⊖ Everything running smoothly!

**TP1 : First order methods on regression models**[**¶**](https://render.githubusercontent.com/view/ipynb?commit=50de7cb8624552ca8abc8743ccc86d6316c0a51b&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f54644f70742f5444312f353064653763623836323435353263613861626338373433636363383664363331366330613531622f7470315f6d61737365745f63616d696c6c655f616e645f6d757a656c6c65635f626f7269732e6970796e62&nwo=TdOpt%2FTD1&path=tp1_masset_camille_and_muzellec_boris.ipynb&repository_id=69846964#TP1-:-First-order-methods-on-regression-models)

**Authors: S. Gaiffas, A. Gramfort[¶](https://render.githubusercontent.com/view/ipynb?commit=50de7cb8624552ca8abc8743ccc86d6316c0a51b&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f54644f70742f5444312f353064653763623836323435353263613861626338373433636363383664363331366330613531622f7470315f6d61737365745f63616d696c6c655f616e645f6d757a656c6c65635f626f7269732e6970796e62&nwo=TdOpt%2FTD1&path=tp1_masset_camille_and_muzellec_boris.ipynb&repository_id=69846964" \l "Authors:-S.-Gaiffas,-A.-Gramfort)**

**Aim**[**¶**](https://render.githubusercontent.com/view/ipynb?commit=50de7cb8624552ca8abc8743ccc86d6316c0a51b&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f54644f70742f5444312f353064653763623836323435353263613861626338373433636363383664363331366330613531622f7470315f6d61737365745f63616d696c6c655f616e645f6d757a656c6c65635f626f7269732e6970796e62&nwo=TdOpt%2FTD1&path=tp1_masset_camille_and_muzellec_boris.ipynb&repository_id=69846964#Aim)

The aim of this material is to code

* proximal gradient descent (ISTA)
* accelerated gradient descent (FISTA)

for

* linear regression
* logistic regression

models.

The proximal operators we will use are the

* ridge penalization
* L1 penalization

**VERY IMPORTANT**[**¶**](https://render.githubusercontent.com/view/ipynb?commit=50de7cb8624552ca8abc8743ccc86d6316c0a51b&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f54644f70742f5444312f353064653763623836323435353263613861626338373433636363383664363331366330613531622f7470315f6d61737365745f63616d696c6c655f616e645f6d757a656c6c65635f626f7269732e6970796e62&nwo=TdOpt%2FTD1&path=tp1_masset_camille_and_muzellec_boris.ipynb&repository_id=69846964#VERY-IMPORTANT)

* This work **must be done by pairs of students**.
* **Each** student must send their work **before the 9th of october at 23:59**, using the **moodle platform**.
* This means that **each student in the pair sends the same file**
* On the moodle, in the "Optimization for Data Science" course, you have a "devoir" section called **Rendu TP du 3 octobre 2016**. This is where you submit your jupyter notebook file.
* The **name of the file must be** constructed as in the next cell

**Gentle reminder: no evaluation if you don't respect this EXACTLY**[**¶**](https://render.githubusercontent.com/view/ipynb?commit=50de7cb8624552ca8abc8743ccc86d6316c0a51b&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f54644f70742f5444312f353064653763623836323435353263613861626338373433636363383664363331366330613531622f7470315f6d61737365745f63616d696c6c655f616e645f6d757a656c6c65635f626f7269732e6970796e62&nwo=TdOpt%2FTD1&path=tp1_masset_camille_and_muzellec_boris.ipynb&repository_id=69846964#Gentle-reminder:-no-evaluation-if-you-don't-respect-this-EXACTLY)

**How to construct the name of your file**[**¶**](https://render.githubusercontent.com/view/ipynb?commit=50de7cb8624552ca8abc8743ccc86d6316c0a51b&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f54644f70742f5444312f353064653763623836323435353263613861626338373433636363383664363331366330613531622f7470315f6d61737365745f63616d696c6c655f616e645f6d757a656c6c65635f626f7269732e6970796e62&nwo=TdOpt%2FTD1&path=tp1_masset_camille_and_muzellec_boris.ipynb&repository_id=69846964#How-to-construct-the-name-of-your-file)

In [ ]:

# Change here using YOUR first and last names

fn1 = "camille"

ln1 = "masset"

fn2 = "boris"

ln2 = "muzellec"

filename = "\_".join(map(lambda s: s.strip().lower(),

["tp1", ln1, fn1, "and", ln2, fn2])) + ".ipynb"

print(filename)

In [ ]:

## to embed figures in the notebook

%matplotlib inline

**Part 0 : Introduction**[**¶**](https://render.githubusercontent.com/view/ipynb?commit=50de7cb8624552ca8abc8743ccc86d6316c0a51b&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f54644f70742f5444312f353064653763623836323435353263613861626338373433636363383664363331366330613531622f7470315f6d61737365745f63616d696c6c655f616e645f6d757a656c6c65635f626f7269732e6970796e62&nwo=TdOpt%2FTD1&path=tp1_masset_camille_and_muzellec_boris.ipynb&repository_id=69846964#Part-0-:-Introduction)

We'll start by generating sparse vectors and simulating data

**Getting sparse coefficients**[**¶**](https://render.githubusercontent.com/view/ipynb?commit=50de7cb8624552ca8abc8743ccc86d6316c0a51b&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f54644f70742f5444312f353064653763623836323435353263613861626338373433636363383664363331366330613531622f7470315f6d61737365745f63616d696c6c655f616e645f6d757a656c6c65635f626f7269732e6970796e62&nwo=TdOpt%2FTD1&path=tp1_masset_camille_and_muzellec_boris.ipynb&repository_id=69846964#Getting-sparse-coefficients)

In [ ]:

import numpy as np

import matplotlib.pyplot as plt

np.set\_printoptions(precision=2) # to have simpler print outputs with numpy

In [ ]:

n\_features = 50

n\_samples = 1000

idx = np.arange(n\_features)

coefs = (-1) \*\* (idx - 1) \* np.exp(-idx / 10.)

coefs[20:] = 0.

plt.stem(coefs)

plt.title("Parameters / Coefficients")

**Functions for the simulation of the models**[**¶**](https://render.githubusercontent.com/view/ipynb?commit=50de7cb8624552ca8abc8743ccc86d6316c0a51b&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f54644f70742f5444312f353064653763623836323435353263613861626338373433636363383664363331366330613531622f7470315f6d61737365745f63616d696c6c655f616e645f6d757a656c6c65635f626f7269732e6970796e62&nwo=TdOpt%2FTD1&path=tp1_masset_camille_and_muzellec_boris.ipynb&repository_id=69846964#Functions-for-the-simulation-of-the-models)

In [ ]:

from numpy.random import multivariate\_normal

from scipy.linalg.special\_matrices import toeplitz

from numpy.random import randn

def simu\_linreg(coefs, n\_samples=1000, corr=0.5):

"""Simulation of a linear regression model

Parameters

----------

coefs : `numpy.array`, shape=(n\_features,)

Coefficients of the model

n\_samples : `int`, default=1000

Number of samples to simulate

corr : `float`, default=0.5

Correlation of the features

Returns

-------

A : `numpy.ndarray`, shape=(n\_samples, n\_features)

Simulated features matrix. It samples of a centered Gaussian

vector with covariance given by the Toeplitz matrix

b : `numpy.array`, shape=(n\_samples,)

Simulated labels

"""

# Construction of a covariance matrix

cov = toeplitz(corr \*\* np.arange(0, n\_features))

# Simulation of features

A = multivariate\_normal(np.zeros(n\_features), cov, size=n\_samples)

# Simulation of the labels

b = A.dot(coefs) + randn(n\_samples)

return A, b

def sigmoid(t):

"""Sigmoid function (overflow-proof)"""

idx = t > 0

out = np.empty(t.size)

out[idx] = 1. / (1 + np.exp(-t[idx]))

exp\_t = np.exp(t[~idx])

out[~idx] = exp\_t / (1. + exp\_t)

return out

def simu\_logreg(coefs, n\_samples=1000, corr=0.5):

"""Simulation of a logistic regression model

Parameters

----------

coefs : `numpy.array`, shape=(n\_features,)

Coefficients of the model

n\_samples : `int`, default=1000

Number of samples to simulate

corr : `float`, default=0.5

Correlation of the features

Returns

-------

A : `numpy.ndarray`, shape=(n\_samples, n\_features)

Simulated features matrix. It samples of a centered Gaussian

vector with covariance given by the Toeplitz matrix

b : `numpy.array`, shape=(n\_samples,)

Simulated labels

"""

cov = toeplitz(corr \*\* np.arange(0, n\_features))

A = multivariate\_normal(np.zeros(n\_features), cov, size=n\_samples)

p = sigmoid(A.dot(coefs))

b = np.random.binomial(1, p, size=n\_samples)

b[:] = 2 \* b - 1

return A, b

In [ ]:

# Samples data to test functions

coefs = randn(n\_features)

A, b = simu\_linreg(coefs, n\_samples=1000, corr=0.5)

**Part 1 : Proximal operators**[**¶**](https://render.githubusercontent.com/view/ipynb?commit=50de7cb8624552ca8abc8743ccc86d6316c0a51b&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f54644f70742f5444312f353064653763623836323435353263613861626338373433636363383664363331366330613531622f7470315f6d61737365745f63616d696c6c655f616e645f6d757a656c6c65635f626f7269732e6970796e62&nwo=TdOpt%2FTD1&path=tp1_masset_camille_and_muzellec_boris.ipynb&repository_id=69846964#Part-1-:-Proximal-operators)

We remind that the proximal operator of a fonction is given by:

where is a non-negative number. We have in mind to use the following cases

* Ridge penalization, where
* Lasso penalization, where

where is a regularization parameter.

**Questions**[**¶**](https://render.githubusercontent.com/view/ipynb?commit=50de7cb8624552ca8abc8743ccc86d6316c0a51b&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f54644f70742f5444312f353064653763623836323435353263613861626338373433636363383664363331366330613531622f7470315f6d61737365745f63616d696c6c655f616e645f6d757a656c6c65635f626f7269732e6970796e62&nwo=TdOpt%2FTD1&path=tp1_masset_camille_and_muzellec_boris.ipynb&repository_id=69846964#Questions)

* Code a function that computes in both cases and for ridge and lasso penalization (use the slides of the first course to get the formulas), using the prototypes given below
* Visualize the functions applied element wise by the proximity operators of the Ridge and Lasso

In [ ]:

def prox\_lasso(x, s, t=1.):

"""Proximal operator for the Lasso at x with strength t"""

return np.multiply(np.sign(x),np.maximum(np.absolute(x) - s\*t,0))

def lasso(x, s):

"""Value of the Lasso penalization at x with strength t"""

return s\*np.linalg.norm(x,1)

def prox\_ridge(x, s, t=1.):

"""Proximal operator for the ridge at x with strength t"""

return 1 / (1+s\*t) \* x

def ridge(x, s):

"""Value of the ridge penalization at x with strength t"""

return s / 2 \* np.linalg.norm(x,2) \*\* 2

**Visualization**[**¶**](https://render.githubusercontent.com/view/ipynb?commit=50de7cb8624552ca8abc8743ccc86d6316c0a51b&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f54644f70742f5444312f353064653763623836323435353263613861626338373433636363383664363331366330613531622f7470315f6d61737365745f63616d696c6c655f616e645f6d757a656c6c65635f626f7269732e6970796e62&nwo=TdOpt%2FTD1&path=tp1_masset_camille_and_muzellec_boris.ipynb&repository_id=69846964#Visualization)

We are now going to visualize the effect of the proximity operators on coefficients.

In [ ]:

x = randn(50)

l\_l1 = 1.

l\_l2 = 0.5

plt.figure(figsize=(15.0, 4.0))

plt.subplot(1, 3, 1)

plt.stem(x)

plt.title("Original parameter", fontsize=16)

plt.ylim([-2, 2])

plt.subplot(1, 3, 2)

plt.stem(prox\_lasso(x, s=l\_l1))

plt.title("Proximal Lasso", fontsize=16)

plt.ylim([-2, 2])

plt.subplot(1, 3, 3)

plt.stem(prox\_ridge(x, s=l\_l2))

plt.title("Proximal Ridge", fontsize=16)

plt.ylim([-2, 2])

**Question**[**¶**](https://render.githubusercontent.com/view/ipynb?commit=50de7cb8624552ca8abc8743ccc86d6316c0a51b&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f54644f70742f5444312f353064653763623836323435353263613861626338373433636363383664363331366330613531622f7470315f6d61737365745f63616d696c6c655f616e645f6d757a656c6c65635f626f7269732e6970796e62&nwo=TdOpt%2FTD1&path=tp1_masset_camille_and_muzellec_boris.ipynb&repository_id=69846964#Question)

* Comment what you observe (1 or 2 sentences).

Proximal Lasso induces sparsity: all coefficients with absolute value under vanish, and the others have their absolute value reduced by .

Proximal Ridge has a less violent action: it scales down coefficients (by the same factor ), but no coefficient vanishes.

**Part 2: Gradients**[**¶**](https://render.githubusercontent.com/view/ipynb?commit=50de7cb8624552ca8abc8743ccc86d6316c0a51b&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f54644f70742f5444312f353064653763623836323435353263613861626338373433636363383664363331366330613531622f7470315f6d61737365745f63616d696c6c655f616e645f6d757a656c6c65635f626f7269732e6970796e62&nwo=TdOpt%2FTD1&path=tp1_masset_camille_and_muzellec_boris.ipynb&repository_id=69846964#Part-2:-Gradients)

The problems we want to minimize take the form: where is -smooth and is prox-capable.

We will consider below the following cases

**Linear regression**, where where is the sample size, is the vector of labels and is the matrix of features.

**Logistic regression**, where where is the sample size, and where labels for all .

We need to be able to compute and its gradient

**Questions**:

* Compute on paper the gradient of for both cases (linear and logistic regression)
* Code a function that computes and its gradient in both cases, using the prototypes below.
* Check that these functions are correct by numerically checking the gradient, using the function checkgrad from scipy.optimize. Remark: use the functions simu\_linreg and simu\_logreg to simulate data according to the right model

In [ ]:

def loss\_linreg(x):

"""Least-squares loss"""

return np.linalg.norm(b - A.dot(x))\*\*2 / (2\*A.shape[0])

def grad\_linreg(x):

"""Leas-squares gradient"""

return A.T.dot(A.dot(x) - b) / A.shape[0]

def loss\_logreg(x):

"""Logistic loss"""

return np.log(1 + np.exp(-np.multiply(b, A.dot(x)))).sum() / A.shape[0]

def grad\_logreg(x):

"""Logistic gradient"""

return -np.multiply(np.multiply(b.reshape((-1, 1)), A), sigmoid(-np.multiply(b, A.dot(x))).reshape((-1, 1))).sum(axis=0) / A.shape[0]

In [ ]:

from scipy.optimize import check\_grad

print(check\_grad(loss\_logreg,grad\_logreg,randn(50)))

**Part 3: Solvers**[**¶**](https://render.githubusercontent.com/view/ipynb?commit=50de7cb8624552ca8abc8743ccc86d6316c0a51b&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f54644f70742f5444312f353064653763623836323435353263613861626338373433636363383664363331366330613531622f7470315f6d61737365745f63616d696c6c655f616e645f6d757a656c6c65635f626f7269732e6970796e62&nwo=TdOpt%2FTD1&path=tp1_masset_camille_and_muzellec_boris.ipynb&repository_id=69846964#Part-3:-Solvers)

We know have a function to compute , and and .

We want now to code the Ista and Fista solvers to minimize

**Questions**:

* Implement functions that compute the Lipschitz constants for linear and logistic regression losses. Note that the operator norm of a matrix can be computed using the function numpy.linalg.norm (read the documentation of the function)
* Finish the functions ista and fista below that implements the ISTA (Proximal Gradient Descent) and FISTA (Accelerated Proximal Gradient Descent) algorithms

In [ ]:

def lip\_linreg(A):

"""Lipschitz constant for linear squares loss"""

return np.linalg.norm(A.T.dot(A),2)/A.shape[0]

def lip\_logreg(A):

"""Lipschitz constant for logistic loss"""

return lip\_linreg(A)/4

def ista(x0, f, grad\_f, g, prox\_g, step, s=0., n\_iter=50,

x\_true=coefs, verbose=True):

"""Proximal gradient descent algorithm

"""

from scipy.optimize import check\_grad

x = x0.copy()

x\_new = x0.copy()

n\_samples, n\_features = A.shape

# estimation error history

errors = []

# objective history

objectives = []

# Current estimation error

err = np.linalg.norm(x - x\_true) / np.linalg.norm(x\_true)

errors.append(err)

# Current objective

obj = f(x) + g(x, s)

objectives.append(obj)

if verbose:

print ("Lauching ISTA solver...")

print (' | '.join([name.center(8) for name in ["it", "obj", "err"]]))

for k in range(n\_iter + 1):

x = prox\_g(x - step\*grad\_f(x))

obj = f(x) + g(x, s)

err = np.linalg.norm(x - x\_true) / np.linalg.norm(x\_true)

errors.append(err)

objectives.append(obj)

if k % 10 == 0 and verbose:

print (' | '.join([("%d" % k).rjust(8),

("%.2e" % obj).rjust(8),

("%.2e" % err).rjust(8)]))

return x, objectives, errors

In [ ]:

def fista(x0, f, grad\_f, g, prox\_g, step, s=0., n\_iter=50,

x\_true=coefs, verbose=True):

"""Accelerated Proximal gradient descent algorithm

"""

x = x0.copy()

x\_new = x0.copy()

# An extra variable is required for FISTA

z = x0.copy()

n\_samples, n\_features = A.shape

# estimation error history

errors = []

# objective history

objectives = []

# Current estimation error

err = np.linalg.norm(x - x\_true) / np.linalg.norm(x\_true)

errors.append(err)

# Current objective

obj = f(x) + g(x, s)

objectives.append(obj)

t = 1.

t\_new = 1.

if verbose:

print ("Lauching FISTA solver...")

print (' | '.join([name.center(8) for name in ["it", "obj", "err"]]))

for k in range(n\_iter + 1):

x\_new = prox\_g(z-step\*grad\_f(z))

t\_new = (1 + np.sqrt(1+4\*t\*\*2))/2

z\_new = x\_new + (t-1)/t\_new \* (x\_new-x)

x, z, t = x\_new, z\_new, t\_new

obj = f(x) + g(x, s)

err = np.linalg.norm(x - x\_true) / np.linalg.norm(x\_true)

errors.append(err)

objectives.append(obj)

if k % 10 == 0 and verbose:

print (' | '.join([("%d" % k).rjust(8),

("%.2e" % obj).rjust(8),

("%.2e" % err).rjust(8)]))

return x, np.array(objectives), np.array(errors)

**Algorithms comparison and numerical experiments**[**¶**](https://render.githubusercontent.com/view/ipynb?commit=50de7cb8624552ca8abc8743ccc86d6316c0a51b&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f54644f70742f5444312f353064653763623836323435353263613861626338373433636363383664363331366330613531622f7470315f6d61737365745f63616d696c6c655f616e645f6d757a656c6c65635f626f7269732e6970796e62&nwo=TdOpt%2FTD1&path=tp1_masset_camille_and_muzellec_boris.ipynb&repository_id=69846964#Algorithms-comparison-and-numerical-experiments)

In [ ]:

# Some definitions before launching the algorithms

x0 = np.zeros(n\_features)

n\_iter = 100

s = 1e-2

**Questions**

* Compute a precise minimum and a precise minimizer of the linear regression with ridge penalization problem using the parameters give above. This can be done by using fista with 1000 iterations.
* Compare the convergences of ISTA and FISTA, in terms of distance to the minimum and distance to the minimizer. Do your plots using a logarithmic scale of the y-axis.

In [ ]:

# Linear regression

A, b = simu\_linreg(coefs, n\_samples=1000, corr=0.5)

L = lip\_linreg(A)

x1, obj1, err1 = ista(x0, loss\_linreg, grad\_linreg, ridge, lambda x : prox\_ridge(x,s,1/L), 1/L, s, n\_iter,

x\_true=coefs, verbose=True)

x2, obj2, err2 = fista(x0, loss\_linreg, grad\_linreg, ridge, lambda x : prox\_ridge(x,s,1/L), 1/L, s, n\_iter,

x\_true=coefs, verbose=True)

In [ ]:

fig, ax = plt.subplots()

ax.semilogy(np.arange(len(err1)), err1, label="ISTA error")

ax.semilogy(np.arange(len(err2)), err2, label="FISTA error")

ax.semilogy(np.arange(len(obj1)), obj1, label="ISTA objective")

ax.semilogy(np.arange(len(obj2)), obj2, label="FISTA objective")

legend = ax.legend(loc='upper center')

In [ ]:

# Logistic regression

s = 1e-3

A, b = simu\_logreg(coefs, n\_samples=1000, corr=0.5)

L = lip\_logreg(A)

x1, obj1, err1 = ista(x0, loss\_logreg, grad\_logreg, ridge, lambda x : prox\_ridge(x,s,1/L), 1/L, s, n\_iter,

x\_true=coefs, verbose=True)

x2, obj2, err2 = fista(x0, loss\_logreg, grad\_logreg, ridge, lambda x : prox\_ridge(x,s,1/L), 1/L, s, n\_iter,

x\_true=coefs, verbose=True)

In [ ]:

fig, ax = plt.subplots()

ax.semilogy(np.arange(len(err1)), err1, label="ISTA error")

ax.semilogy(np.arange(len(err2)), err2, label="FISTA error")

ax.semilogy(np.arange(len(obj1)), obj1, label="ISTA objective")

ax.semilogy(np.arange(len(obj2)), obj2, label="FISTA objective")

legend = ax.legend(loc='upper right')

**Questions**

* In linear regression and logistic regression, study the influence of the correlation of the features on the performance of the optimization algorithms. Explain.
* In linear regression and logistic regression, study the influence of the level of ridge penalization on the performance of the optimization algorithms. Explain.
* In linear regression and logistic regression, compare the performance of the optimization algorithms for ridge and lasso penalizations. Explain

In [ ]: