Lasso and Slope estimators

Neste projeto iremos implementar métodos iterativos para achar a solução do seguinte estimador de mínimos quadrados regularizado:

$$ \min_{b\in\mathbb{R}^{p\times1}}f(b)=\frac{1}{2n}\sum_{i=1}^n(y_i-x_i^\top b)^2+\lambda\Vert b\Vert_\sharp, $$

onde $(y_1,x_1), \ldots, (y_n,x_n)$ é uma amostra de labels/features em $\mathbb{R}\times\mathbb{R}^p$. Àcima, $\lambda>0$ é um hyper-parâmetro positivo e

$$ \Vert b\Vert_\sharp := \sum_{j=1}^p\omega[j]b_\sharp[j], $$

é a *norma Slope* do vetor b, onde $\omega\in\mathbb{R}^{p\times1}$ é um vetor de coordenadas positivas não-decrescente e $b_\sharp[1]\ge\ldots,b_\sharp[p]$ simboliza as coordenadas de b postas em ordem não-crescente. Precisamente, iremos considerar duas opcões:

1. $\omega[j]\equiv1$,
2. $\omega[j]:=\sqrt{\log(2p/j)}$.

Soft-Thresholding e `sortedL1Prox()`

Recorde que o passo de iteração do método gradiente proximal é calcular o *operador proximal* da norma $\lambda\Vert\cdot\Vert_\sharp$:

$$ P(v,\lambda\omega)\in\argmin_{b\in\mathbb{R}^{p\times1}}\left\{ \frac{1}{2}\Vert v - b\Vert_2^2 + \lambda\Vert b\Vert_\sharp \right\}. $$

1. Suponha primeiro que $\omega[j]\equiv1$ então $P(v,\lambda\omega)$ tem fórmula explicíta. Dados $\gamma\in\mathbb{R}$, defina

$$ S(\gamma,\lambda):=(\gamma)\cdot\max\{\gamma-\lambda,0\}. $$

Àcima, (γ) é o sinal de γ. Então, para um vetor $v\in\mathbb{R}^{p\times1}$,

$$ P(v,\lambda\omega)=[S(v[1],\lambda) \ldots S(v[p],\lambda)]^\top $$

2. No caso do item 2, não existe uma fórmula explícita para $P(v,\lambda\omega)$. Mas podemos usar a função `sortedL1Prox()` do pacote `SLOPE` escrito em R. Veja SLOPE, assim como introduções em An intro to SLOPE e Proximal Operators. Em nossa notação, `sortedL1Prox(v,lambda*omega)` computa $P(v,\lambda\omega)$ para a sequência ω.

Antes precisaremos criar um código para boter ler a função `sortedL1Prox()` em Python. Para tanto, precisaremos da biblioteca `rpy2`. Para instalá-lo chame `pip install rpy2` no terminal base de sua maquina. Em R, o pacote `SLOPE` precisa estar instalado e fazer o chamado `library(SLOPE)`.

In [1]:
```
import matplotlib
import numpy as np
import scipy
import seaborn as sns
import matplotlib.pyplot as plt
import numpy.linalg as la

from rpy2 import robjects
from rpy2.robjects.packages import importr
```

In [2]:
```
def sortedL1Prox(v,seq):
    '''
    Função que toma (v,seq) e retorna P(v,seq).
    '''
    # R representation in Python of (v,seq):
    r_var = robjects.vectors.FloatVector(np.hstack((v,seq)))
    # Python function to call R function sortedL1Prox()
    func = robjects.r(
    '''
```

```
    library(SLOPE)
    function(v,seq) sortedL1Prox(v,seq)
    '''
    )
    # Calling Python function sortedL1Prox using R representation r_var:
    return np.asarray(func(r_var[0:len(r_var)//2], r_var[len(r_var)//2:]))
```

In [3]:
```
#Exemplo:
v = 90.0*np.arange(10)
omega = np.ones(10)
lambd = 2.5
seq = lambd*omega
v.shape, omega.shape
```

Out[3]:
```
((10,), (10,))
```

In [4]:
```
sortedL1Prox(v,seq)
```

Out[4]:
```
array([  0. ,  87.5, 177.5, 267.5, 357.5, 447.5, 537.5, 627.5, 717.5,
       807.5])
```

Exercício 1: Gerando dados

1. Construa uma função `data_genP(p,s,b_mag)` que toma p, $s\in[p]$ e número positivo `b_mag` e retorna vetor p-dimensional cujas s-ézimas primeiras coordenadas são b_{mag} e as outras são zero.

In [5]:
```
#Escreva código aqui
def data_genP(p:int, s:int, b_mag:float) -> np.ndarray:
    x = np.zeros(p)
    x[0: s] = b_mag
    return x.reshape(-1,1)
```

In [6]:
```
#Exemplo:
p=100
s=10
b_mag=2
b_true = data_genP(p,s,b_mag)
b_true
```

```
Out[6]: array([[2.],
       [2.],
       [2.],
       [2.],
       [2.],
       [2.],
       [2.],
       [2.],
       [2.],
       [2.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
```

```
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.]])
```

2. Construa uma função `data_genXe(n,p,b_true,sd)` que toma n, p, b_{true} e um número positivo `sd` e constrói uma matriz de dados `X` $n\times p$ cujas linhas são vetores $x_i^\top$ normais padrão p-dimensionais independentes e o vetor `y` de dimensão n cujas coordenadas satisfazem

$$ y_i = x_i^\top b_{\text{true}} + \text{sd}\cdot\epsilon_i, $$ onde $\{\epsilon_i\}_{i=1}^n$ é uma sequência iid de normais padrão.

In [7]:
```python
#Escreva código aqui
def data_genXe(n:int, p:int, b_true:np.ndarray, sd:float) -> list[np.ndarray]:
    X = np.random.normal(loc=0, scale=1, size=(n, p))

    y = X @ b_true + sd * np.random.normal(loc=0, scale=1, size=n).reshape(-1,1)

    return X, y
```

In [8]:
```python
#Exemplo:
sd = 0.1
n = 5

b_true = data_genP(p,s,b_mag)
X,y = data_genXe(n,p,b_true,sd)
X,y
```

Out[8]:
```
(array([[ 7.79450049e-01, -3.60720701e-01, -8.07910587e-01,
         -2.32556243e+00,  1.38738207e+00,  1.02386639e+00,
         -9.63775042e-02, -5.38455941e-01,  1.07010421e-01,
         -8.31367460e-01, -1.02504508e+00, -1.39745983e+00,
          6.22854457e-01, -1.01736397e+00, -7.74675202e-01,
         -1.95197531e-01,  1.01807788e-01, -5.01093297e-01,
          4.30595215e-01, -9.06138977e-01, -2.26901518e-01,
          2.79299633e-01, -1.39539775e-01,  1.00321211e+00,
         -5.85746685e-01,  1.86425535e-01, -1.41410218e+00,
         -1.25289103e+00, -7.70241645e-01,  6.07520380e-01,
          7.57798625e-01, -1.91813545e-01,  3.96631019e-01,
         -9.43345560e-02,  1.40441510e+00, -1.71249897e+00,
          8.20440982e-01, -1.13533756e+00, -1.54868967e+00,
          3.82693364e-01,  1.39655848e+00,  2.62679034e-01,
          1.80822650e-01, -1.65468585e+00,  4.44031903e-01,
         -1.09253250e+00,  1.54885443e+00, -4.02867023e-01,
          1.73816196e+00,  1.35893931e+00,  3.56420247e-01,
         -7.41964513e-01,  5.62207956e-01, -1.09027066e+00,
          7.23065509e-01,  1.86620659e+00,  1.40528326e-01,
          6.60223121e-01,  3.27055242e-01, -6.48571036e-02,
          9.01895747e-02,  9.12684099e-01, -6.12658954e-01,
         -2.21752785e-01, -2.69347236e-01, -1.11399444e+00,
         -8.88738094e-01,  6.08886285e-01,  7.23165067e-01,
         -1.48329186e-01, -1.25614250e+00,  1.26675336e-01,
          4.06518828e-01,  8.55130638e-01, -1.04315947e+00,
          4.43346903e-01,  5.44498384e-01,  4.72021422e-01,
         -9.87429370e-02,  2.05478739e-01, -2.48908316e-01,
          2.11025945e-01, -1.18079177e+00,  7.21012254e-01,
         -1.61465927e-01, -1.24210529e+00, -6.66212876e-01,
         -3.60410821e-01, -9.35779209e-01, -1.00616009e+00,
          2.38828888e-01,  2.42699477e+00, -1.14938583e+00,
         -6.22661579e-01, -2.88316029e-02,  5.02718678e-01,
          1.02072063e+00,  2.84000332e-01, -1.08326672e+00,
          1.41179601e+00],
        [ 1.58666147e+00,  5.20601790e-01,  1.03830639e+00,
          2.02789150e+00, -5.33094357e-01,  9.53776961e-01,
         -1.04456436e+00,  1.18936899e+00,  1.81010886e+00,
          2.88842601e-01, -7.53256431e-02,  1.17244336e+00,
          1.50798658e+00,  4.21062131e-01, -3.09266586e-01,
         -1.39019137e-01,  7.40088804e-01,  7.11411743e-01,
          2.73049339e-01,  5.17545694e-03, -2.79607623e-01,
          1.34906709e+00,  4.32042890e-01,  1.76930818e-01,
          5.12809550e-01, -5.84273095e-01, -7.81517811e-01,
          1.93471905e-01,  7.72261991e-01, -1.19652780e+00,
          1.15400803e+00, -5.20262308e-01, -1.05108675e+00,
          6.03780259e-01, -6.03721981e-01, -8.76274263e-01,
         -1.93215295e-01, -6.28161739e-01, -2.61684846e-02,
          8.77312354e-01, -2.31480449e-01,  2.92889960e-01,
          1.87619928e+00,  8.38321386e-01, -3.78793963e-01,
          7.82259253e-01, -1.82669453e-01, -1.83063818e-01,
         -3.50559685e-01,  4.64132651e-01, -6.36560726e-01,
         -1.37882540e+00, -1.13821477e+00,  5.28155908e-01,
          5.48582656e-01, -2.67573299e+00,  1.53712315e+00,
         -6.05972326e-01,  6.88478654e-01,  1.39409500e-01,
         -1.80588722e+00,  9.28960985e-01,  1.11911081e+00,
          1.09088954e+00, -1.85821313e+00,  1.26339781e+00,
         -2.77425862e-01, -1.12329571e+00, -1.76247651e-01,
          1.26327346e+00, -1.61227081e+00,  3.54807009e-01,
         -7.66538132e-01,  7.24279187e-01, -1.09962603e-01,
         -3.21370837e-01,  2.38617256e-01, -1.28896164e+00,
         -2.25741192e-01,  3.59268939e-01,  1.11480134e+00,
         -7.45358237e-01, -4.22185297e-02,  2.18848166e-01,
          6.17724768e-01, -5.56312797e-01, -6.36630328e-01,
         -4.04435748e-01,  1.80986852e+00,  3.65640594e-01,
          5.92289893e-01, -1.45859297e+00, -1.24150515e+00,
          2.23755081e-01,  1.05893449e+00, -4.23317883e-01,
          1.56744191e+00, -1.78218613e+00,  9.15746221e-01,
          1.05796311e+00],
        [-4.15385726e-01,  7.78841216e-03, -4.65523823e-01,
         -1.15744132e+00,  1.17147747e+00,  1.02395902e+00,
         -1.42853670e+00,  1.35502566e+00, -6.25836814e-01,
          6.84121691e-01, -4.76315159e-02,  1.55059476e+00,
         -1.01414220e-01,  3.74330241e-01, -9.51050223e-01,
         -4.10251840e-01, -1.20624061e+00,  2.05273598e-01,
          4.95293108e-01,  6.49195175e-02,  1.08550218e+00,
          3.20044943e+00, -5.51830634e-01,  3.47490077e-01,
         -6.72681489e-01,  9.41818826e-01,  1.27776320e+00,
          1.55320934e+00,  8.09872422e-01,  9.51798796e-02,
```

```
 -4.32673365e-01,  2.15585645e+00,  8.02239712e-01,
  1.21572634e-01,  2.06345970e-01,  1.39547913e+00,
 -1.34132268e-01, -1.05101771e+00,  1.50370560e-01,
 -4.34061196e-01,  8.30607001e-01,  1.61191208e+00,
  4.28424557e-01, -2.69145984e-01,  5.05720608e-01,
 -7.76023037e-01,  9.04343789e-01,  4.92153436e-01,
  1.07049344e-01, -1.35066289e+00,  1.15370540e+00,
 -9.13130006e-01, -2.70753227e-02, -6.38322819e-01,
  3.83812417e-02,  1.01238023e+00,  7.06358082e-01,
 -8.99564506e-01,  1.49937969e+00, -9.25900010e-01,
 -4.63416799e-01, -4.52038255e-01,  2.10666522e+00,
 -1.05945176e+00, -8.46308517e-01,  1.25150571e+00,
  1.85062407e+00, -3.48926028e-01, -7.02488323e-01,
 -1.94121662e-01, -2.37960577e+00, -5.92565743e-01,
  6.50495313e-01, -5.89806938e-02, -1.51579257e-01,
 -2.14310796e-02,  4.97073651e-01,  8.87981656e-01,
  3.19228371e-01, -3.52011513e-01, -6.49868965e-01,
  6.71031083e-01,  1.21284873e+00, -9.12338010e-01,
  7.50732170e-01, -1.53091557e+00,  1.07776871e-01,
  7.51569241e-01,  1.62463396e+00, -5.96541546e-01,
  7.50035414e-01, -4.44733268e-01,  4.24020490e-01,
 -4.48167886e-01, -1.29312578e-01,  1.51660931e+00,
 -1.02841314e+00,  1.19601473e-01, -7.92490978e-01,
 -1.48033120e+00],
[-5.42940703e-01, -1.65348153e+00, -5.06529172e-01,
 -2.41634016e-01,  1.89786059e+00,  9.78095364e-02,
  2.07116530e-01, -6.89492382e-01, -1.77060246e+00,
  6.25001494e-01, -5.51111927e-02, -9.37788637e-01,
  9.74371963e-01, -1.38251940e+00, -3.55490285e-01,
  7.93481086e-02, -1.20031179e+00,  2.51347489e+00,
 -5.83421143e-01,  1.17276714e+00, -6.80446064e-01,
 -1.68452442e-01,  6.36101767e-01,  1.79657542e-01,
 -3.66263687e-01, -2.87042864e-01,  1.71711825e+00,
  1.58674818e+00, -6.15513495e-01,  1.76905068e+00,
  1.59734873e+00,  9.89211610e-01,  1.18638504e+00,
  8.14119397e-01,  6.54494203e-02, -2.90354727e-01,
 -6.07238180e-01, -1.76172044e-01,  1.69936284e+00,
 -1.45834549e+00,  9.41804658e-02, -1.06711659e+00,
  1.49938542e+00, -1.06742481e+00,  3.39658513e-01,
  1.39469832e-01, -2.63120902e-01, -5.62160288e-01,
 -1.52448608e+00,  1.13214476e+00, -4.33197244e-01,
  5.85525992e-01, -9.69439658e-01,  5.60907625e-01,
  5.00494266e-01,  7.75267019e-02,  2.10170296e+00,
  1.78956322e-01,  2.19349926e-01, -1.17565124e+00,
 -1.93663860e-01,  1.66874243e+00, -1.72214464e+00,
 -1.05728256e+00, -1.14850645e+00, -1.48644969e+00,
  1.78763847e+00,  2.87591828e+00,  1.16258607e+00,
  1.11487266e+00,  6.76286232e-01, -1.25383485e-01,
  1.31904835e+00,  6.61999745e-01, -6.21390888e-01,
 -6.76321907e-02, -5.08410051e-01,  2.46697366e+00,
 -1.78355590e-02,  1.28660780e+00,  8.00001497e-02,
 -2.38226003e-01, -1.09734587e+00,  3.38084376e-01,
 -1.61193700e+00,  1.48733878e-01, -1.19111046e+00,
  3.19840399e-02, -8.68945305e-01,  1.74271751e+00,
  6.88713936e-01,  2.29584082e-03,  7.91333795e-01,
 -1.10911843e+00, -1.99932693e+00,  4.87183768e-02,
 -7.44390435e-01, -9.01455228e-01, -3.23809327e-01,
  7.22003603e-02],
[-9.46426499e-01, -3.04008631e-01,  7.72748610e-01,
 -7.36345542e-01,  8.92408198e-01, -3.66473987e-01,
 -1.59940450e-01, -2.31054692e-01, -1.51626530e+00,
  1.02548567e-01, -1.18065700e+00,  5.15654523e-02,
 -6.06892103e-01,  6.82445970e-01, -1.33092897e+00,
 -3.66874029e-01,  1.53145212e+00,  7.26186393e-01,
  9.00488820e-02,  2.05357321e+00, -6.50917607e-01,
  7.95451662e-01,  1.41177196e+00,  3.52060159e-01,
  2.38180965e-02,  9.86527763e-01, -1.16213533e+00,
 -1.42390696e+00, -2.12852389e-01,  7.68465769e-01,
  1.58567657e-01, -2.61081272e-02, -2.54498129e-01,
  1.85137667e+00, -4.78496639e-01, -1.11455457e+00,
  4.24009084e-01, -3.97493655e-01, -1.45731754e+00,
  1.85812146e+00, -1.80777832e-01, -2.01200852e+00,
 -1.45294317e+00,  1.08938188e+00,  6.54025238e-02,
 -4.99356839e-01,  5.78412727e-01, -1.12247938e+00,
  8.97953099e-01,  8.75725922e-01, -2.22360626e+00,
 -3.01122309e-01, -7.31873718e-02,  5.20294376e-01,
  4.01968110e-01, -8.95118884e-01,  2.70963817e-01,
  1.22799265e+00, -6.70079636e-01, -1.74280091e+00,
 -2.00262214e-01,  1.04361969e+00, -3.75010028e-01,
```

```
         -1.57340564e+00,  2.10185959e-02,  1.05542345e-01,
         -1.23818370e-01,  1.58545470e+00, -3.17735566e-01,
         -1.71886316e+00, -1.97960162e-01,  8.23076851e-01,
          2.61294916e-01,  1.29086094e+00, -3.80566811e-02,
          1.46645701e+00,  1.24265114e+00,  1.89921999e-03,
          2.34105776e+00,  1.58849568e-01, -3.81005081e-01,
         -1.49862358e+00,  1.08248689e+00, -1.32771854e+00,
          2.39491624e-02, -7.33958265e-02,  9.25227864e-01,
         -2.26941421e+00,  2.87912470e-01,  7.03707378e-02,
          2.07700942e-01,  8.78137149e-01, -1.06019363e+00,
          1.36890524e+00,  9.79259731e-01,  7.28056470e-01,
         -2.00818402e-01, -4.50500209e-01, -8.89194676e-01,
          1.29060284e-01]]),
 array([[-3.33875597],
        [15.52472174],
        [ 0.1220726 ],
        [-5.35030625],
        [-4.8260223 ]]))
```

Exercício 2:

Vamos usar o método gradiente proximal para resolver o problema àcima:

\begin{align*} w_{k+1} &:= b_k - \frac{1}{L}\nabla f(b_k),\\ b_{k+1} &:= P\left(w_{k+1},\frac{\lambda}{L}\omega\right). \end{align*}

Construa uma função `linear_reg(n,p,SlopeB,X,y,L,lambd,b0,t_final)` onde, $1/L$ é o passo, `lambd` ($=\lambda$) é o fator de penalização, `b0` é o ponto inicial, a variável `t_final` é o número de iterações e:

1. a variável Booleana `SlopeB` vale `False` se a sequêndia $\omega[j]\equiv1$.
2. a variável Booleana `SlopeB` vale `True` se $\omega[j]=\sqrt{\log(2p/j)}$.

A função deve retornar a sequência $k\mapsto\frac{1}{2n}\sum_{i=1}^n(y_i-x_i^\top b_k)^2$ e o último iterado $b_{\text{t final}}$. Use penalização

$$ \lambda = \text{sd}\sqrt{\frac{\log p}{n}}, $$

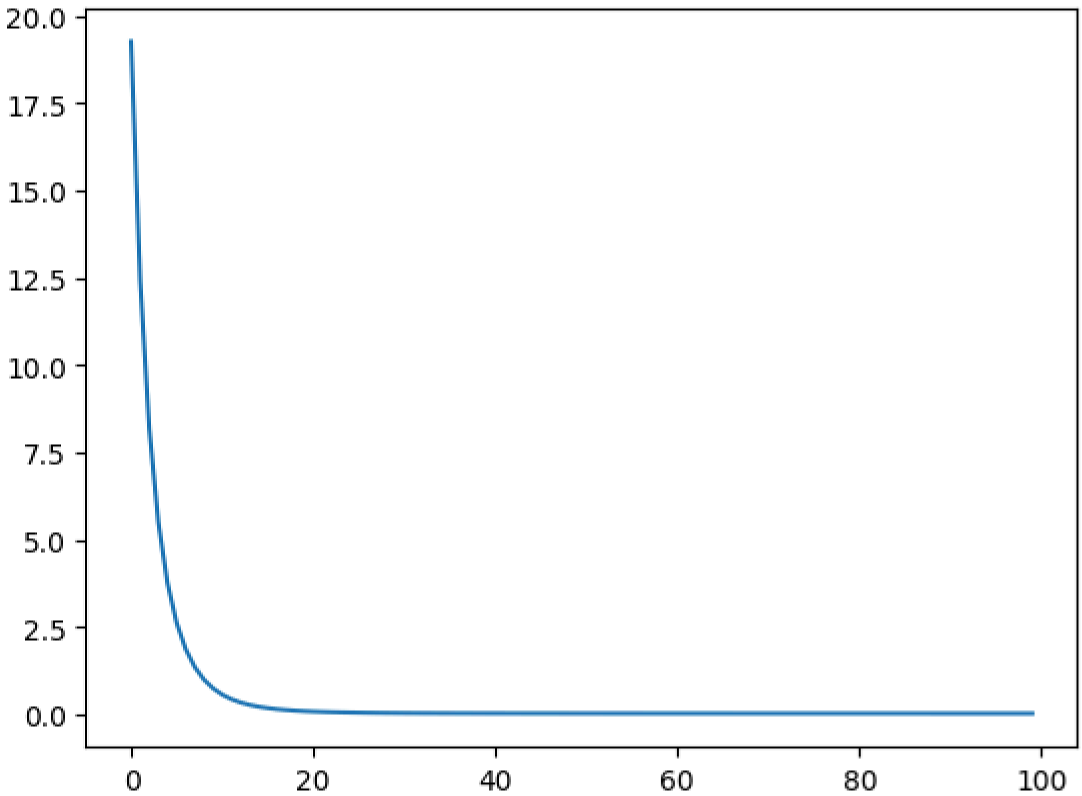
e passo $1/L$ com

$$ L = A\lambda_{\max}\left(\frac{X^\top X}{n}\right), $$ para algum ajuste fino $A>0$.

In [9]:
```python
#Escreva código aqui
def linear_reg(n:int, p:int, SlopeB:bool, X:np.ndarray, y:np.ndarray, L:float, lambd:float, b0:n

    omega = np.ones(p)
    if SlopeB:
        for j in range(1, p + 1):
            omega[j-1] = np.sqrt(np.log(2 * p / j))

    b = b0.reshape(-1,1)
    loss_history = list()

    for each_epoch in range(t_final):
        # grad = -(1/n) * X.T @ (y - X @ b) + lambd * omega[np.argsort(b)]
        grad = -(1/n) * X.T @ (y - X @ b)

        w = b - (1/L) * grad
        w = np.array([i[0] for i in w.tolist()])

        b = sortedL1Prox(w, (lambd/L) * omega).reshape(-1, 1)

        loss = (1/(2*n)) * la.norm(y - X @ b)**2
        loss_history.append(loss)

    return loss_history, b
```

In [10]:
```python
#Exemplo:
b0 = np.asmatrix(np.zeros(p)).T
t_final = 100
SlopeB = True
A = 4
L = A*np.max(la.eigvalsh(X.T @ X / X.shape[0]))
```

```
lambd = sd*np.sqrt(np.log(p)/n)

f1 = linear_reg(n,p,SlopeB,X,y,L,lambd,b0,t_final)
plt.plot(f1[0])
plt.show()
```



Pergunta:

Varie A. O que acontece se A for muito pequeno? Alguma intuição?

Quando A é muito pequeno, ocorre uma convergência à 0 mais veloz, o que se assemelha ao sentido de learning rate, ou, taxa de aprendizado, onde, quando maior a taxa, maiores as atualizações do gradiente, levando a saltos "maiores" das atualizações de pesos, deste modo, como A é inversamente proporcional ao peso que o gradiente terá na atualização de pesos, quando A é muito pequeno, o gradiente dará saltos "maiores", assim, convergindo mais rapidamente para uma perda menor, entretanto, isto pode dificultar para o gradiente convergir para o mínimo global, ficando preso em mínimos locais ou não conseguindo uma boa convergência de saltos pequenos para encontrar o mínimo global. Deste modo, mesmo que um A menor diminua o tanto de atualizações necessárias para um bom resultado, pode influenciar na função não convergir para um mínimo global ótimo.

Exercício 3:

Agora, vamos usar o método gradiente proximal acelerado: iniciando de `b0=z0` e $t_0=1$:

\begin{align*} z_{k+1}&:= P\left(b_k - (1/L)\nabla f(b_k), \frac{\lambda}{L}\omega \right),\\ t_{k+1} &:= \frac{1+\sqrt{1+4t_k^2}}{2},\\ b_{k+1} &:= z_{k+1} + \frac{t_{k}-1}{t_{k+1}}(z_{k+1} - z_{k}). \end{align*}

Construa uma função `linear_reg_acc(n,p,SlopeB,X,y,L,lambd,b0,t_final)` onde, $1/L$ é o passo, `lambd` ($=\lambda$) é o fator de penalização, `b0` é o ponto inicial, a variável `t_final` é o número de iterações e:

1. a variável Booleana `SlopeB` vale `False` se a sequêndia $\omega[j]\equiv1$.
2. a variável Booleana `SlopeB` vale `True` se $\omega[j]=\sqrt{\log(2p/j)}$.

A função deve retornar a sequência $k\mapsto\frac{1}{2n}\sum_{i=1}^n(y_i-x_i^\top b_k)^2$ e o último iterado $b_{\text{t_final}}$. Use penalização

$$ \lambda = \text{sd}\sqrt{\frac{\log p}{n}}. $$

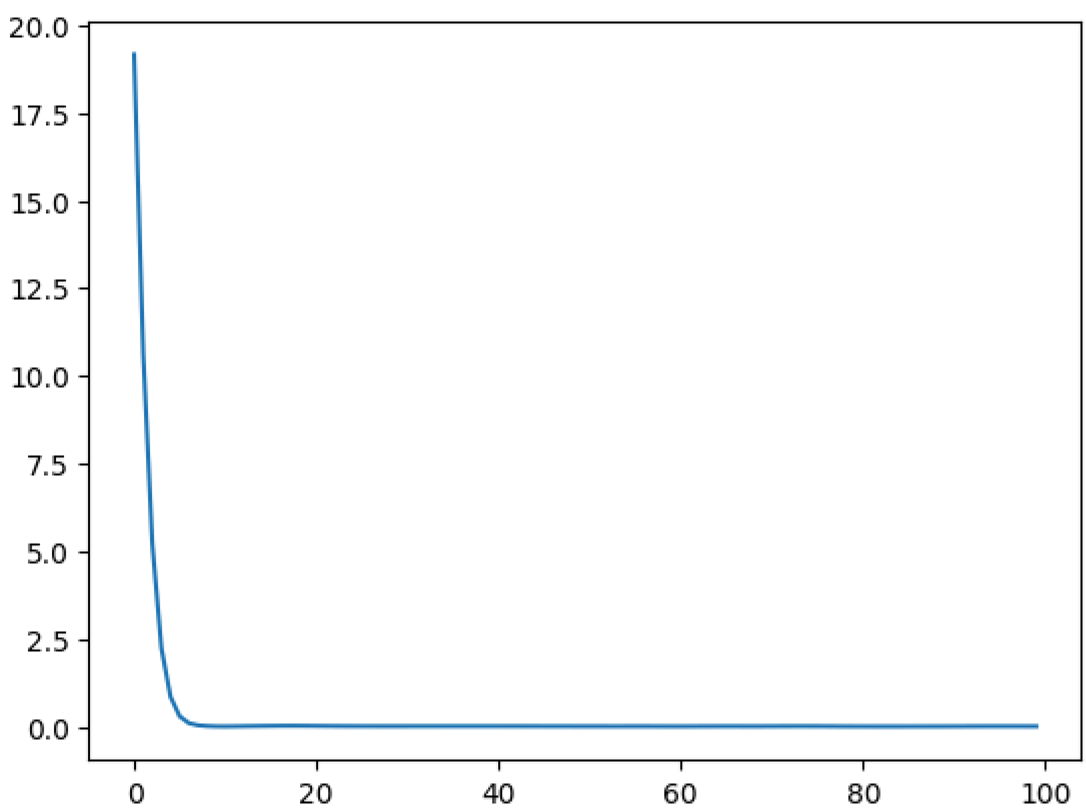
In [11]:
```
#Escreva código aqui
def linear_reg_acc(n:int, p:int, SlopeB:bool, X:np.ndarray, y:np.ndarray, L:float, lambd:float,

    omega = np.ones(p)
```

```
    if SlopeB:
        for j in range(1, p + 1):
            omega[j-1] = np.sqrt(np.log(2 * p / j))

    z_k = b0.reshape(-1,1)
    b = b0.reshape(-1,1)
    t_k = 1
    loss_history = list()

    for each_epoch in range(t_final):
        # grad = -(1/n) * X.T @ (y - X @ b) + lambd * omega[np.argsort(b)]
        grad = -(1/n) * X.T @ (y - X @ b)

        w = b - (1/L) * grad
        w = np.array([i[0] for i in w.tolist()])

        z_kn = sortedL1Prox(w, (lambd/L) * omega).reshape(-1, 1)

        t_kn = (1 + np.sqrt(1 + 4 * t_k**2))/2

        b = z_kn + ((t_k - 1)/t_kn) * (z_kn - z_k)

        loss = (1/(2*n)) * la.norm(y - X @ b)**2
        loss_history.append(loss)

        z_k = z_kn
        t_k = t_kn

    return loss_history, b
```

In [12]:

```
#Exemplo:
b0 = np.asmatrix(np.zeros(p)).T
t_final = 100
SlopeB = False
A = 4
L = A * np.max(la.eigvalsh(X.T @ X / X.shape[0]))
lambd = sd*np.sqrt(np.log(p)/n)

f2 = linear_reg_acc(n,p,SlopeB,X,y,L,lambd,b0,t_final)
plt.plot(f2[0])
```

Out[12]: [<matplotlib.lines.Line2D at 0x7f6c2c561110>]



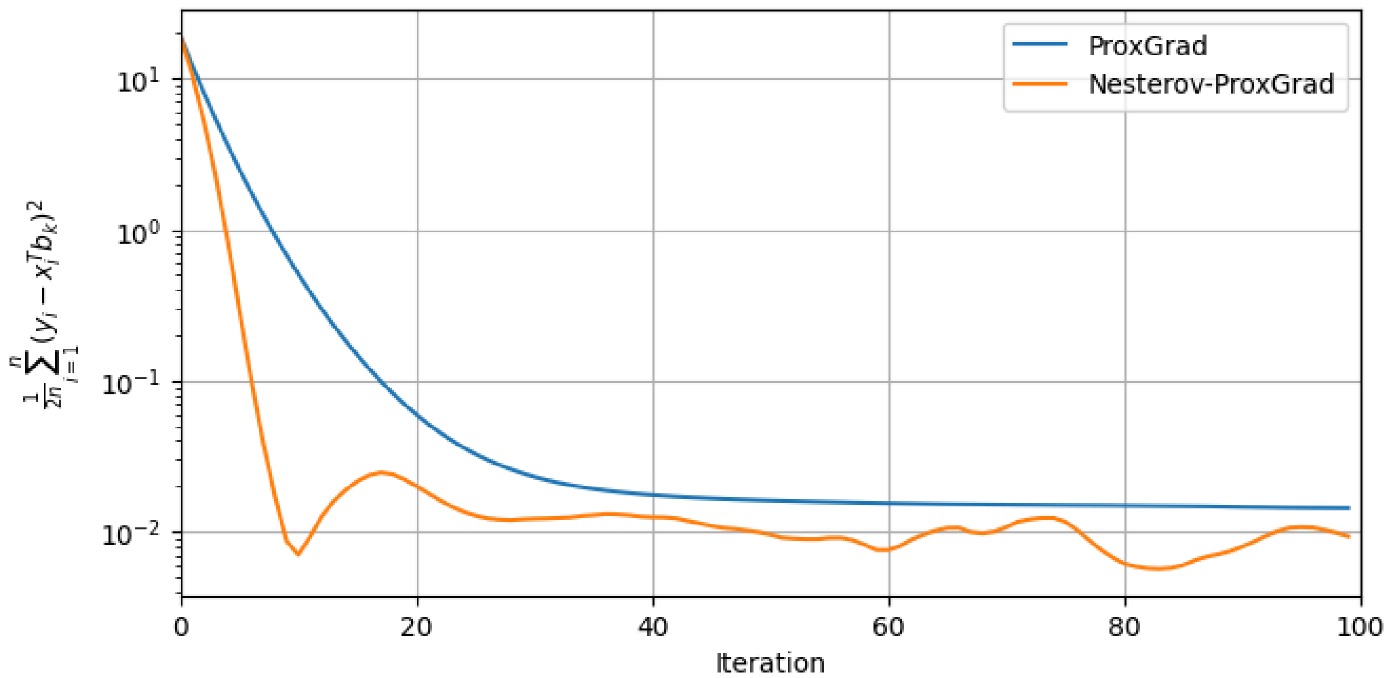
Exercício 4:

Implemente num mesmo gráfico os erros $k\mapsto\frac{1}{2n}\sum_{i=1}^n(y_i-x_i^\top b_k)^2$ de cada método em função no número de iterações.

In [13]:

```
#Escreva código aqui
b0 = np.asmatrix(np.zeros(p)).T
t_final = 100
```

```
SlopeB = False
A = 4
L = A * np.max(la.eigvalsh(X.T @ X / X.shape[0]))
lambd = sd*np.sqrt(np.log(p)/n)

f1 = linear_reg(n,p,SlopeB,X,y,L,lambd,b0,t_final)
f2 = linear_reg_acc(n,p,SlopeB,X,y,L,lambd,b0,t_final)

plt.figure(figsize=(8, 4))
plt.plot(f1[0], label='ProxGrad')
plt.plot(f2[0], label='Nesterov-ProxGrad')
plt.yscale('log')
plt.grid()
plt.xlabel('Iteration')
plt.ylabel('$\\frac{1}{2n}\sum_{i=1}^n(y_i-x_i^Tb_k)^2$')
plt.xlim(0, t_final)
plt.legend()
plt.show()
```

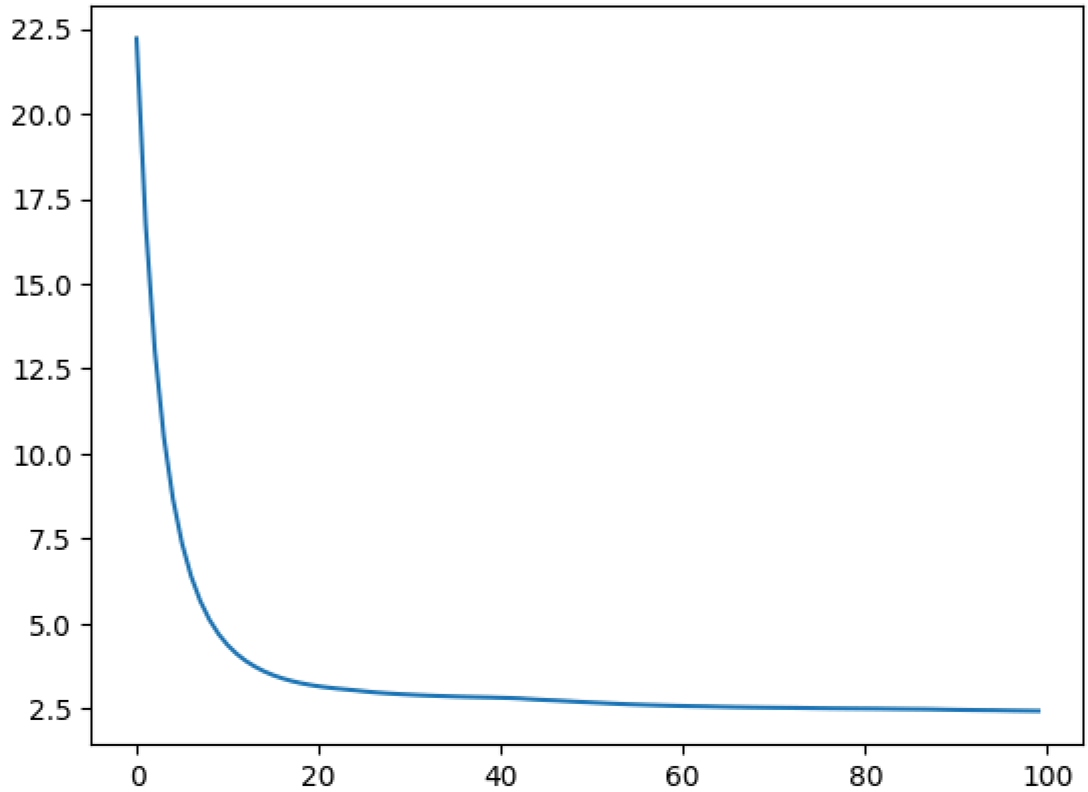


Exercício 5:

Refaça os exercícios com $\sigma=1$ e $\sigma=10$. Há alguma diferença quando $\sigma=10$? Tem alguma intuição de porque isso acontece?

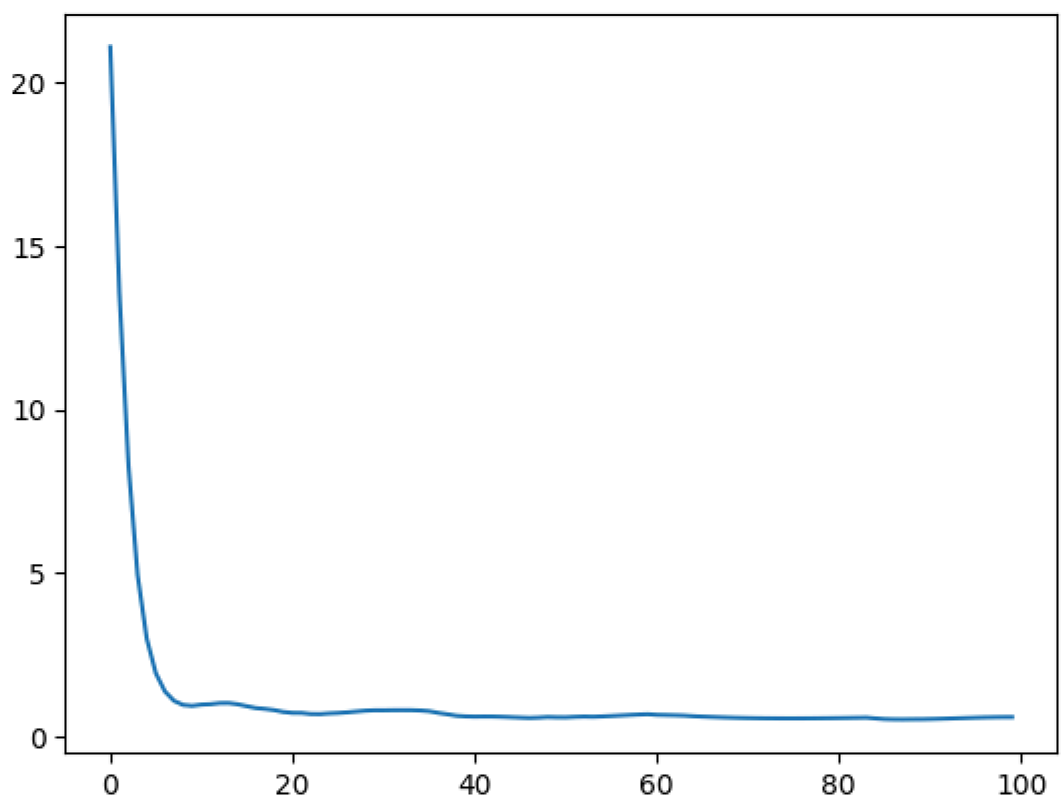
In [14]:
```
sd = 1
```

In [15]:
```
b0 = np.asmatrix(np.zeros(p)).T
t_final = 100
SlopeB = True
A = 4
L = A*np.max(la.eigvalsh(X.T @ X / X.shape[0]))
lambd = sd*np.sqrt(np.log(p)/n)

f1 = linear_reg(n,p,SlopeB,X,y,L,lambd,b0,t_final)
plt.plot(f1[0])
plt.show()
```



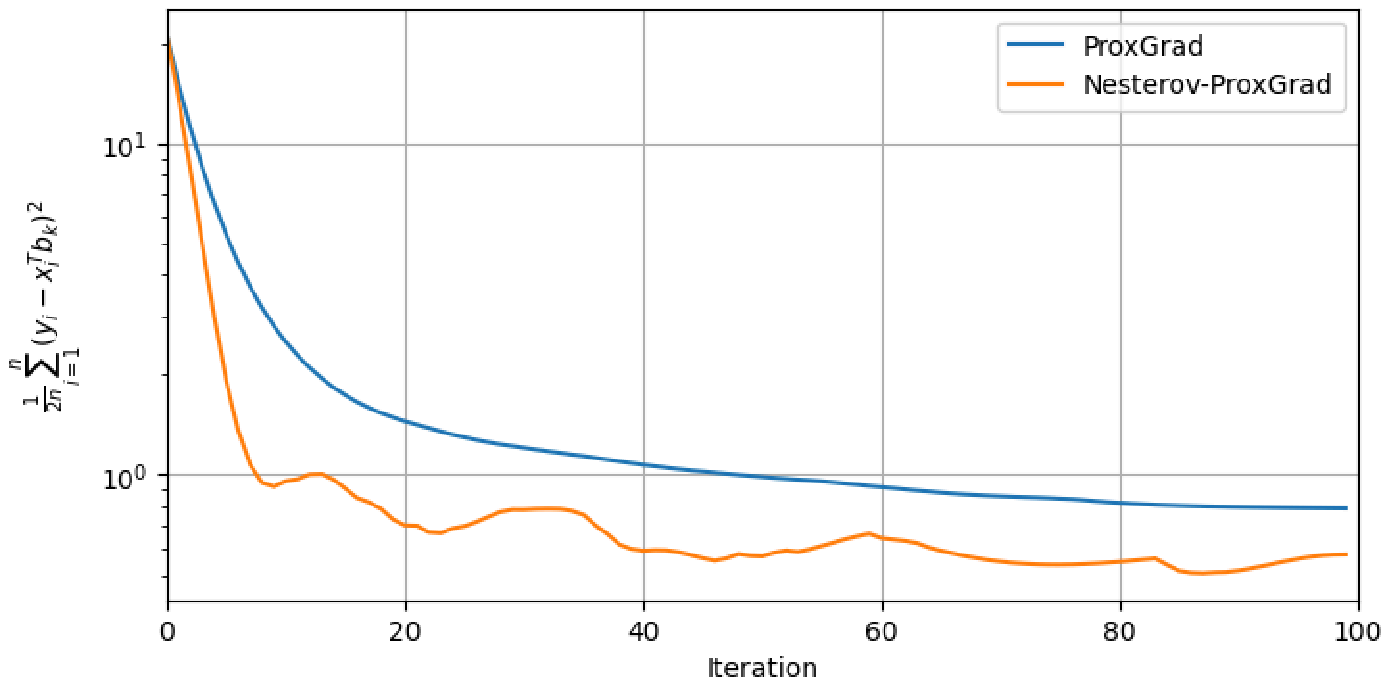
In [16]:
```
b0 = np.asmatrix(np.zeros(p)).T
t_final = 100
SlopeB = False
A = 4
L = A * np.max(la.eigvalsh(X.T @ X / X.shape[0]))
lambd = sd*np.sqrt(np.log(p)/n)

f2 = linear_reg_acc(n,p,SlopeB,X,y,L,lambd,b0,t_final)
plt.plot(f2[0])
```

Out[16]: [<matplotlib.lines.Line2D at 0x7f6bc44c0710>]



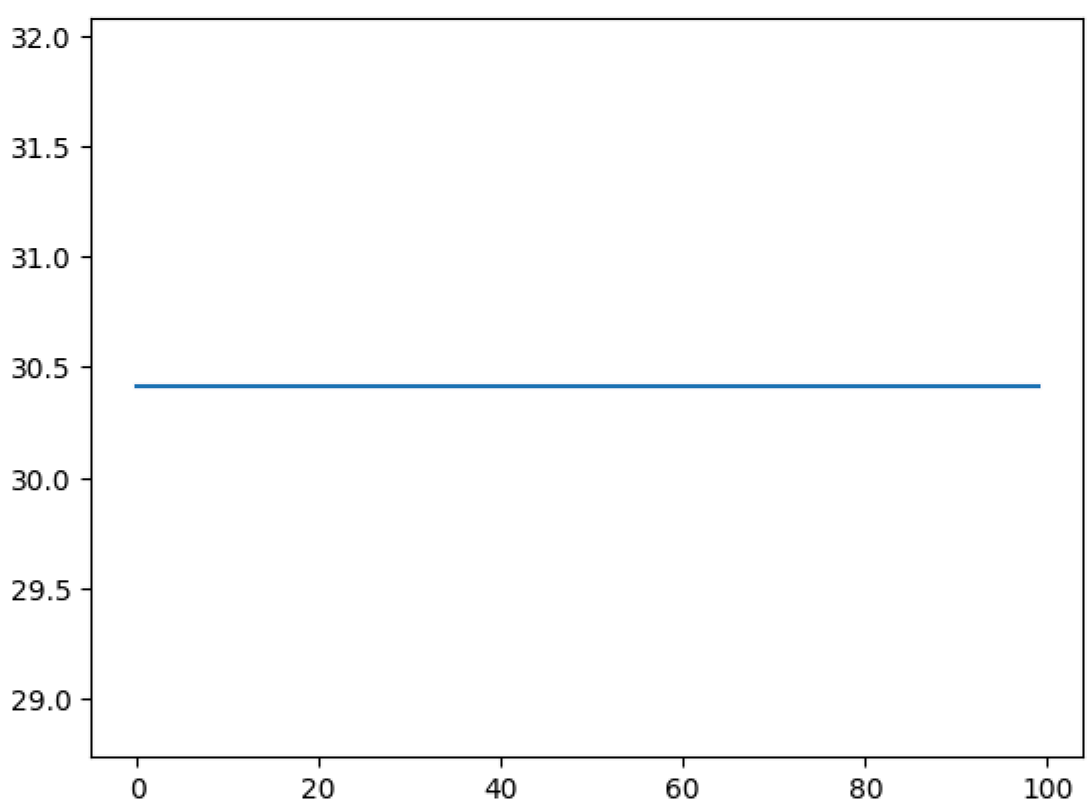
In [17]:
```
b0 = np.asmatrix(np.zeros(p)).T
t_final = 100
SlopeB = False
A = 4
L = A * np.max(la.eigvalsh(X.T @ X / X.shape[0]))
lambd = sd*np.sqrt(np.log(p)/n)

f1 = linear_reg(n,p,SlopeB,X,y,L,lambd,b0,t_final)
f2 = linear_reg_acc(n,p,SlopeB,X,y,L,lambd,b0,t_final)

plt.figure(figsize=(8, 4))
plt.plot(f1[0], label='ProxGrad')
plt.plot(f2[0], label='Nesterov-ProxGrad')
plt.yscale('log')
plt.grid()
plt.xlabel('Iteration')
```

```
plt.ylabel('$\\frac{1}{2n}\sum_{i=1}^n(y_i-x_i^Tb_k)^2$')
plt.xlim(0, t_final)
plt.legend()
plt.show()
```



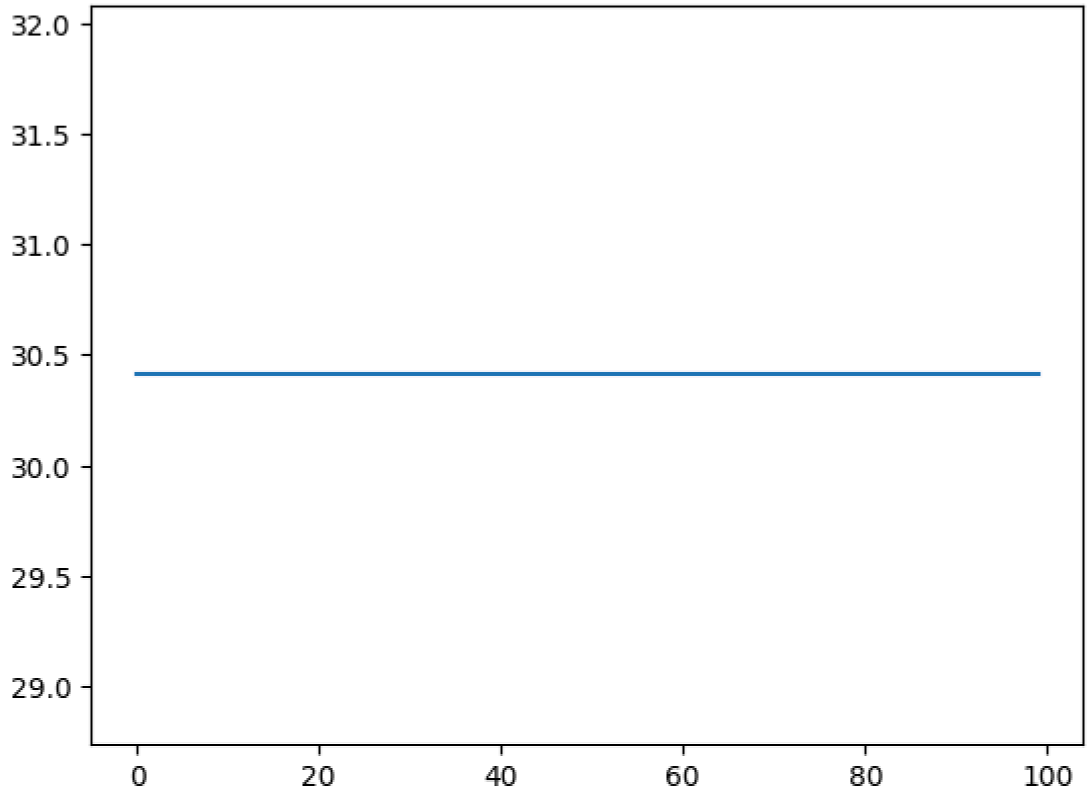
In [18]:
```
sd = 10
```

In [19]:
```
b0 = np.asmatrix(np.zeros(p)).T
t_final = 100
SlopeB = True
A = 4
L = A*np.max(la.eigvalsh(X.T @ X / X.shape[0]))
lambd = sd*np.sqrt(np.log(p)/n)

f1 = linear_reg(n,p,SlopeB,X,y,L,lambd,b0,t_final)
plt.plot(f1[0])
plt.show()
```

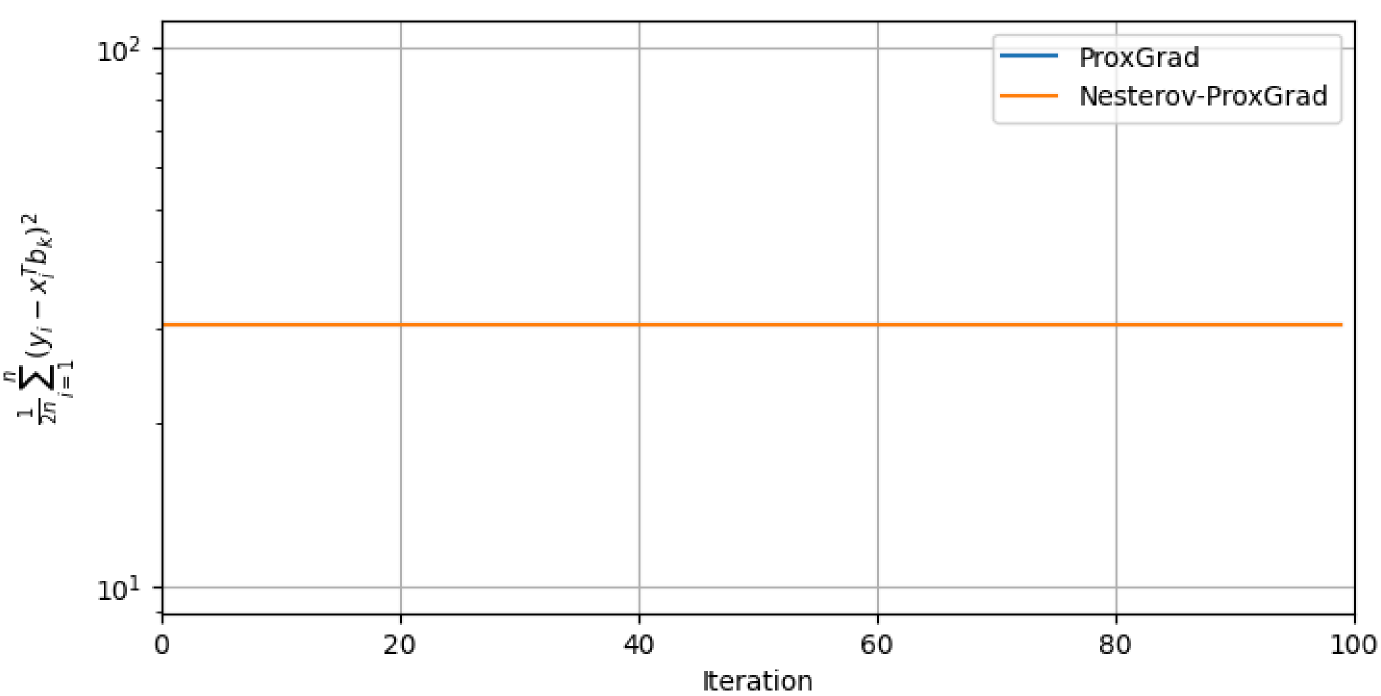


In [20]:
```
b0 = np.asmatrix(np.zeros(p)).T
t_final = 100
SlopeB = False
A = 4
L = A * np.max(la.eigvalsh(X.T @ X / X.shape[0]))
lambd = sd*np.sqrt(np.log(p)/n)

f2 = linear_reg_acc(n,p,SlopeB,X,y,L,lambd,b0,t_final)
plt.plot(f2[0])
```

Out[20]:
```
[<matplotlib.lines.Line2D at 0x7f6bc42511d0>]
```



In [21]:
```
#Escreva código aqui
b0 = np.asmatrix(np.zeros(p)).T
t_final = 100
SlopeB = False
A = 4
L = A * np.max(la.eigvalsh(X.T @ X / X.shape[0]))
lambd = sd*np.sqrt(np.log(p)/n)

f1 = linear_reg(n,p,SlopeB,X,y,L,lambd,b0,t_final)
f2 = linear_reg_acc(n,p,SlopeB,X,y,L,lambd,b0,t_final)

plt.figure(figsize=(8, 4))
plt.plot(f1[0], label='ProxGrad')
plt.plot(f2[0], label='Nesterov-ProxGrad')
plt.yscale('log')
plt.grid()
plt.xlabel('Iteration')
plt.ylabel('$\\frac{1}{2n}\sum_{i=1}^n(y_i-x_i^Tb_k)^2$')
plt.xlim(0, t_final)
plt.legend()
plt.show()
```



É perceptível que alterar o σ altera, e em muito, a descida do grandiente, isto se dá pois o σ impacta diretamente o lambda, que é usado na função de slope, ou seja, quanto maior o σ, mais fácil será para que o máximo entre a diferença da entrada do vetor e o lambda e 0 seja 0, já que quanto maior o lambda, maior deverá ser a entrada do grandiente para que ele seja atualizado, caso contrário continuará estagnado com atualização no 0, ou seja, se a atualização for muito pequena, algo que é decidido pelo lambda, a função não realiza essa atualização.