Lab session analysis

Homework 1

- **Consider the random matrix. Describe the Behavior and relationship between K2(A) and K∞(A). Why their overall trend is similar? Does the definition of ill-conditioning depend on the norm used? Is there a relationship between the condition number of a matrix and the relative error E(xtrue, x) of the computed solution?**

1. Regarding the first matrix, we observe that the behavior of the condition number is independent from the choice of the norm. Both norms measure in some way the order of magnitude of the matrix (growth with respect to the shape).

2. The ill-conditioning depends on the norm, in fact in the cases observed we noticed that the inf-norm condition number was always greater than the 2-Norm one, so a matrix is more likely ill-conditioned if we use $K_{inf(A)}$. In fact once we know that the norms measure the shape of the matrix, the ill conditioning is strictly related to the accuracy of the solution of the linear system.

3. There's a relation between the relative error and the condition number, in fact it measures how the solution of the linear system is accurate. Actually the condition number is just an upper bound of the relative error, so the relation isn't precise. In this case the matrix is accurate because we generated the numbers, so there's no error on A.

$$\frac{||\delta x||}{||x||} \leq K(A)\frac{||\delta b||}{||b||}$$

- **Consider the Vandermonde matrix. Describe the behavior and relationship between K2(A) and K∞(A). Why their overall trend is similar? Does the definition of ill-conditioning depend on the norm used? Is there a relationship between the condition number of a matrix and the relative error E(xtrue, x) of the computed solution?**

Note: The Vandermonde matrix computation has problems if it takes as inputs python integers, which are of unlimited size, (so when the numpy function computes the exp some of them become negative). On the other hand if we use floats to create the array x the numpy functions use the float arithmetic, which convert to inf the element not representable.

1. Again, the norms measure the order of magnitude of the problem, so the norm chosen for the computation doesn't affect the overall result. (which is the same for both norms)
2. The ill-condition of a matrix is dependent from the norm used as in the first matrix.

3. The relative error and K are strongly related. In case the input data were machine-real number (they had exact representation in our floating point system) the error on the input data $\frac{||\delta b||}{||b||}$ would be exactly zero, so there would be no error to amplify (considering the possible rounding error in the output). Moreover, in this case we must consider also the error on the Vander matrix. In fact the generation of the Vander matrix is a procedure which include exponential computation, so it's prone to errors.

$$\frac{||\delta x||}{||x||} \leq \frac{K(A)}{1 - K(A)\frac{||\delta A||}{||A||}}(\frac{||\delta A||}{||A||} + \frac{||\delta b||}{||b||})$$

- **Consider the Hilbert matrix. Describe the behavior and relationship between K2(A) and K∞(A). Why their overall trend is similar? Does the definition of ill-conditioning depend on the norm used? Is there a relationship between the condition number of a matrix and the relative error E(xtrue, x) of the computed solution?**

Same consideration of Vander matrix. In this case the norm of the matrix is small, but the condition number of Hilbert grows. The inverse tend to be bigger (observations)

Machine epsilon

Smallest representable number by the machine, clearly it depends on the precision of the arithmetic which is been used.

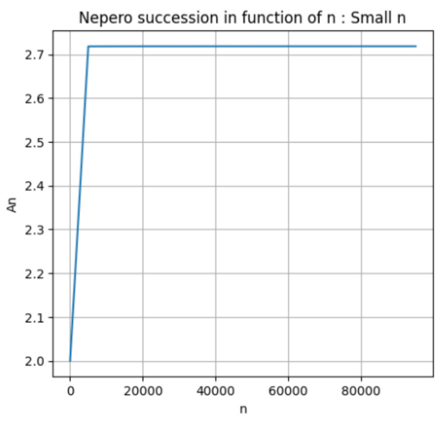
$$\epsilon = min\, fl(1+\epsilon) > 1$$

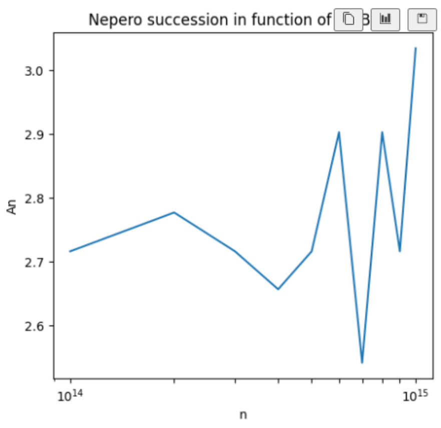
Nepero succession

The computation of a limit such that:

$$\lim_{n\to\infty}\left(1+\frac{1}{n}\right)^{n}$$

Using machine floating point arithmetic could be a problem when the value of n is too big, in fact the division 1/n, which is evaluated before the power, will become zero, due to the representation limit of floats.





When the value of n grows the precision of the floating computation decreases, so the results aren't reliable anymore. If n grows too much the division 1/n will become 0 and the result of the limit will be 1.

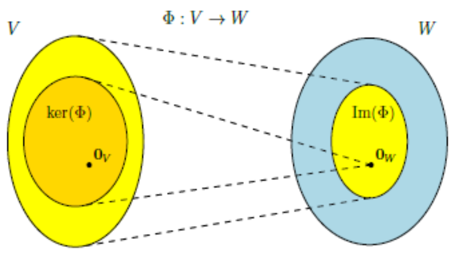
Eigenvalues and rank

The eigenvalues of a linear mapping will tell us how a special set of vectors, the eigenvectors, is transformed by the linear mapping.

$$Ax = \lambda x$$

Kernel and image

- The kernel is the set of vectors v in V that φ maps onto the neutral element 0_w of W.
- The image is the set of vectors w in W that can be "reached" by φ from any vector in V.



Relation between eigenvalues and rank

- Well, if **A** is an n×n matrix, the rank of **A** plus the nullity of **A** is equal to n; that's the **rank-nullity theorem**. The nullity is the dimension of the kernel of the matrix, which is all vectors v of the form:

$$\boldsymbol{Av} = 0 = 0\boldsymbol{v}$$

- The kernel of **A** is precisely the eigenspace corresponding to eigenvalue 0. So, to sum up, the rank is n minus the dimension of the eigenspace corresponding to 0. If 0 is not an eigenvalue, then the kernel is trivial, and so the matrix has full rank n. The rank depends on no other eigenvalues.
- The rank of the matrix can be computed as the dimension of the matrix minus the geometric multiplicity of the eigenvalue **0**. (less or equal than the algebraic multiplicity)

Homework 2

Clustering algorithms

Clustering algorithms try to group together data point which are similar. Often, we deal with high dimensional data, which is impossible to observe and difficult to understand.

If the points in a projected space forms isolated groups such that *inside* of each of them the points are close, while the distance *between* them is large, we call them **clusters**.

There are algorithms to reduce the dimension of data without losing the important information, to help avoiding the curse of dimensionality.

Principal Component Analysis (PCA)

The starting point is a dataset $X = [x_1, x_2, \dots, x_n] \in R^{NxD}$ where N is the dimensionality of data and D is the number of data points. The result expected can be written as $Z = [z_1, z_2, \dots, z_n] \in R^{kxD}$.

When you want to project a matrix by keeping information, a good idea is to use the Singular Value Decomposition (SVD) of it and, in particular, the Truncated SVD (TSVD).

$$X = U\Sigma V^T$$

Since the singular values represents the *quantity of information* contained in the corresponding singular vectors, we can say that the first columns of our decomposition are the most valuable. So PCA consider as the projector only the first columns of U_k.

$$\boldsymbol{Z = U_k X}$$

Where the columns of U_k are called **feature vectors**, and Z columns are the **principal components** of X. We have actually converted each image into a set of k elements. Those k elements are the most informative of the matrix.

Linear Discriminant Analysis (LDA)

Since PCA is an unsupervised learning technique, the lack of information due to the fact that it is not using the associated label reduce its ability on observing clusters. Consequently it can be defined Linear Discriminant Analysis (LDA), a supervised version of PCA, whose ability on generating clusters is way higher.

Method

1. Compute the global centroid and the centroid for each class, which are related to axis = 1
2. Compute the within-cluster scatter matrix Sw, which represents the correlation of points in each cluster.
3. Compute the between-cluster scatter matrix Sb, which represents the correlation between points in different clusters.

4. Compute a projector such that the Sw is maximized and Sb is minimized. First S_w is factorized in two components (Cholesky factorization), then we can compute W, whose columns are the first k eigenvectors of $L^{-1}S_bL$, finally we can compute the projector as:

$$Q = L^{-T}W$$

5. Compute the projected dataset as $Z = Q^TX$

Questions

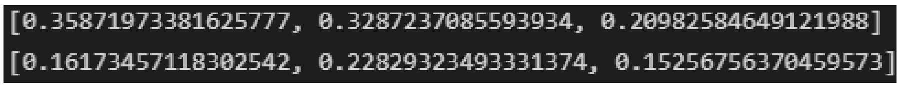
1. **Compare the ability of PCA and LDA in clustering the data. Explain the different behavior of the two methods.**

PCA is an unsupervised learning technique, so it perform the dimensionality reduction using the properties of SVD decomposition, so it tries to maximize the direction of maximum variation of the dataset (we compress the data on the two eigenvectors which contains richer information (Eckert-Young). On the other hand, LDA, which is way more complex, uses a labeled training set to compute a within cluster and a between cluster matrix. LDA is way more accurate in the classification task because it exploits also the supervised information to perform the reduction and consequently the clustering.

Compute the average distance of the centroid of the two methods in the training and test set. There are some differences?

The average distance from the centroid is similar if we move from train to test data, but if we compare the average distances between the two algorithms, we observe a big difference. If we apply a min max scaler on the distances during the computations we notice immediately that the average distance in the case of LDA is clearly smaller.

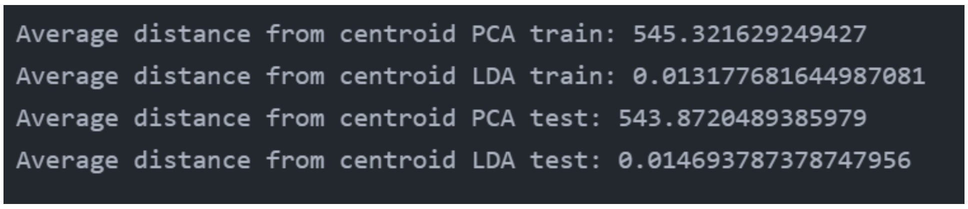
Avg distance PCA
Avg distance LDA


```
[0.35871973381625777, 0.3287237085593934, 0.20982584649121988]
[0.16173457118302542, 0.22829323493331374, 0.15256756370459573]
```

The different order of magnitude between the distances in PCA and LDA is due to the different order of magnitude in the eigenvectors. In fact the columns of Q in PCA are almost one order of magnitude bigger than the eigenvectors of P in LDA.

	1st cluster	2nd cluster	3rd cluster
LDA (train)	542.72	521.01	342.51
PCA (train)	0.012	0.013	0.101

Table 1 Average distances from the cluster centroid on train set.


```
Average distance from centroid PCA train: 545.321629249427
Average distance from centroid LDA train: 0.013177681644987081
Average distance from centroid PCA test: 543.8720489385979
Average distance from centroid LDA test: 0.014693787378747956
```

Define the classifier associated with PCA and LDA. Compare the results.

Once we have the projectors, the classifier is the same for both the algorithms. We project into the k-dimensional space the test set, then we compute the distance of each point w.r.t. the centroids of each class (obtained in the training set) and we assign each data point to the cluster with less distance. Then we compute the accuracy comparing the labels with the predicted values.

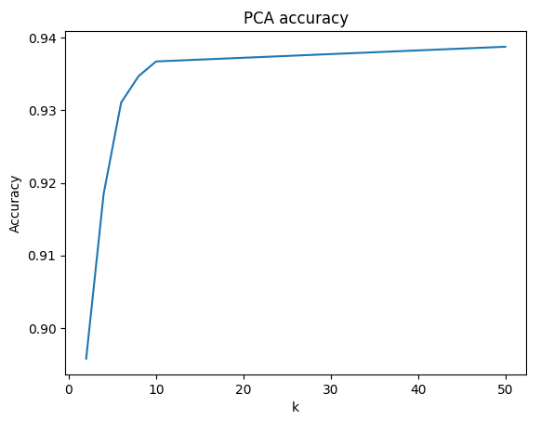
PCA accuracy	LDA accuracy
0.8958	0.9757

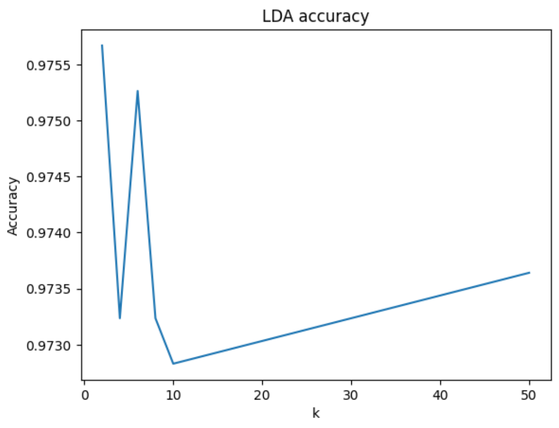
Table 2 Accuracy on MNIST dataset with k = 2.

Define the classifier associated with PCA and LDA. What happens to the accuracy when k grows? Why the accuracy over the test set does not increase monotonically?

When k grows the accuracy on test set for PCA grows very slowly, while the LDA algorithm accuracy on test set decreases a little. The reasons for which the accuracy doesn’t increase monotonically in LDA could be various:

1. Overfitting (LDA and PCA) when k grows too much we don’t capture the semantics of the image.
2. Curse of dimensionality (LDA and PCA): problem due to the dispersions of the pattern of a big volume of data. We are working with high dimensional data so noise and insignificant data are present.

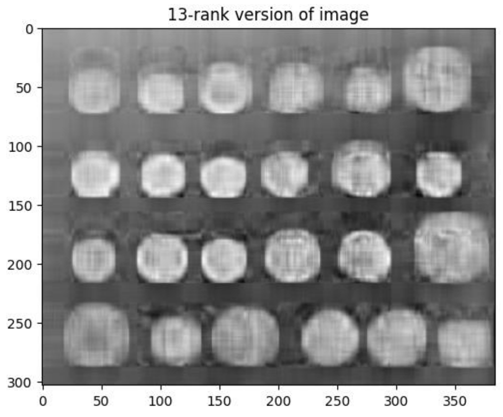


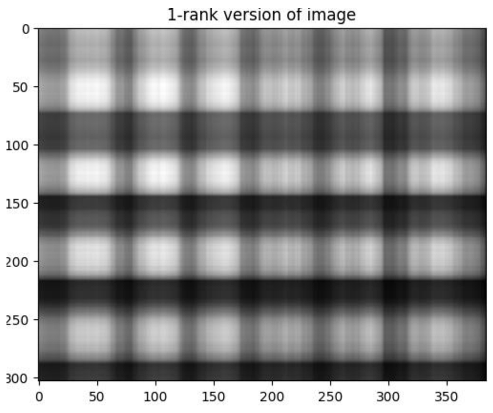


Dyads and SVD

Consider two different images. What do you observe if you compare the k rank approximation of an image X for increasing values of k?

If the value of k grows, it means that we are considering a more complete part of the decomposition, so the quality of the image will increase until $k = rk(A)$.



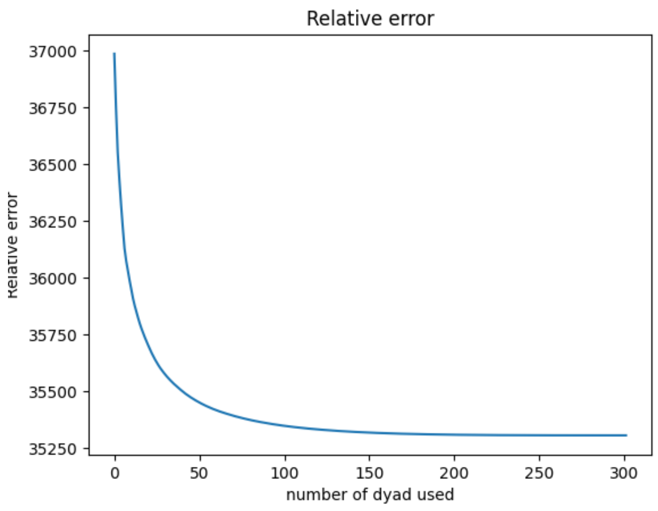


Is there a relationship between the meaningfulness of the dyad of X for a given k and the value of the associated singular value?

Yes, every dyad is the product of two eigenvectors, the ones associated with the greatest singular values are the ones which contain more information about the matrix.

What do you observe if you plot the approximation error $||\mathbf{Xk} - \mathbf{X}||_2$ compared with the plot of σ_k, for increasing values of k?

When we compute the low-rank approximation of a matrix, if we use the ordered SVD, with decreasing singular values, the error of representation is minimized (Eckart-Young theorem).



Homework 3

Gradient descent and SGD

- **Comparison between GD with and without backtracking (for different α > 0). What is the behavior for different functions? Explain.**

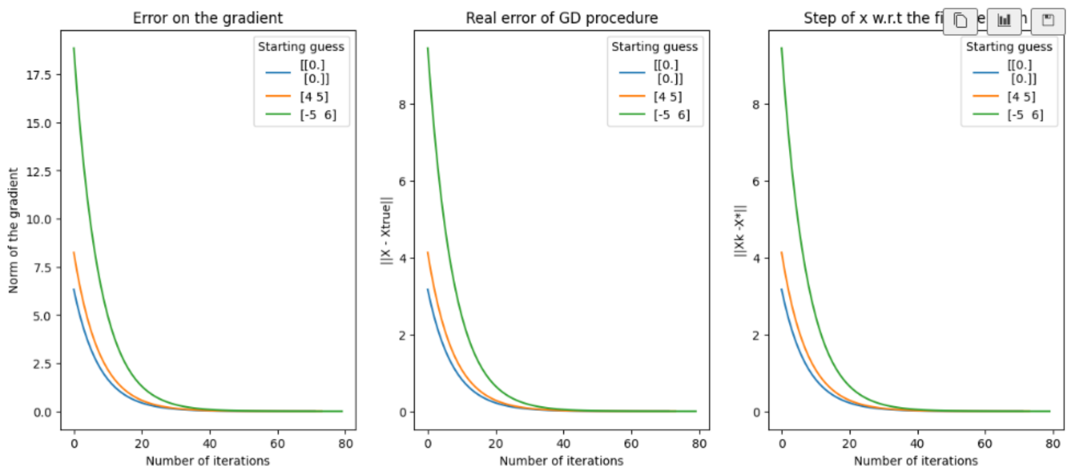
  The backtracking procedure, which dynamically compute the value of α at each step ensure the convergence of the gradient descent algorithm to a stationary point. On the other hand if we use a value of α fixed we risk that our descent won't converge because the steps are too short (α too small). If α is too great the procedure will diverge.

- **By looking the plots of $\left\|\nabla f\left(x^k\right)\right\|_2$, of the error $\|X_{true} - X\_k\|$ and of $\|x_k - x^*\|$ and compare the convergence speed for different functions and for different values of α > 0, constant and chosen with backtracking procedure.**

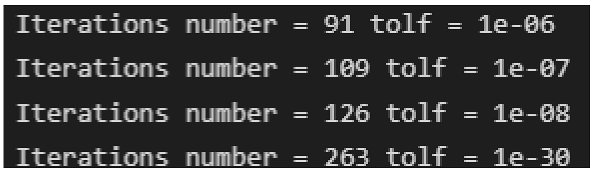
  In general, the convergence speed is faster when we use the backtracking procedure. With a limit of 100 iterations, if the value of alpha is too small the procedure doesn't converge (valid for all functions). If the values of alpha are chosen properly, (with some experiments) the convergence speed became similar to the Armijo one, or, in rare cases, even better (gradient descent of a LSP with regularization term).

- **Consider the function 1. By looking the plots of $\left\|\nabla f\left(x^k\right)\right\|_2$, of the error $\left\|\nabla f\left(x^k\right)\right\|_2$ and of $\|x_k - x^*\|$, discuss the convergence by changing the starting iterate, the tolerances and the step size.**
  Optional:* Observe the behavior of the contour plot for the given examples.

  - If we change the starting guess the initial error change, as it's showed on the graphs, if the starting guess is far away from the minimum the procedure will be significantly slower. The starting guess [0, 0] seems to be the most precise for the function f1.



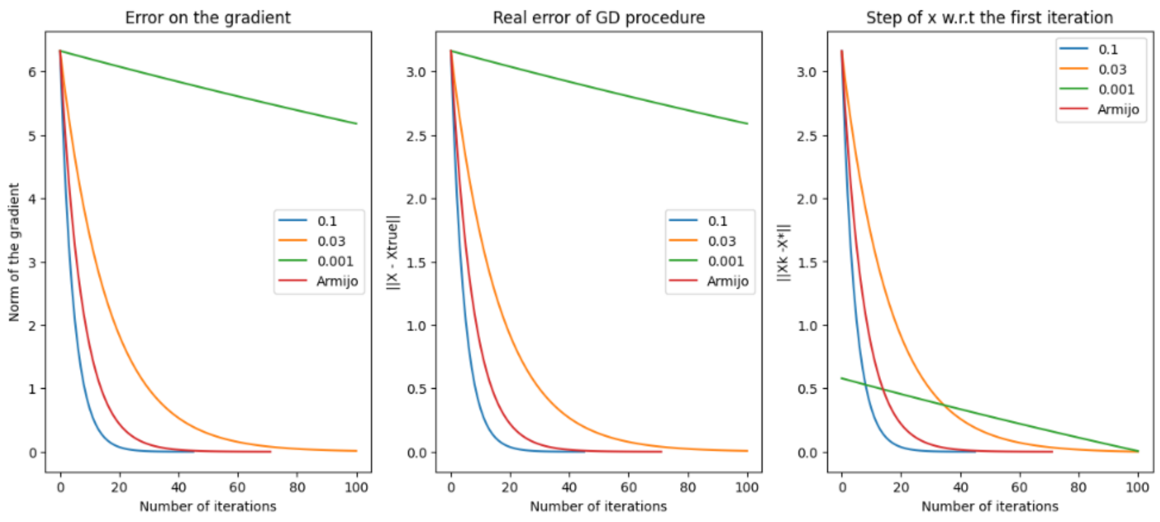
- When we change the tolerance, we are acting on the conditions that we want to apply to stop the gradient descent procedure, if we choose a low level of tolerance the algorithm will proceed more. In fact the tolerance value avoid to proceed if the algorithm got stuck in a **flat region** or if the gradient **vanishes.**



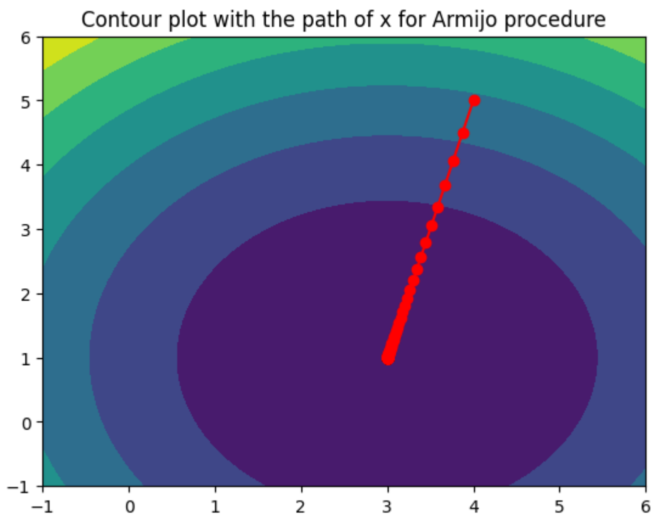
```
Iterations number = 91 tolf = 1e-06
Iterations number = 109 tolf = 1e-07
Iterations number = 126 tolf = 1e-08
Iterations number = 263 tolf = 1e-30
```

Figure 1When tolerance grows the number of iteration before the algorithms stops grows

- If we change the step size the procedure will be generally faster, but non for every value of α. In fact if the step-size big enough the gradient descent results will certainly diverge. If instead α is too small, the total number of iteration will be reached before the GD algorithm finds the minimum.



Finally, we can plot the path from the starting guess to the minimum on the contour plot of the function

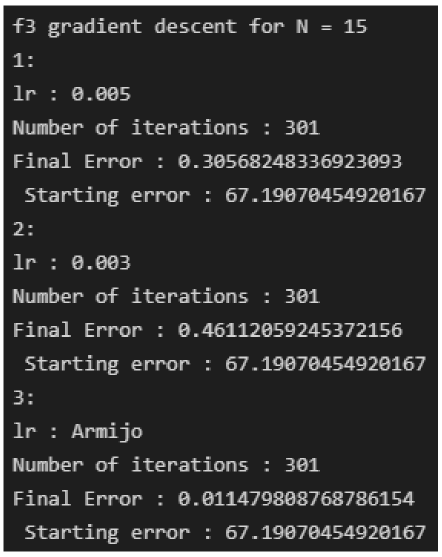


- **Consider the function 2. By looking the plots of $\|\nabla f(x^k)\|_2$, of the error $\|\nabla f(x^k)\|_2$ and of $\|x_k - x^*\|$ |, discuss the convergence by changing the starting iterate, the tolerances and the step size.**
  Optional:* Observe the behavior of the contour plot for the given examples.

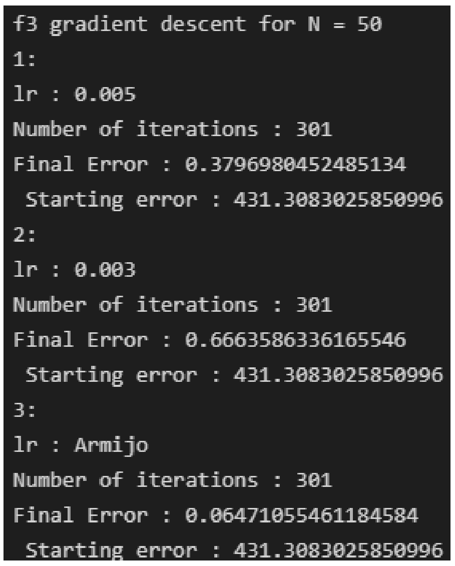
  The considerations are the same of f1.

- **Consider the function 3. By looking the plots of $\|\nabla f(x^k)\|_2$, of the error $\|\nabla f(x^k)\|_2$ and of $\|x_k - x^*\|$ |, discuss the convergence by changing the starting iterate, the tolerances and the step size.**

  If the value of N grows, we can observe that the value of the step-size make the procedure diverge even with low values (E.g. if $N = 50, \alpha = 0.01$ the gradient explodes). The final error obtained grows with N and Armijo procedure became always the most reliable for convergence.

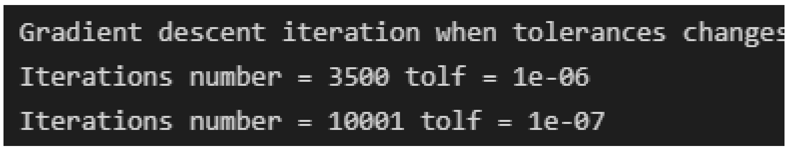


```
f3 gradient descent for N = 15
1:
lr : 0.005
Number of iterations : 301
Final Error : 0.30568248336923093
 Starting error : 67.19070454920167
2:
lr : 0.003
Number of iterations : 301
Final Error : 0.46112059245372156
 Starting error : 67.19070454920167
3:
lr : Armijo
Number of iterations : 301
Final Error : 0.011479808768786154
 Starting error : 67.19070454920167
```



```
f3 gradient descent for N = 50
1:
lr : 0.005
Number of iterations : 301
Final Error : 0.3796980452485134
 Starting error : 431.3083025850996
2:
lr : 0.003
Number of iterations : 301
Final Error : 0.6663586336165546
 Starting error : 431.3083025850996
3:
lr : Armijo
Number of iterations : 301
Final Error : 0.06471055461184584
 Starting error : 431.3083025850996
```

In fact the convergence speed can be measured changing the tolerance. As we can see, when the value is 10^{-6} the convergence requires 3500 iterations. If we increase more the GD doesn't converge neither after 10000 iter.



```
Gradient descent iteration when tolerances change
Iterations number = 3500 tolf = 1e-06
Iterations number = 10001 tolf = 1e-07
```

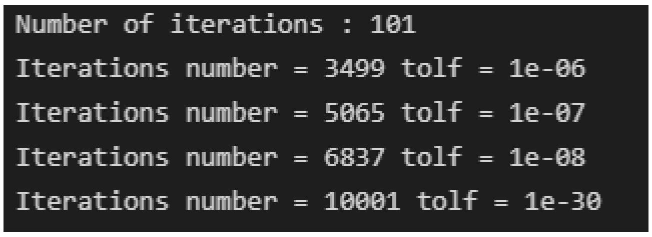
When we find proper values of α the procedure is always faster when α is big.

- **Consider the function 4. By looking the plots of $\left\|\nabla f\left(x^k\right)\right\|_2$, of the error $\left\|\nabla f\left(x^k\right)\right\|_2$ and of $\|x_k - x^*\|$, discuss the convergence by changing the starting iterate, the tolerances and the step size.**

  With the introduction of the regularization term, the values of α needed to make the procedure converge are higher. Moreover, some choices of α are far faster than the Armijo procedure.

  The convergence speed become significantly higher than f3, even with extremely low tolerance values.

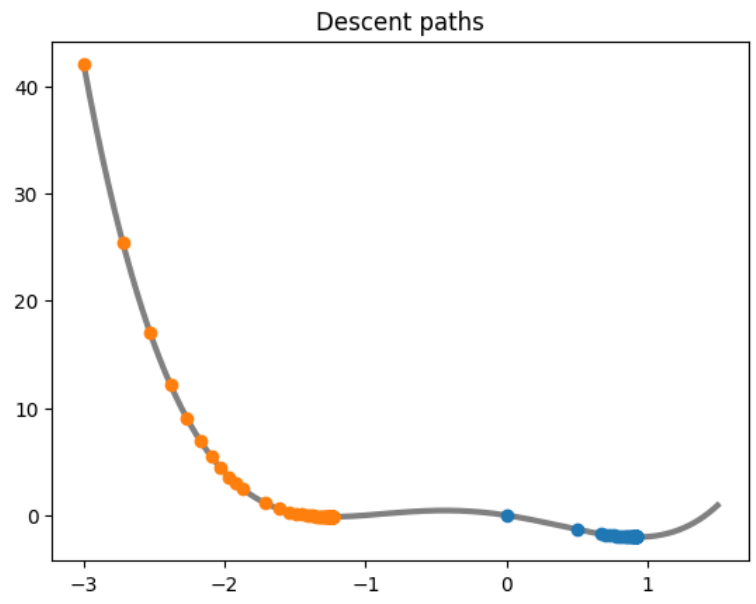
  When N grows we need to be careful selecting the values of the step-size because the system could more likely diverge. Clearly, the convergence speed slow down when the value of N grows.



```
Number of iterations : 101
Iterations number = 3499 tolf = 1e-06
Iterations number = 5065 tolf = 1e-07
Iterations number = 6837 tolf = 1e-08
Iterations number = 10001 tolf = 1e-30
```

- **Consider the function 5. By looking the plots of $\left\|\nabla f\left(x^k\right)\right\|_2$, of the error $\left\|\nabla f\left(x^k\right)\right\|_2$ and of $\|x_k - x^*\|$ |, discuss the convergence by changing the starting iterate, the tolerances and the step size.**

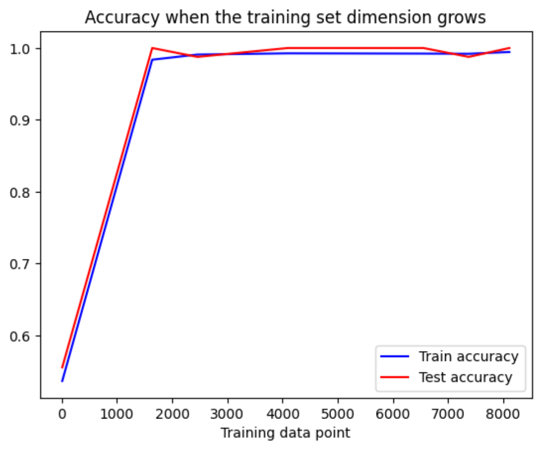
  The function $f5: R \to R$ represent a common problem in gradient descent method, the convergence into **local optima.** In fact, the starting iterate influences the minimum which will be found by the algorithm.



The tolerance variation doesn't change w.r.t f1 or f2. Convergence is quite fast even if the tolerances are low (E.g. $7400\ Iter\ for\ tolf = tolx = 10^{-13}$).

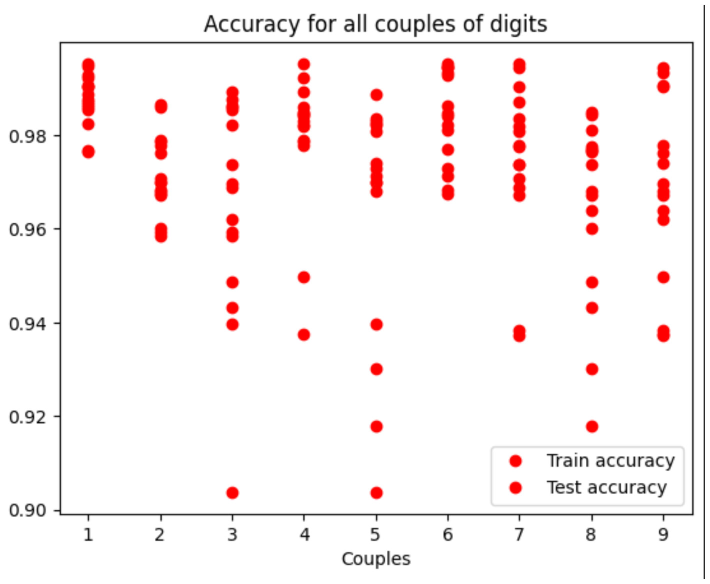
Stochastic Gradient Descent (SGD)

1. **Discuss the behavior of the logistic regression classifier varying the training set dimension (Ntrain).**



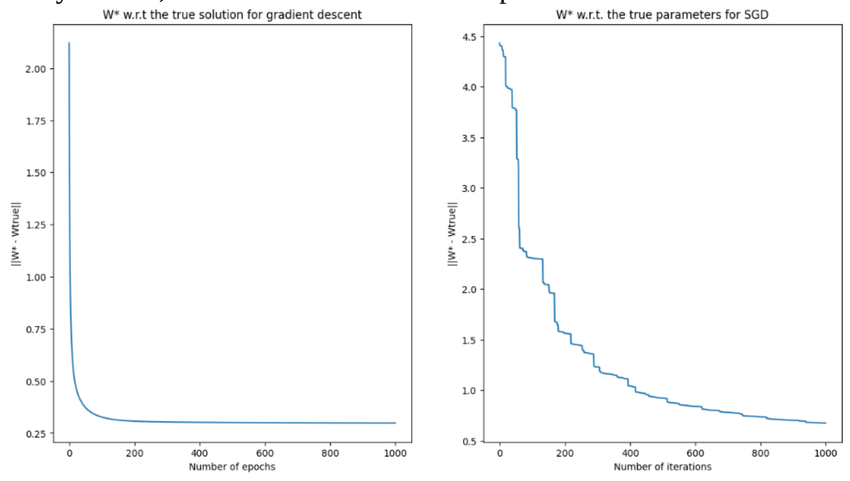
The table shows the variation of accuracy in both training and test set we the train-set grows in dimension, the logistic regressor is been trained for 10 epochs with batch size 1. The growth of the training set imply a significant improvement of the performances.

2. **Discuss the behavior of the logistic regression classifier varying the two considered digits.**
With a proper selection of the hyperparameters, the results aren't affected by the digits selected.



3. **What are the differences at convergence of the parameters w ∗ when computed by GD and SGD, in particular the error of w ∗ against the true solution.**

The SGD isn't an exact procedure because it computes and propagates the gradient only on specific directions randomly chosen, while the GD uses all the inputs at each iteration.



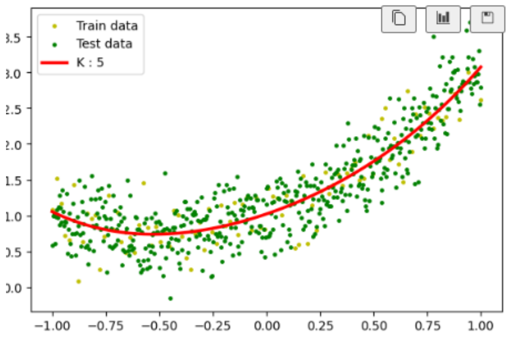
This have an important effect on the modification of the parameters all over the procedure. In fact when we apply GD the weights will move always in the direction of the minimum, at least if we consider simple function with a single minimum and no saddle points. SDG, on the contrary, computes and propagates the gradient over the weights just in some directions, so there's no assurance of the convergence into a minimum. The final error on weights is greater in SGD.

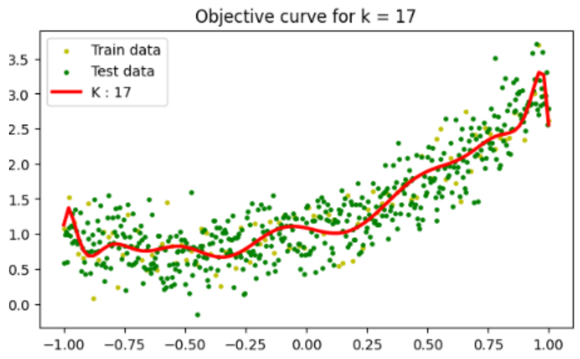
4. **Compare the accuracy of the Logistic Regression Classifier against PCA and LDA for the same considered digits (two digits only).**

If the logistic regressor is well parametrized, the accuracy for two digits is similar to the one of LDA, while PCA remain lower than the others. In fact, we can conclude that supervised algorithm are generally more precise.

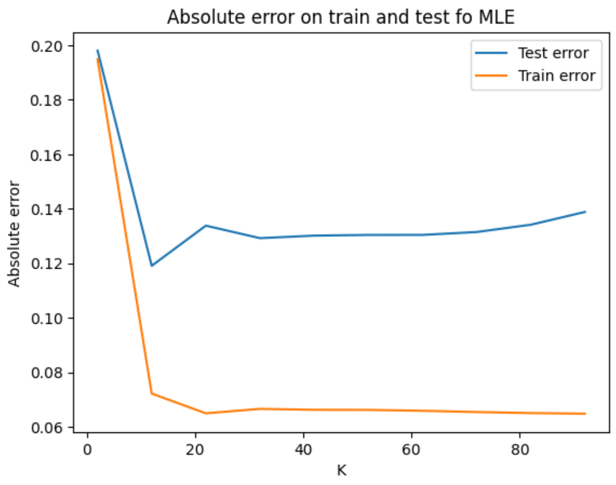
Homework 4:

1. **What is the behavior of the trained regressor model fθ(x), where θ is the MLE solution under Gaussian assumptions, for increasing values of K? Explain the plot where the training and the test error are compared for increasing values of K.**
   When k grows, the MLE implementation overfits, this is due to the fact that the algorithm tries to maximize the likelihood of the output y given the input data x, so there isn't a regularization term.



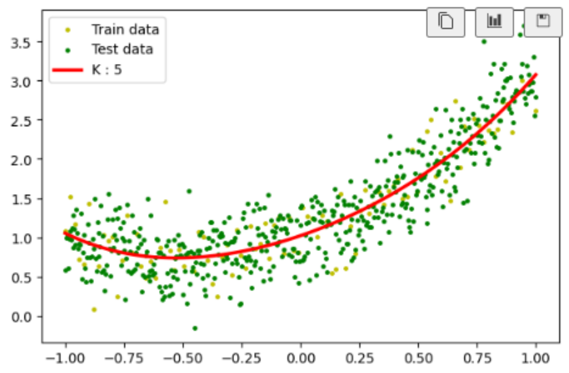


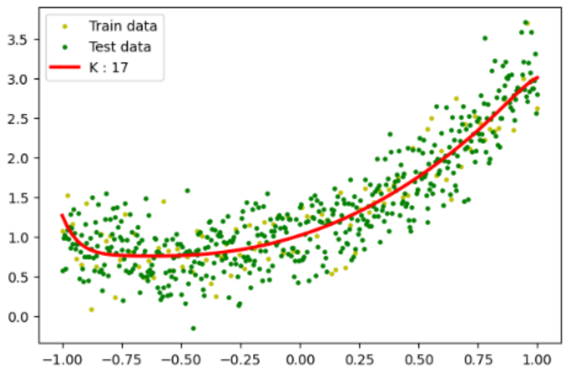
   As expected, the overfitting behavior can be seen if we plot the accuracy on train and test set when k grows.



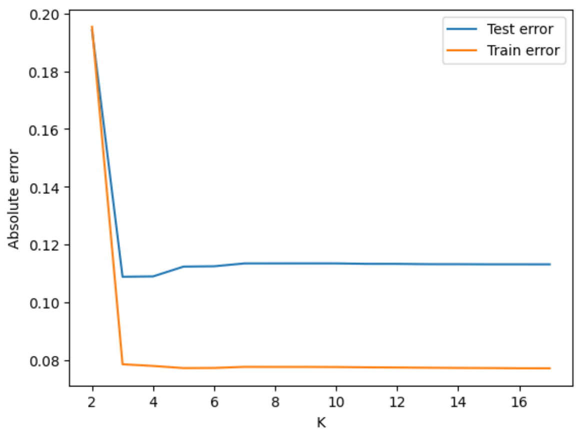
2. **What is the behavior of the trained regressor model fθ(x), where θ is the MAP solution under Gaussian assumptions, for increasing values of K and fixed λ? Explain the plot where the training and the test error are compared for increasing values of K.**

MAP algorithm, with gaussian assumption is rubust to overfitting, in fact when k grows, the curve doesn't change in relation with the data.





The map procedure is stronger to avoid overfitting, Test error don't grow.



3. **What is the behavior of the trained regressor model fθ(x), where θ is the MAP solution under Gaussian assumptions, for fixed value K lower and/or greater than the true K and different λ?**

Lambda measures in some way how much we "trust" our model, when k is lower than ktrue the behaviors is worse when lambda grows. On the other hand, when k grows the system overfits more likely if lambda is small (extreme case lambda = 0)