176835,0.406589,-0.623764,-0.653021,-0.0826
45,-0.947933,-0.495712,-0.465077,-0.157861,-0.157189,0.380951,1.08847
```

In [89]:
```
1 shape(music_data)
```

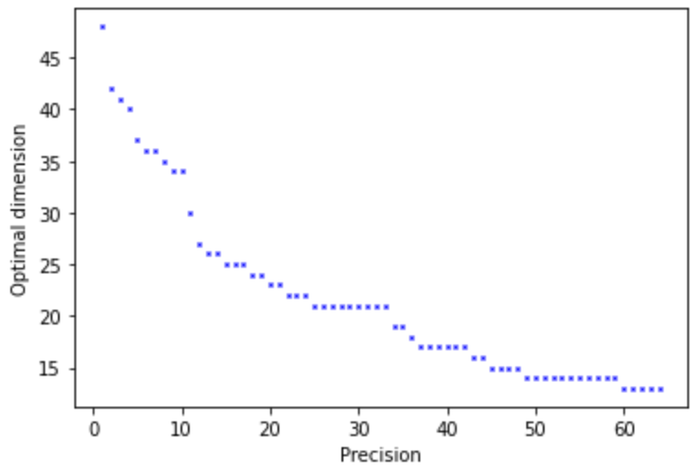
Out[89]:
```
(1059, 70)
```

In [90]:
```
1 music_datacut=music_data[:,:68]#last two features have very differ
```

1. GC optimal dimensionality reduction and precision

In [93]:
```
precision(music_datacut,6,b=1),GC_optk(music_datacut,precision(mus
```

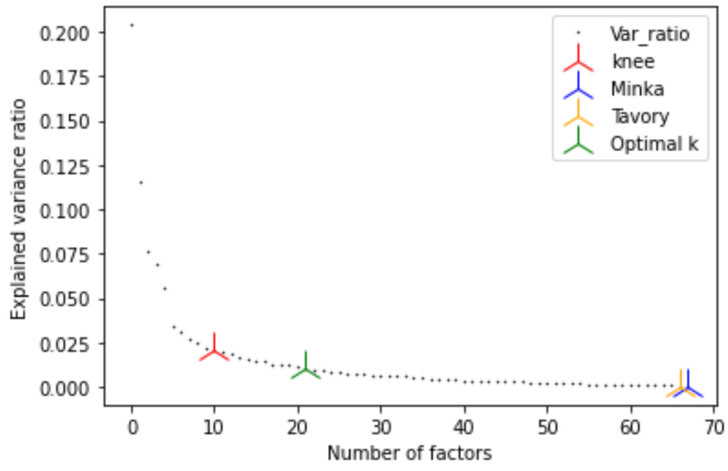
Out[93]: (27, 21, None)



2. Scree plot and algorithms-comparing

In [94]:
```
PCA_test(music_datacut,s=27)
```

```
knee 10.0
Minka Complexity 67
Tavory Complexity 66
Geometric Complexity 21
```

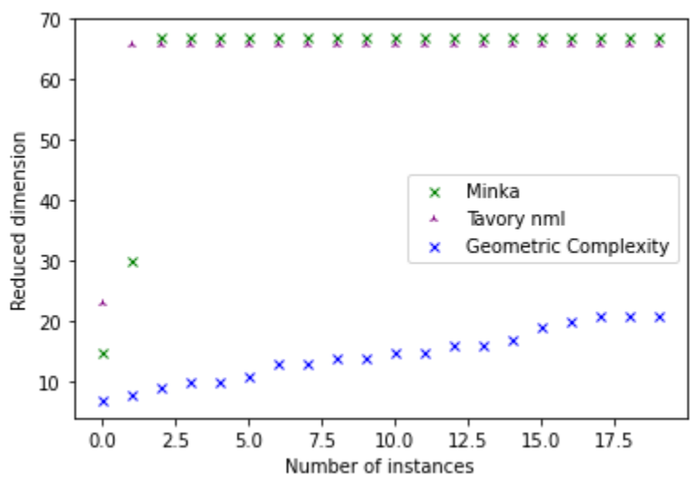


3. Optimal dimensionality in function of data-length N

In [95]:
```
shape(music_datacut)
```

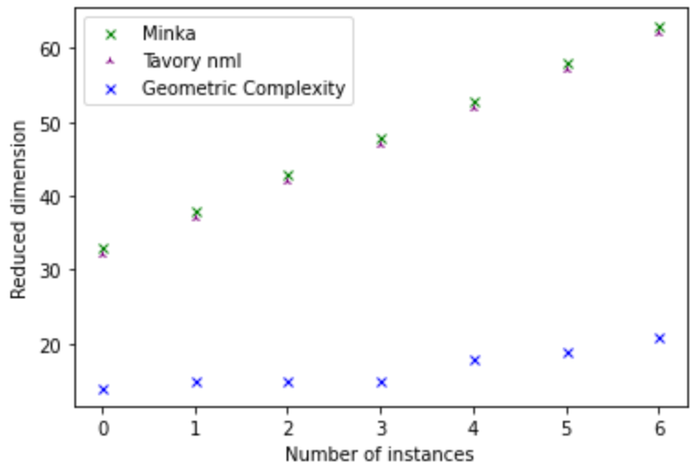
Out[95]: (1059, 68)

In [97]:
```
PCA_datalen(music_datacut,num=50,s=27)
```



4. Optimal dimensionality in function of data-features d

In [99]:
```
1 PCA_datafeat(music_datacut,num=5,s=27)
```



We are now going to analyze the same dataset including the two features with different precision and using the function `data_int()` for pre-processing the data and determining the parameter s.

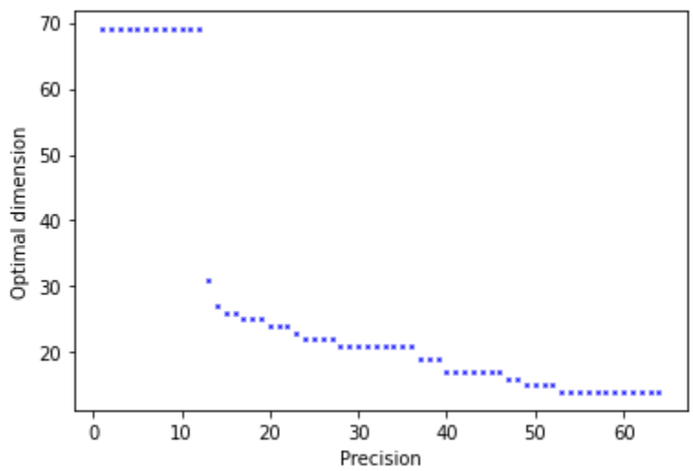
I.e. we rescale the covariance matrix s.t. every features is written as multiple of a fundamental precision -- 1e6 for the first 68 features, 1e2 for the last 2. We are then going to analyse the resulting integers-matrix. We remark that the optimal-dimensionalities computed via Tavory and Minka algorithms o the original matrix and on the rescaled one are equivalent.

In [106]:
```
1 music_data = genfromtxt(music_path, delimiter=',')
2
3 music_dataint=music_data[:]
4 music_dataint[:,:68]=music_dataint[:,:68]*1e6
5 music_dataint[:,69:]=music_dataint[:,69:]*1e2
6 music_dataint
7
8 music_dataint=data_int(music_dataint,1)
```

1. GC optimal dimensionality reduction and precision

In [107]:
```
1 precision(music_dataint,0,b=1),GC_optk(music_dataint,precision(mus
```

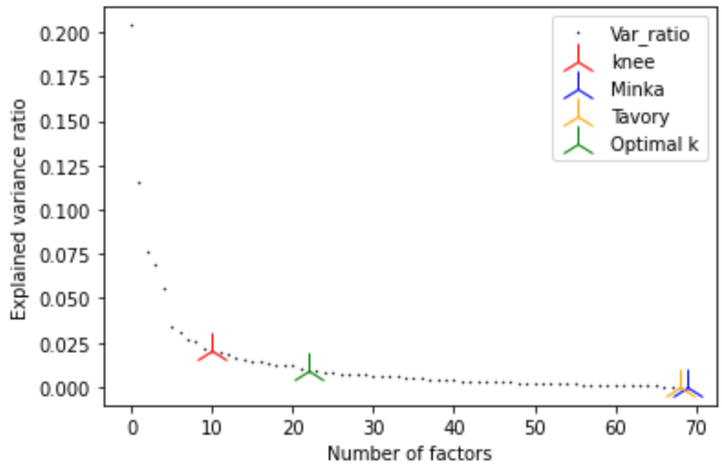
Out[107]:
```
(27, 22, None)
```



2. Scree plot and algorithms-comparing

In [108]:
```
1 PCA_test(music_dataint,s=27)
```

```
knee 10.0
Minka Complexity 69
Tavory Complexity 68
Geometric Complexity 22
```

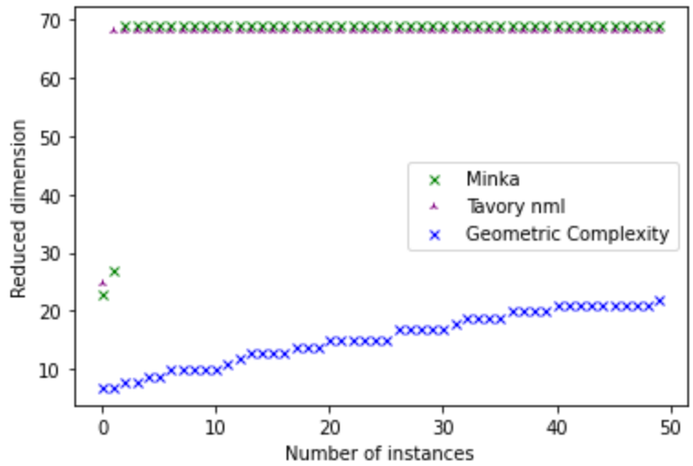


3. Optimal dimensionality in function of data-length N

In [110]:
```
1 shape(music_dataint)
```

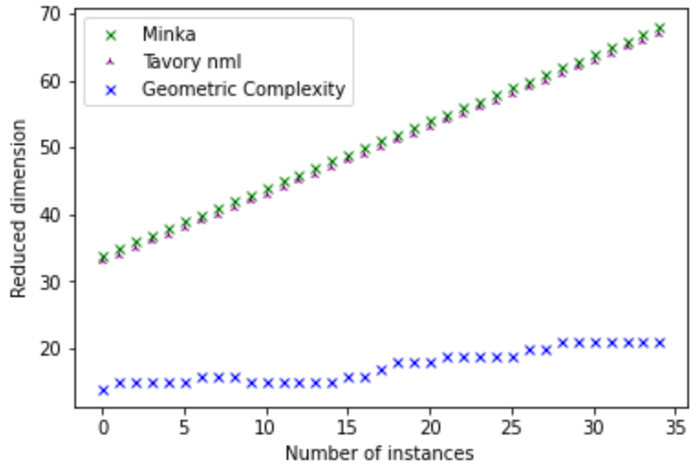
Out[110]: (1059, 70)

In [111]:
```
1 PCA_datalen(music_dataint,num=20,s=27)
```



4. Optimal dimensionality in function of data-features d

In [113]:
```
1 PCA_datafeat(music_dataint,s=27)
```



Ceramic composition data - source and description (https://archive.ics.uci.edu/ml/datasets/Chemical+Composition+of+Ceramic+Samples) △

In [15]:
```
1 #loading data
2 ceramic_path='Data/Chemical_Composion_Ceramic.csv'
```

```
ceramic_data = genfromtxt(ceramic_path, delimiter=',')
cer=open(ceramic_path,'r')
print(cer.read())
cer.close()
```

```
Ceramic Name,Part,Na2O,MgO,Al2O3,SiO2,K2O,CaO,TiO2,Fe2O3,MnO,CuO,ZnO,
PbO2,Rb2O,SrO,Y2O3,ZrO2,P2O5
FLQ-1-b,Body,0.62 ,0.38 ,19.61 ,71.99 ,4.84 ,0.31 ,0.07 ,1.18 ,630,1
0,70,10,430,0,40,80,90
FLQ-2-b,Body,0.57 ,0.47 ,21.19 ,70.09 ,4.98 ,0.49 ,0.09 ,1.12 ,380,2
0,80,40,430,-10,40,100,110
FLQ-3-b,Body,0.49 ,0.19 ,18.60 ,74.70 ,3.47 ,0.43 ,0.06 ,1.07 ,420,2
0,50,50,380,40,40,80,200
FLQ-4-b,Body,0.89 ,0.30 ,18.01 ,74.19 ,4.01 ,0.27 ,0.09 ,1.23 ,460,2
0,70,60,380,10,40,70,210
FLQ-5-b,Body,0.03 ,0.36 ,18.41 ,73.99 ,4.33 ,0.65 ,0.05 ,1.19 ,380,4
0,90,40,360,10,30,80,150
FLQ-6-b,Body,0.62 ,0.18 ,18.82 ,73.79 ,4.28 ,0.30 ,0.04 ,0.96 ,350,2
0,80,10,390,10,40,80,130
FLQ-7-b,Body,0.45 ,0.33 ,17.65 ,74.99 ,3.53 ,0.70 ,0.07 ,1.28 ,650,2
0,90,90,410,30,30,90,140
FLQ-8-b,Body,0.59 ,0.45 ,21.42 ,71.46 ,3.47 ,0.35 ,0.05 ,1.20 ,500,1
0,70,50,380,70,40,80,440
FLQ-9-b,Body,0.42 ,0.53 ,23.12 ,67.41 ,3.81 ,0.74 ,0.16 ,2.81 ,340,4
```

Data pre-processing

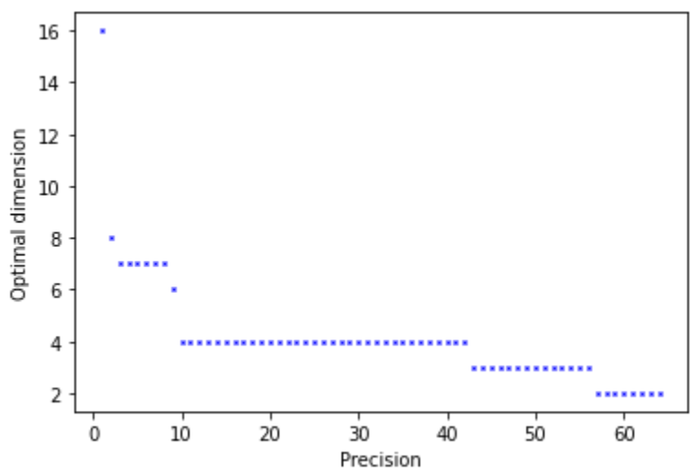
In [115]:
```
#add ppm (*1e-4) from MnO on
#separating body-glaze in two distinct dataset
#removing classes
```

In [116]:
```
ceramic_dataint=ceramic_data[:]#dataset cloning
ceramic_dataint=ceramic_dataint[1:]#reoving labelling
for i in range(2,10):
    ceramic_dataint[:,i]=ceramic_dataint[:,i]*1e2
ceramic_dataint_body=ceramic_dataint[:44,2:]
ceramic_dataint_body=data_int(ceramic_dataint_body,1)
ceramic_dataint_glaze=ceramic_dataint[44:,2:]
ceramic_dataint_glaze=data_int(ceramic_dataint_glaze,1)
```

1. GC optimal dimensionality reduction and precision

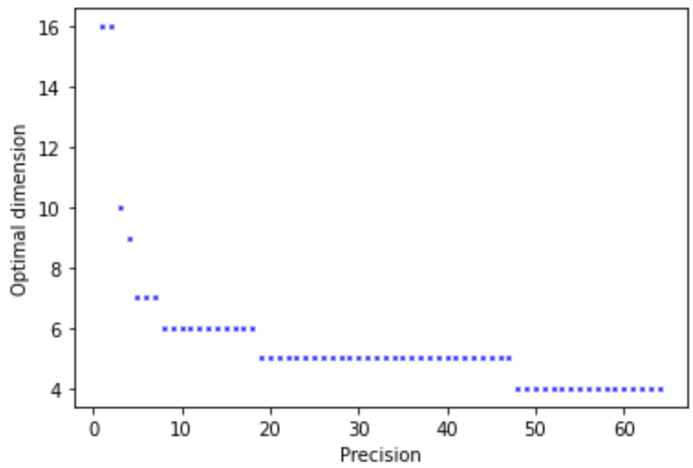
In [124]:
```
precision(ceramic_dataint_body,0),GC_optk(ceramic_dataint_body,pre
```

Out[124]: (10, 4, None)



In [122]:
```
1 precision(ceramic_dataint_glaze,0),GC_optk(ceramic_dataint_glaze,p
```

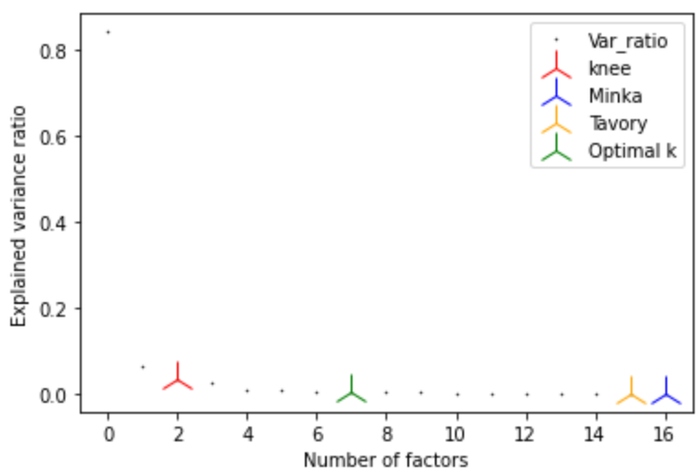
Out[122]: (11, 6, None)



2. Scree plot and algorithms-comparing

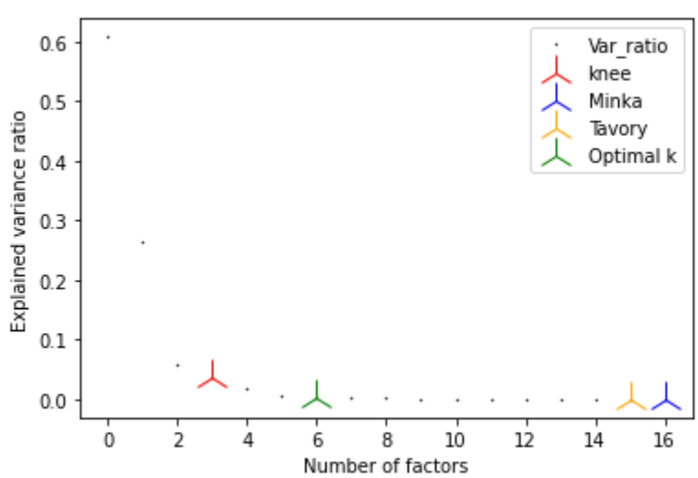
In [125]:
```
1 PCA_test(ceramic_dataint_body,s=8)
```

```
knee 2.0
Minka Complexity 16
Tavory Complexity 15
Geometric Complexity 7
```



In [123]:
```
1 PCA_test(ceramic_dataint_glaze,s=11)
```

```
knee 3.0
Minka Complexity 16
Tavory Complexity 15
Geometric Complexity 6
```

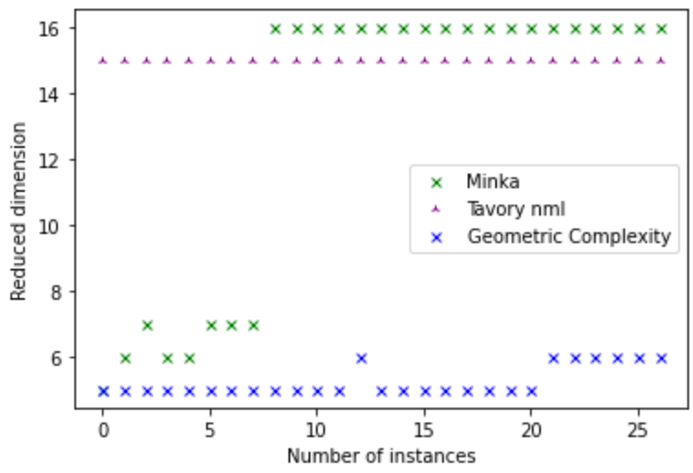


3. Optimal dimensionality in function of data-length N

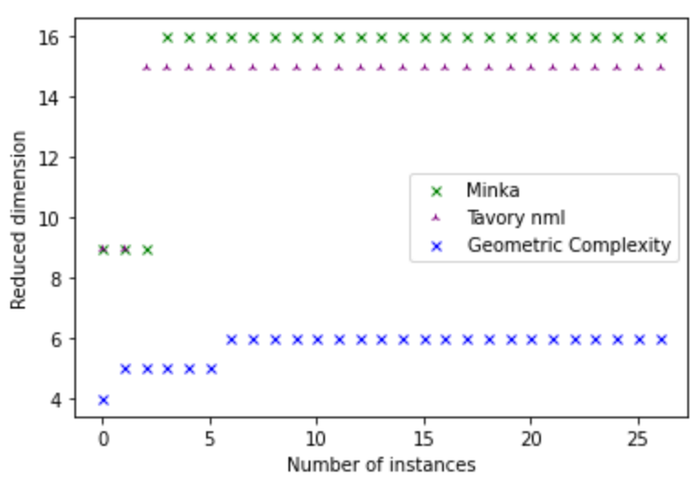
In [126]:
```
1 shape(ceramic_dataint_body),shape(ceramic_dataint_glaze)
```

Out[126]: ((44, 17), (44, 17))

In [127]:
```
1 PCA_datalen(ceramic_dataint_body,s=8)
```

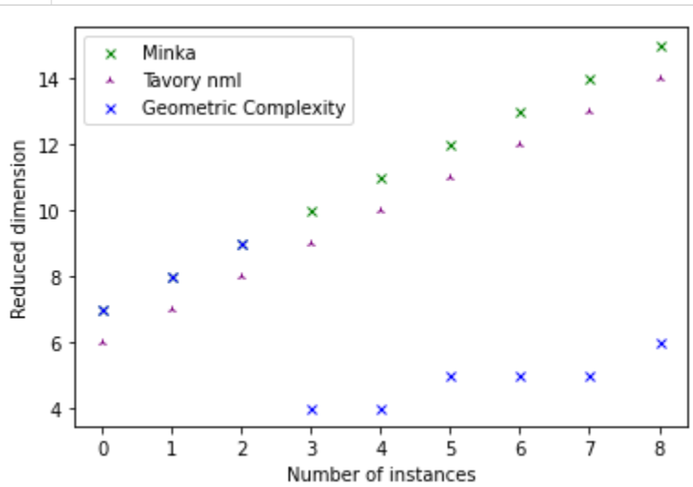


In [128]:
```
1 PCA_datalen(ceramic_dataint_glaze,s=11)
```

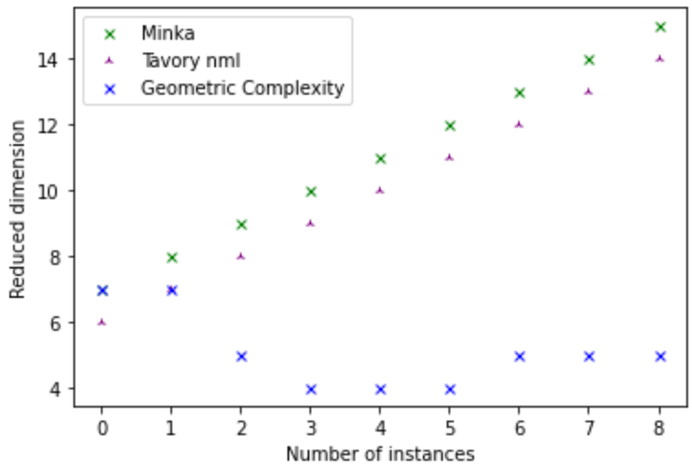


4. Optimal dimensionality in function of data-features d

In [129]:
```
1 PCA_datafeat(ceramic_dataint_body,s=8)
```



In [130]:
```
1 PCA_datafeat(ceramic_dataint_glaze,s=11)
```



Comments and future work ◇

- The *Geometric Complexity algorithm* (GC-alg.), in general underestimates the optimal dimensionality reduction with respect to the two existing algorithms of Tavory and Minka. Indeed, in the GC_alg. the model length that plays an important role in the two-part-code optimisation. When the model is more complex -- high data-precision for example, i.e. high s -- more the model-code-legth weighs in the total code-length. Indeed in general Minka's and/or Tavory's results are recovered for low s by the GC-alg.

- We gave and euristic procedure for determining the s-parameter, which deserves furter tests. For example, by using the `data_Tavory()` and changing the noise-variance.
- The GC-algorithm has a slower convergence to a 'stable' value for the optimal-dimensionality in function of the dataset-dimesion as seen in the plots (3) and (4), result that justifies the relevant underestimation obtained for small dataset. The algorithm deserves further tests on higher-dimension datasets.

Appendix ◇

Data-processing routine

1. GC optimal dimensionality reduction and precision

In [ ]:
```
1 precision(data,sign),GC_optk(data,precision(data,sign)),GC_s(data)
```

2. Scree plot and algorithms-comparing

In [ ]:
```
1 PCA_test(data,s=)
```

3. Optimal dimensionality in function of data-length N

In [ ]:
```
1 shape(data)
```

In [ ]:
```
1 PCA_datalen(data,num=,s=)
```

4. Optimal dimensionality in function of data-features d

In [ ]:
```
1 PCA_datafeat(data,num=,s=)
```

In [ ]:
```
1
```

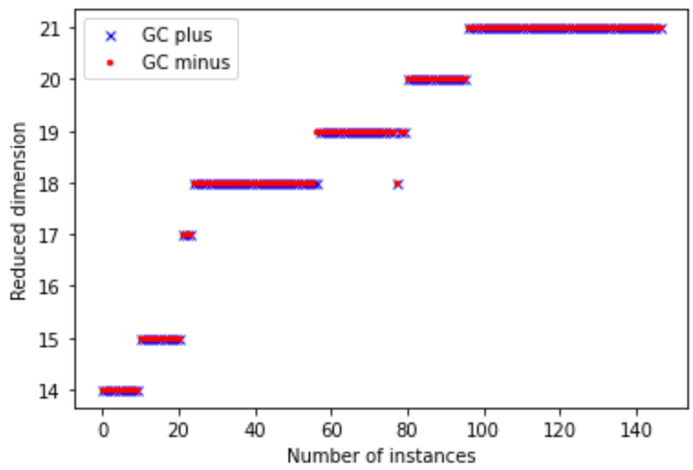
Plus-minus test /24N test

In [33]:
```
def GC_optk_test(data,s=64,b=0):

    def sum_logf(length):
        '''Return a list of given length whose entries are:
           list[i-1]=sum_a=1^i(np.log(a)+np.log(gamma(a/2)))'''
        sumlog_vec=[]
        sumlog1=0
        for a in range(1,length):
            sumlog1=sumlog1+np.log(a)+lgm(a/2)
            sumlog_vec.append(sumlog1)
        return sumlog_vec

    def log_gauss(N,d,k,lambdas,Sw):

        'lambdas=pca.explained_variance_'
        #k=k-1
        if k==d:
            lambda_bar=1
        else:
            lambda_bar=sum(lambdas[k:])/(d-k)
        #print('k:',k,'lambda_bar',lambda_bar)
        Q=np.diag([1/eig for eig in lambdas[:k]]+[1/lambda_bar for
        r=d*np.log(2*np.pi)
        r=r+sum(np.log(lambdas[:k])) #kth-eigen included
        r=r+(d-k)*np.log(lambda_bar)
        r=r+np.trace(Q@Sw)
        return r*N/2

    zm_data=zeromean(data)
    pca = PCA().fit(zm_data)
    dimrange=pca.components_.shape[0] #number of features
    N=zm_data.shape[0] #number of instances
    if (dimrange != zm_data.shape[1]):
            print("attributes have linear dependencies, ther is a
            print("Solution: remove dependent variables eg: averag
            return "error";
    rotdata=changebasis(zm_data,pca.components_)
    optd=-1;
    optscore=np.Infinity
    S=np.matmul(rotdata.T,rotdata)
    sumlog_vec=sum_logf(dimrange+1)
    for m in range(1,dimrange+1):
        r=log_gauss(N,dimrange,m, pca.explained_variance_,S)
        r=r+m*(m+1)/4*np.log(N/(2*np.pi)) #a
        r=r+(-1)*b*m*(m+2)*(m-1)/(24*N) #b
        r=r-m/2*np.log(2)+m*(m+1)/4*np.log(np.pi) #c
        sumlog=sumlog_vec[m-1]
        r=r-sumlog
        r=r+m*np.log(s*np.log(2))+(2*s+1)*m*(m-1)/4*np.log(2)
        if(r<optscore):
            optd=m;
            optscore=r

    return optd
```

In [76]:
```
def PCA_datalen_test(data, num=1, s=64, vec=0, Nmax=0):

    '''INPUT data-matrix Nxd, integer num, integer s
    OUTPUT plot of the optimal dimensionality reduction obtained w
    Geometric Complexity -- in function of the dimension of the da
    Options The integer num indicate the interval between two diff
    for the Geometric Complexity algorithm.'''

    zmdata=zeromean(data)
    data_pca=PCA().fit(X=data)
    gc_k=[]
```

```
    gc_k1=[]
    if Nmax==0:
        Nmax=shape(zmdata)[0]
    for i in range(shape(zmdata)[1]+1,Nmax+1,num):
    #for i in range(shape(zmdata)[1]+1,shape(zmdata)[0]+1,num):
        gc=GC_optk_test(zmdata[:i],s)
        gc_k.append(gc)
        gc1=GC_optk_test(zmdata[:i],s,b=1)
        gc_k1.append(gc1)
    plot(gc_k, 'x', markersize=5,label='GC plus',color="blue")
    plot(gc_k1, '.', markersize=5,label='GC minus',color="red")

    legend();
    xlabel('Number of instances').set_color('black');
    ylabel('Reduced dimension').set_color('black');
    if vec==1:
        return gc_k,gc_k1
```

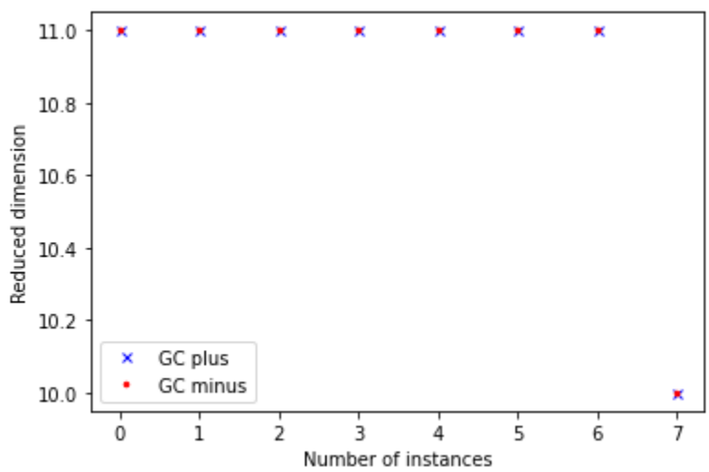
In [77]:
```
PCA_datalen_test(my_data, num=1, s=15, vec=0)
```



In [51]:
```
shape(my_data)
```

Out[51]:
```
(208, 61)
```

In [78]:
```
PCA_datalen_test(my_data, num=1, s=20, vec=0, Nmax=69)
```



List of functions ◇

- changebasis(data, components)
- data_int(data,order)
- data_Tavory(data,k,tau)
- GC_optk(data,s=64)
- GC_s(data)
- GC_table(data,s=64,table=False)
- Minka_optk(data)
- nml_optk(data)

- `PCA_datafeat(data, num=1, minka=1, tavory=1, GC=1, s=64)`
- `PCA_datalen(data, num=1, minka=1, tavory=1, GC=1, s=64)`
- `PCA_test(data, knee=1, rounds=10, minka=1, tavory=1, GC=1, s=64)`
- `precision(data,sign_digits,b=0)`
- `zeromean(data)`

In [34]:
```
1 ?PCA_test
```

References ◇

Data repository https://archive.ics.uci.edu/ml/index.php (https://archive.ics.uci.edu/ml/index.php)

[Minka2000] (https://papers.nips.cc/paper/2000/file/7503cfacd12053d309b6bed5c89de212-Paper.pdf) Minka, Thomas P. "Automatic choice of dimensionality for PCA." Nips. Vol. 13. 2000.

[Satopaa2011] Ville Satopaa, Jeannie R. Albrecht, David E. Irwin, and Barath Raghavan. Finding a "kneedle" in a haystack: Detecting knee points in system behavior. In 31st IEEE International Conference on Distributed Computing Systems Workshops (ICDCS 2011 Workshops), 20-24 June 2011, Minneapolis, Minnesota, USA, pages 166{171, 2011.

[Tavory2019] (https://arxiv.org/abs/1901.00059) Tavory, Ami. Determining Principal Component Cardinality Through the Principle of Minimum Description Length. International Conference on Machine Learning, Optimization, and Data Science. Springer, Cham, 2019.

[Mera&al.2020] (https://dev.arxiv.org/abs/2007.02904?context=math) Mera, Bruno, Mateus, Paulo, and Carvalho, Alexandra M.. "On the minmax regret for statistical manifolds: the role of curvature." arXiv preprint arXiv:2007.02904 (2020).

In [ ]:
```
1
```