Completação de matrizes (*Matrix Completion*)

O objetivo deste projeto é implementar alguns métodos iterativos de otimização para o *completamento de matrizes*. Por exemplo, o problema de estimar as notas/avaliações faltantes de itens de consumo dadas por usuários (sabendo-se uma amostra pequena). O Netflix se utiliza de algoritmos deste tipo para sugerir filmes ao usuário. De fato, existe toda uma área chamada *Sistemas de Recomendação (Recommendation Systems)* que trata de problemas do tipo.

No description has been provided for this image
Para facilitar, iremos à seguir apresentar algumas definições. Denotaremos por bfX^* a matriz $n_1 \times n_2$ não observada. As entradas observadas da matriz bfX^* correspondem aos índices:

$$\Omega := \{(i,j) : \text{ usuário } i \text{ avalia item } j\}.$$

Definiremos o *operador de amostragem* $\mathcal{P}_\Omega : re^{n_1\times n_2} \to re^{n_1\times n_2}$ que leva uma matriz $\mathbf{A} \in re^{n_1\times n_2}$, à matriz

$$(\mathcal{P}_\Omega[\mathbf{A}])_{i,j} = \begin{cases} \mathbf{A}_{i,j}, & \text{if } (i,j) \in \Omega, \\ 0, & \text{if } (i,j) \notin \Omega. \end{cases}$$

A operação complementar é

$$\left(\mathcal{P}_\Omega^\perp[\mathbf{A}]\right)_{i,j} = \begin{cases} 0, & \text{if } (i,j) \in \Omega, \\ \mathbf{A}_{i,j}, & \text{if } (i,j) \notin \Omega. \end{cases}$$

A matriz observada pode ser escrita como

$$\mathbf{Y} := \mathcal{P}_\Omega[bfX^*].$$

Portanto, as entradas não observadas correspondem à zero na matriz $\mathbf{Y}$.

Soft-Thresholding

Dado números $\alpha > 0$ and $\gamma \in re$,

$$calS_\alpha(\gamma) := sign(\gamma) \cdot \max\{\gamma - \alpha, 0\}.$$

é chamado *soft-thresholding* de γ com threshold α. Àcima, $sign(\gamma)$ é o sinal de γ.

Seja uma matriz $\mathbf{W}$ com decomposição de valores singulares (SVD):

$$\mathbf{W} = \mathbf{U} \cdot \mathbf{D}(\gamma_1, \dots, \gamma_r) \cdot \mathbf{V}^\top.$$

Então, para $\alpha > 0$, definimos a matriz *soft-thresholding* de $\mathbf{W}$ com threshold α por

$$\mathcal{S}_\alpha(\mathbf{W}) := \mathbf{U} \cdot \mathbf{D}\left(\mathcal{S}_\alpha(\gamma_1), \dots, \mathcal{S}_\alpha(\gamma_r)\right) \cdot \mathbf{V}^\top.$$

Norma de Frobenius e norma nuclear

Dada matrix $\mathbf{X}$, sua *norma de Frobenius* é dada por

$$\|bfX\|_F := \sqrt{\sum_{i=1}^{n_1}\sum_{j=1}^{n_2} bfX_{ij}^2}.$$

A *norma nuclear* de bfX é dada por

$$\|bfX\|_N := \sum_{i=1}^{r} \sigma_i(\mathbf{X}),$$

onde r é o posto de $\mathbf{X}$ e $\sigma_1(\mathbf{X}) > \cdots > \sigma_r(\mathbf{X})$ são os valores singulares de $\mathbf{X}$.

Exercício 1: Gerando dados

Primeiro, construa a função `data_genX(n,r,B_mag,m)` que retorna 3 matrizes $n \times n$: (1) a matriz de dados bfX^* de posto r e autovalores todos iguais a B_{mag}, (2) a matriz Ω contendo 1 se a entrada correspondente é observada e 0 caso contrário e (3) a matriz $\Omega^{\perp}$ contendo 0 se a entrada correspondente é observada e 1 caso contrário. Ao gerar $\mathbf{X}^*$, use a função `scipy.stats.ortho_group` para construir duas matrizes $\mathbf{U}$ e $\mathbf{V}$ aleatórias ortogonais de dimensões $n \times r$; retorne $\mathbf{X}^* = \mathbf{U} \cdot D(B_{mag}, \dots, B_{mag}) \cdot \mathbf{V}^{\top}$. Ao construir Ω, dado $m = 1, \dots, n^2$, selecione aleatoriamente as m entradas de $\mathbf{X}^*$ observadas.

In [1]:
```
#Escreva código aqui
import numpy as np
from scipy.stats import ortho_group

def data_genX(n, r, B_mag, m):
    U = ortho_group.rvs(dim=n)[:, :r]
    V = ortho_group.rvs(dim=n)[:, :r]

    D = np.diag([B_mag] * r)
    X_new = U @ D @ V.T

    observed_indices = np.random.choice(n * n, m, replace=False)

    Omega = np.zeros((n, n))
    for index in observed_indices:
        i, j = divmod(index, n)
        Omega[i, j] = 1

    Omega_perp = 1 - Omega

    return X_new, Omega, Omega_perp
```

In [2]:
```
#Exemplo:
n=3
r=1
B_mag=1
m=2
data = data_genX(n,r,B_mag,m)
data
```

Out[2]:
```
(array([[ 0.02523989, -0.02607868, -0.00089101],
        [ 0.2833069 , -0.29272191, -0.01000122],
        [ 0.63440403, -0.6554869 , -0.02239555]]),
 array([[0., 1., 0.],
        [0., 0., 0.],
        [0., 1., 0.]]),
 array([[1., 0., 1.],
        [1., 1., 1.],
        [1., 0., 1.]]))
```

Exercício 2: Funções auxiliares

1. Construe uma função `P_omega(X.Omega)` cujas entradas são uma matriz $n \times n$ bfX e Ω e retorna a matriz $\mathcal{P}_{\Omega}(bfX)$.
2. Construe uma função `soft(x,l)` que toma dois números x e l retorna $\mathcal{S}_l(x)$.
3. Construe uma função `Frob_sq(A)` que retorna o quadrado da norma de Frobenius da matriz A, isto é,

$\|\mathbf{A}\|_F^2 := \sum_{i,j} A_{ij}^2$.

In [3]:
```
#Escreva código aqui
def P_omega(X, Omega):
    return X * Omega
```

In [4]:
```
#Exemplo:
P_omega(data[0],data[1])
```

Out[4]:
```
array([[ 0.        , -0.02607868, -0.        ],
       [ 0.        , -0.        , -0.        ],
       [ 0.        , -0.6554869 , -0.        ]])
```

In [5]:
```python
#Escreva código aqui
def soft(x, l):
    return np.sign(x) * np.maximum(abs(x) - l, 0)
```

In [6]:
```python
#Exemplo:
soft(10,2.1)
```

Out[6]: 7.9

In [7]:
```python
#Escreva código aqui
def Frob_sq(A):
    return np.sum(A ** 2)
```

In [8]:
```python
#Exemplo:
Frob_sq(data[0])
```

Out[8]: 1.0000000000000002

Exercício 3: Regularized Matrix Completion (RMC)

Iremos resolver estimar $\mathbf{X}^*$ usando um método iterativo que resolve o seguinte problema de otimização:

$$\begin{aligned} \min_{\mathbf{X}} \quad & \tfrac{1}{2}\|\mathbf{X}\|_F^2 + \lambda\|\mathbf{X}\|_N \\ \text{s.t.} \quad & \mathcal{P}_\Omega[\mathbf{X}] = \mathbf{Y}. \end{aligned} \tag{1}$$

Escolhendo $\lambda, L > 0$ e ponto inicial $\mathbf{W}^0$, o seguinte método resolve o problema àcima:

$$\mathbf{X}^k := \mathcal{S}_\lambda\left(\mathbf{W}^k\right), \tag{2}$$

$$\mathbf{W}^{k+1} := \mathbf{W}^k - \frac{n}{L}\left[\mathcal{P}_\Omega(\mathbf{X}^k) - \mathbf{Y}\right] \tag{3}$$

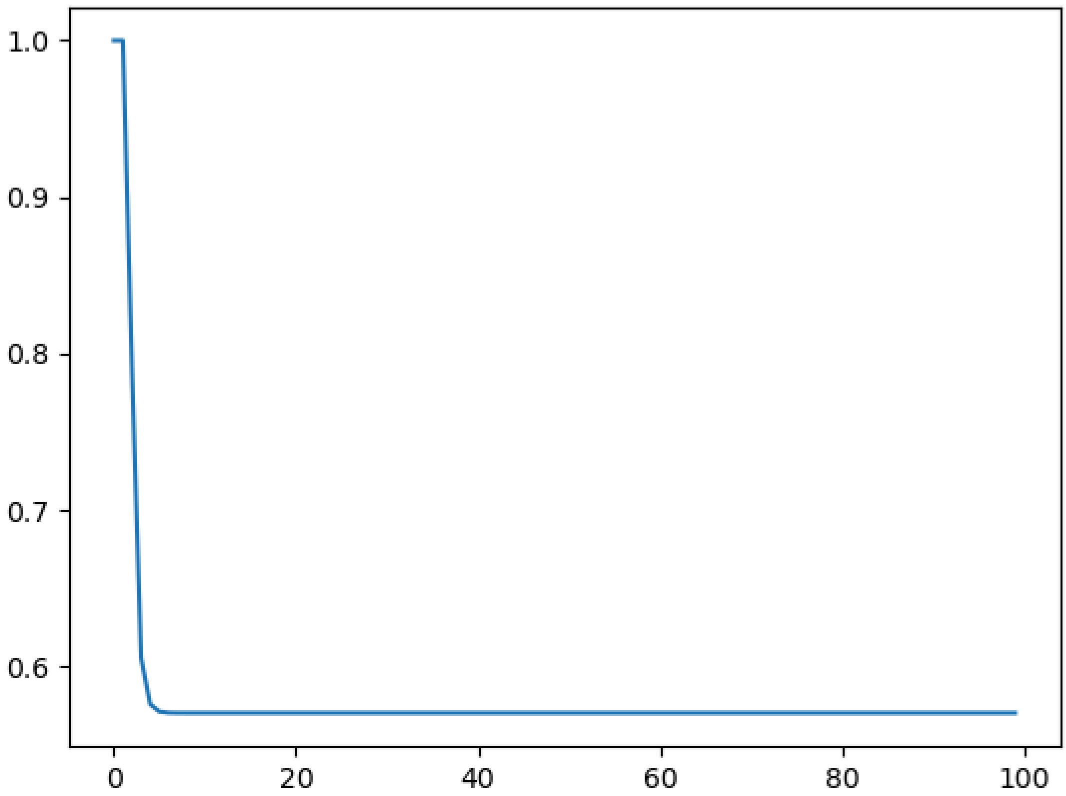
Referência: (https://arxiv.org/abs/0810.3286)

Para tanto, construa uma função `SVT(l,L,X,Y,Omega,it)` que toma números positivos $\lambda = l$ e L, as matrizes $\mathbf{X}^*$, $\mathbf{Y}$, e Ω e um número de iterações escolhido e retorna os últimos iterados $(\mathbf{X}^{it}, \mathbf{W}^{it})$ e a sequência de erros $\|\mathbf{X}^k - \mathbf{X}^*\|^2$.

In [9]:
```python
#Escreva código aqui
from scipy.linalg import svd
def SVT(l, L, X_star, Y, Omega, it):
    n = X_star.shape[0]
    W = np.zeros_like(X_star)
    errors = []

    for k in range(it):
        U, sigma, Vt = svd(W, full_matrices=False)
        sigma_thresholded = soft(sigma, l)
        X = U @ np.diag(sigma_thresholded) @ Vt

        error = Frob_sq(X - X_star)
        errors.append(error)

        P_X_minus_Y = P_omega(X, Omega) - Y
        W = W - (n / L) * P_X_minus_Y

    return X, W, errors
```

In [10]:
```python
#Exemplo:
X = data[0]
Omega = data[1]
Y = P_omega(X,Omega)
n = X.shape[0]
l = n/5
L = 5
it = 100

f1 = SVT(l,L,X,Y,Omega,it)
error_SVT = f1[2]

import matplotlib.pyplot as plt
```

```
plt.plot(error_SVT)
plt.show()
```



Exercício 4: Stable Matrix Completion (SMC)

Em seguida, iremos resolver estimar $\mathbf{X}^*$ usando um método iterativo que resolve o seguinte problema de otimização:

$$\min_{\mathbf{X}} \quad \frac{1}{2}\|\mathbf{Y} - \mathcal{P}_\Omega[\mathbf{X}]\|_F^2 + \lambda\|\mathbf{X}\|_N. \tag{4}$$

Escolhendo $\lambda, L > 0$ e ponto inicial $\mathbf{W}^0$, o seguinte método resolve o problema àcima:

$$\begin{aligned} \mathbf{X}^k &:= \mathcal{S}_{\frac{\lambda}{L}}\left(\mathbf{W}^k\right), \\ \mathbf{W}^{k+1} &:= \mathbf{X}^k - \frac{1}{L}\left[\mathcal{P}_\Omega(\mathbf{X}^k) - \mathbf{Y}\right]. \end{aligned}$$

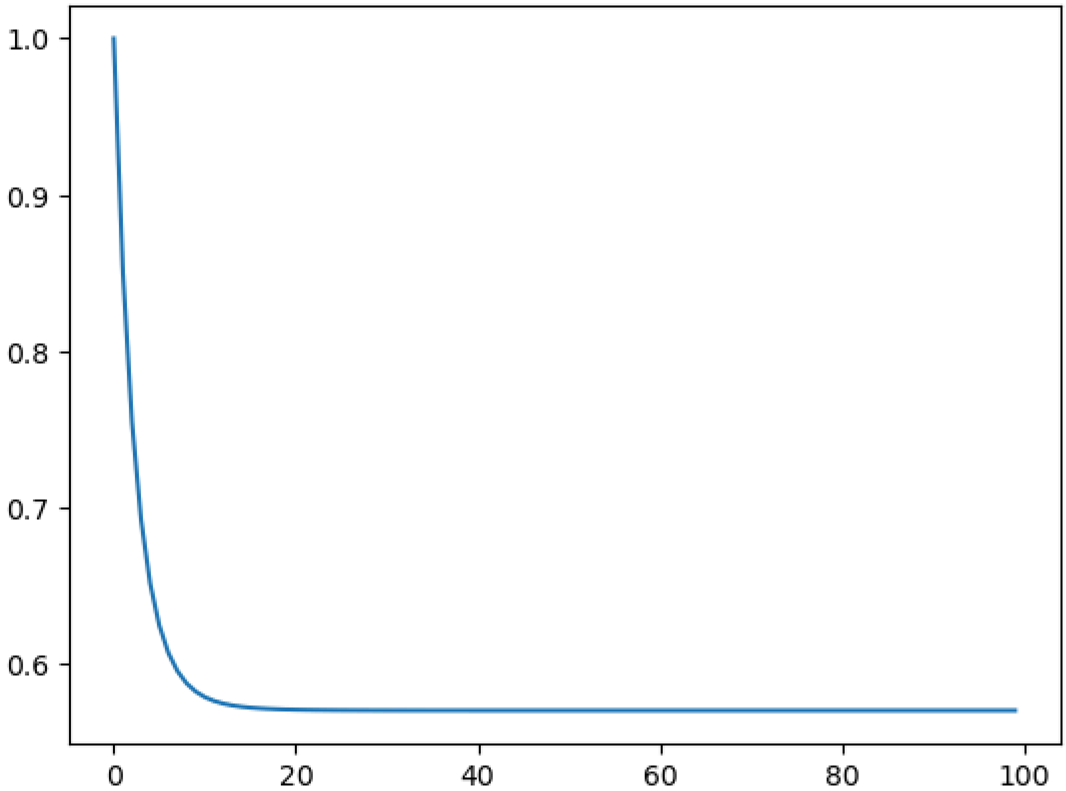
Referência: (https://optimization-online.org/2009/03/2268/)

Para tanto, construa uma função `PG(l,L,X,Y,Omega,it)` que toma números positivos $\lambda = l$ e L, as matrizes $\mathbf{X}^*$, $\mathbf{Y}$, e Ω e um número de iterações escolhido e retorna os últimos iterados $(\mathbf{X}^{it}, \mathbf{W}^{it})$ e a sequência de erros $\|\mathbf{X}^k - \mathbf{X}^*\|^2$.

In [11]:
```
#Escreva código aqui
def PG(l, L, X_star, Y, Omega, it):
    W = np.zeros_like(X_star)
    errors = []

    for k in range(it):
        U, sigma, Vt = svd(W, full_matrices=False)
        sigma_thresholded = soft(sigma, l / L)
        X = U @ np.diag(sigma_thresholded) @ Vt

        error = Frob_sq(X - X_star)
        errors.append(error)

        P_X_minus_Y = P_omega(X, Omega) - Y
        W = X - (1 / L) * P_X_minus_Y

    return X, W, errors
```

In [12]:
```
#Exemplo:
X = data[0]
Omega = data[1]
Y = P_omega(X,Omega)
n = X.shape[0]
l = n/100
L = 5
it = 100
```

```
f2 = PG(l,L,X,Y,Omega,it)
error_PG = f2[2]

import matplotlib.pyplot as plt
plt.plot(error_PG)
plt.show()
```



Exercício 5: Accelerated SMC

Em seguida, iremos resolver estimar $\mathbf{X}^*$ usando um método iterativo *com aceleração de Nesterov* que resolve SMC. Escolhendo $\lambda, L > 0$, inicializações $\mathbf{X}^0 = \mathbf{X}^1$ e $t_0 = t_1 = 1$, o seguinte método resolve o problema àcima:

$$\begin{aligned}
\mathbf{Z}^k &:= \mathbf{X}^k + \frac{t_{k-1}-1}{t_k}(\mathbf{X}^k - \mathbf{X}^{k-1}), labelalgo: APG1 \\
\mathbf{W}^k &:= \mathbf{Z}^k - \frac{1}{L}\left[\mathcal{P}_\Omega(\mathbf{Z}^k) - \mathbf{Y}\right], \\
\mathbf{X}^{k+1} &:= \mathcal{S}_{\frac{\lambda}{L}}\left(\mathbf{W}^k\right), \\
t_{k+1} &:= \frac{1+\sqrt{1+4t_k^2}}{2}.
\end{aligned}$$

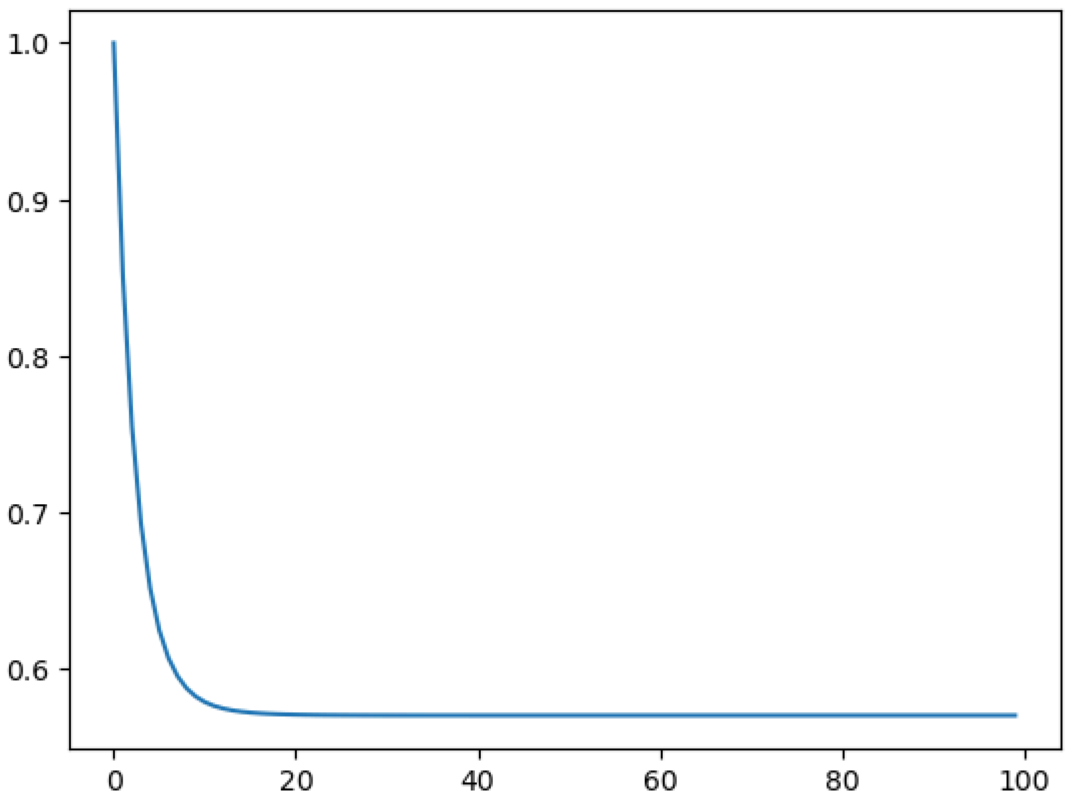
Referência: (https://optimization-online.org/2009/03/2268/)

Construa uma função `APG(l,L,X,Y,Omega,it)` que toma números positivos $\lambda = l$ e L, as matrizes $\mathbf{X}^*$, $\mathbf{Y}$, e Ω e um número de iterações escolhido e retorna os últimos iterados $(\mathbf{X}^{it}, \mathbf{W}^{it})$ e a sequência de erros $\|\mathbf{X}^k - \mathbf{X}^*\|^2$.

In [13]:
```
#Escreva código aqui
def APG(l, L, X_star, Y, Omega, it):
    X_prev = np.zeros_like(X_star)
    X = np.zeros_like(X_star)
    t_prev = 1
    t = 1
    errors = []

    for k in range(it):
        Z = X + ((t_prev - 1) / t) * (X - X_prev)

        P_Z_minus_Y = P_omega(Z, Omega) - Y
        W = Z - (1 / L) * P_Z_minus_Y

        U, sigma, Vt = svd(W, full_matrices=False)
        sigma_thresholded = soft(sigma, l / L)
        X_next = U @ np.diag(sigma_thresholded) @ Vt

        error = Frob_sq(X_next - X_star)
```

```
        errors.append(error)

        t_next = (1 + np.sqrt(1 + 4 * t**2)) / 2
        X_prev, X = X, X_next
        t_prev, t = t, t_next

    return X, W, errors
```

In [14]:
```
#Exemplo:
X = data[0]
Omega = data[1]
Y = P_omega(X,Omega)
n = X.shape[0]
l = n/100
L = 5
it = 100

f3 = APG(l,L,X,Y,Omega,it)
error_APG = f3[2]

import matplotlib.pyplot as plt
plt.plot(error_PG)
plt.show()
```



Exercício 6: SOFT-IMPUTE

Finalmente, iremos ver outro algoritmo iterativo cuja heurística é bem diferente dos métodos anteriores baseados num problema de otimização regularizado. A idéia aqui se baseia em *imputar* iterativamente a matriz observada. Escolhendo-se $\lambda > 0$ e ponto inicial $\mathbf{X}^0 = 0$, iteramos:

$$\mathbf{X}^{k+1} := \mathcal{S}_\lambda\left(\mathbf{Y} + \mathcal{P}_\Omega^\perp(\mathbf{X}^k)\right).$$

Equivalentemente:

$$\begin{aligned} \mathbf{W}^k &:= \mathbf{Y} + \mathcal{P}_\Omega^\perp(\mathbf{X}^k), \\ \mathbf{X}^{k+1} &:= \mathcal{S}_\lambda\left(\mathbf{W}^k\right). \end{aligned}$$

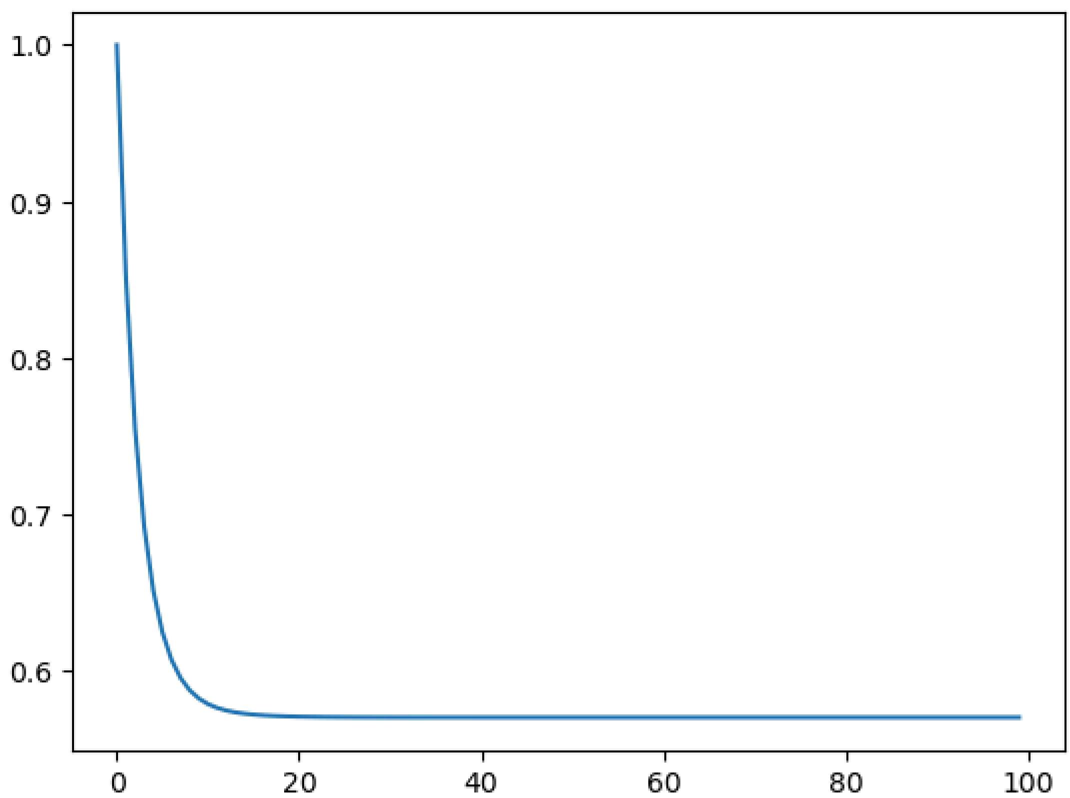
Referência: (https://jmlr.org/papers/v11/mazumder10a.html)

Construa uma função `SImp(l,X,Y,Omega_perp,it)` que toma número positivo $\lambda = l$, as matrizes $\mathbf{X}^*$, $\mathbf{Y}$, e $\Omega^\perp$ e um número de iterações escolhido e retorna os últimos iterados $(\mathbf{X}^{it}, \mathbf{W}^{it})$ e a sequência de erros $\|\mathbf{X}^k - \mathbf{X}^*\|^2$.

In [15]:
```
#Escreva código aqui
def SImp(l, X_star, Y, Omega_perp, it):
    X = np.zeros_like(X_star)
    errors = []
```

```
    for k in range(it):
        W = Y + P_omega(X, Omega_perp)

        U, sigma, Vt = svd(W, full_matrices=False)
        sigma_thresholded = soft(sigma, l)
        X_next = U @ np.diag(sigma_thresholded) @ Vt

        error = Frob_sq(X_next - X_star)
        errors.append(error)

        X = X_next

    return X, W, errors
```

In [16]:
```
#Exemplo:
X = data[0]
Omega = data[1]
Omega_perp = data[2]
Y = P_omega(X,Omega)
n = X.shape[0]
l = n/100
it = 100

f4 = SImp(l,X,Y,Omega_perp,it)
error_SImp = f4[2]

import matplotlib.pyplot as plt
plt.plot(error_PG)
plt.show()
```



Exercício 7: Plots

Implemente num mesmo gráfico os erros quadráticos de cada método em função no número de iterações com os seguintes parâmetros: `n=10 r=1 B_mag=1 m = 10*n*r`

e:

1. SVT: `l = n/5, L = 5, it = 100`
2. PG: `l = n/100, L = 5, it = 100`
3. APG: `l = n/100, L = 5, it = 100`
4. SImp: `l = n/100, it = 100`

In [17]:
```
n = 10
r = 1
B_mag = 1
m = 10 * n * r
iterations = 100
```

```python
lambda_svt = n / 5
L_svt = 5
lambda_pg = n / 100
L_pg = 5
lambda_apg = n / 100
L_apg = 5
lambda_simp = n / 100

X_star, Omega, Omega_perp = data_genX(n, r, B_mag, m)
Y = P_omega(X_star, Omega)

X_SVD, W_SVD, error_svt = SVT(lambda_svt, L_svt, X_star, Y, Omega, iterations)
X_PG, W_PG, error_pg = PG(lambda_pg, L_pg, X_star, Y, Omega, iterations)
X_APG, W_APG, error_apg = APG(lambda_apg, L_apg, X_star, Y, Omega, iterations)
X_SImp, W_SImp, error_simp = SImp(lambda_simp, X_star, Y, Omega_perp, iterations)

plt.figure(figsize=(10, 6))
plt.plot(range(iterations), error_svt, label="SVT", color="blue")
plt.plot(range(iterations), error_pg, label="PG", color="orange")
plt.plot(range(iterations), error_apg, label="APG", color="green")
plt.plot(range(iterations), error_simp, label="SImp", color="red")
plt.xlabel("Iteration")
plt.ylabel("$||X^k-X*||^2_F$")
plt.yscale("linear")
plt.title("Erro Quadrático por Iteração para cada Método")
plt.legend()
plt.grid(True)
plt.show()
```