Recomendação via Fatorização de matrizes (Matrix factorization)

Num projeto anterior, fizemos o completamento de matrizes (por exemplo para sistemas de recomendação como o Netflix) usando uma abordagem *convexa*: algoritmos de otimização de um problema com regularização convexa (norma nuclear).

Uma segunda abordagem bem sucedida é estimar a matriz não observada X^* resolvendo o problema:

$$\min_X f(X) = f(U,V) := \frac{1}{2}\|UV^\top - Y\|_F^2,$$

onde $Y \in \mathbb{R}^{m\times n}$ é a matriz observada, $U \in \mathbb{R}^{m\times r}$, $V \in \mathbb{R}^{n\times r}$ and $r < \min\{m,n\}$. O gradiente é dado por

$$\nabla f(X) = [(UV^\top - Y)V, (UV^\top - Y)^\top U].$$

Note que diferentemente do caso convexo (onde tínhamos a penalização $\lambda > 0$ da norma nuclear como hyper-parâmetro), nesta abordagem temos uma estimativa do posto r como hyper-parâmetro. A idéia aqui é "regularizar" a solução com posto r impondo a fatorização $UV^\top$. Do ponto de vista computacional, a diferença é que o problema é não convexo. Entretanto, algoritmos de otimização iterativos funcionam bem na prática.

Iremos utilizar os dados Movielens 100K dataset. Em particular usamos o arquivo `u.data` desta pasta, gravado em `~/datasets`. Este arquivo tem avaliações de filmes de 943 usuários e 1682 filmes. Começamos carregando alguns módulos necessários:

In [ ]:
```
#Chamando módulos necessários:
import numpy as np
import scipy.linalg as LA
import pandas as pd
import seaborn as sns
import matplotlib
from matplotlib import pyplot as plt
import scipy.sparse as spr
import scipy.sparse.linalg as spr_LA
```

À seguir iremos carregar os dados e escrevê-los numa matriz esparsa Y.

In [2]:
```
#Carregando dados:
names = ['user_id', 'item_id', 'rating', 'timestamp']
df = pd.read_csv('datasets/u.data', sep='\t', names=names)
n_users = df.user_id.unique().shape[0]
n_items = df.item_id.unique().shape[0]

#Criando a matriz Y de avaliações:
ratings = np.zeros((n_users, n_items))
for row in df.itertuples():
    ratings[row[1]-1, row[2]-1] = row[3]

Y = ratings
m, n = Y.shape
print("As dimensões de Y são:", m, n)
```

```
As dimensões de Y são: 943 1682
```

Exercício 1: Funções auxiliares

1. Construa uma função `f()` que toma $X = (U,V)$ e retorna o valor funcional $f(X) = \frac{1}{2}\|UV^\top - Y\|_F^2$.
2. Construa uma função `df()` que toma $X = (U,V)$ e retorna o gradiente $\nabla f(X) = [(UV^\top - Y)V, (UV^\top - Y)^\top U]$.
3. Construa uma função `J()` que dada matriz D retorna a norma de Frobenius $\|D\|_F$.

In [3]:
```
#Escreva código aqui
def J(D:np.ndarray) -> float:
    return np.linalg.norm(D, 'fro')
```

```python
def f(X:tuple[np.ndarray, np.ndarray], Y:np.ndarray) -> float:
    U, V = X
    diff = U @ V.T - Y
    return (J(diff)**2) / 2

def df(X:tuple[np.ndarray, np.ndarray], Y:np.ndarray) -> tuple[np.ndarray, np.ndarray]:
    U, V = X
    diff = U @ V.T - Y
    dU = diff @ V
    dV = diff.T @ U
    return dU, dV
```

Inicialização

À seguir ponha `r=20` e inicialize $X_0 = (U_0, V_0)$ aleatoriamente de uma normal multivariada. Para tanto use `np.random.randn()`. Ponha `N=30000` para o número de iterações.

In [4]:
```python
r = 20

# the starting point
np.random.seed(0)
X0 = (np.random.randn(m, r), np.random.randn(n, r))

# number of iterations
N = 30000
```

Exercício 2: Método do gradiente

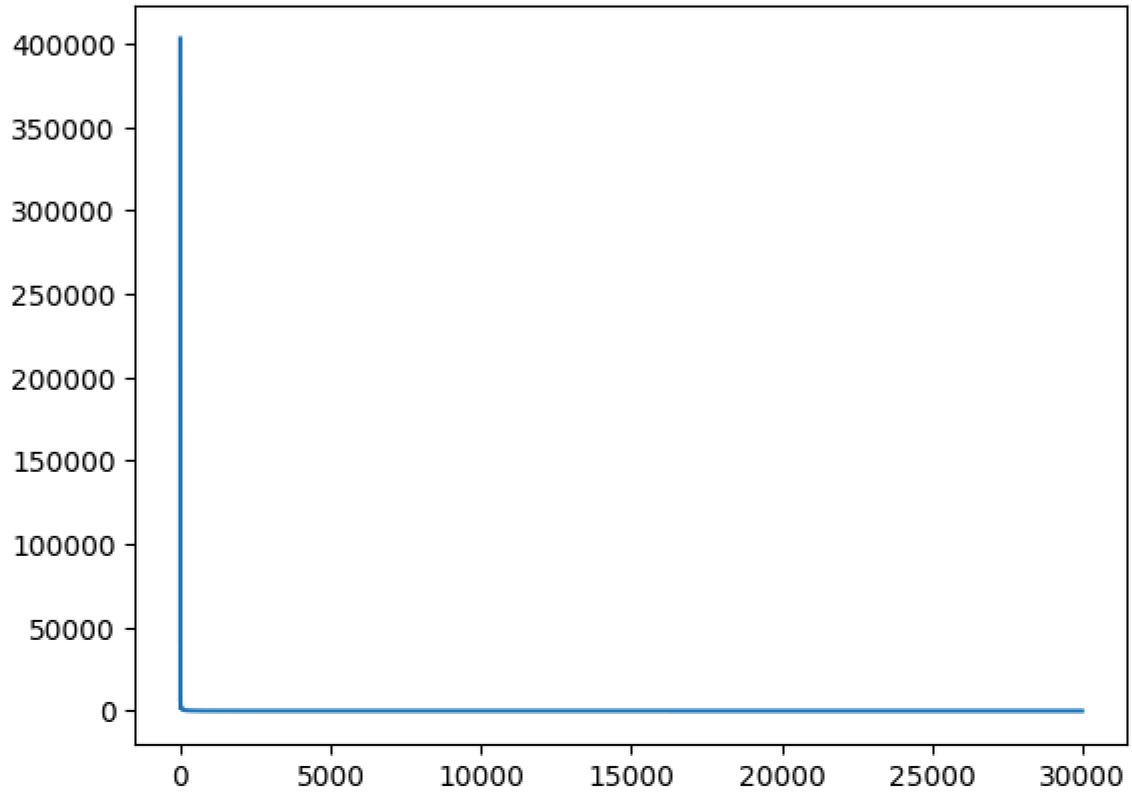
Construa uma função `gd(J, df, x0, la=1, numb_iter=100)` que toma como entrada as funções `J()`, `df`, o ponto inicial `x0`, o passo `la` e o número de iterações `numb_iter` e implementa o método gradiente iniciando de `x0`. Esta função deve retornar a sequência de valores da função `J(df(x))` em cada um dos iterados `x`, isto é, a sequência das normas dos gradientes ao longo da trajetória do método. A função também deve retornar o último iterado.

Implemente a função com passo `la=1./L` com `L=1000`.

In [5]:
```python
#Escreva código aqui
def gd(J, df, X0, la=1, numb_iter=100):
    U, V = X0[0].copy(), X0[1].copy()
    grad_norms = list()

    for i in range(numb_iter):
        dU, dV = df((U, V), Y)

        grad_norm = J(dU) + J(dV)
        grad_norms.append(float(grad_norm))

        U -= la * dU
        V -= la * dV

    return grad_norms, (U, V)
```

In [6]:
```python
# gradient descent
L = 1000
f1 = gd(J, df, X0, 1./L, numb_iter=N)
plt.plot(f1[0])
```

Out[6]: `[<matplotlib.lines.Line2D at 0x7f79f04354d0>]`



Exercício 3: Método do gradiente acelerado

Construa uma função `accel_gd(J, df, x0, la=1, numb_iter=100)` que toma como entrada as funções `J()`, `df`, o ponto inicial `x0`, o passo `la` e o número de iterações `numb_iter` e implementa o método gradiente com aceleração de Nesterov iniciando de `y_0=x0` e $t_0 = 1$:

$$\begin{aligned} X_{k+1} &:= Y_k - la\nabla f(Y_k), \\ t_{k+1} &:= \frac{1+\sqrt{1+4t_k^2}}{2}, \\ Y_{k+1} &:= X_{k+1} + \frac{t_k - 1}{t_{k+1}}(X_{k+1} - X_k). \end{aligned}$$

Esta função deve retornar a sequência de valores da função `J(df(y))` em cada um dos iterados `y`, isto é, a sequência das normas dos gradientes de Y_k ao longo da trajetória do método. A função também deve retornar o último iterado.

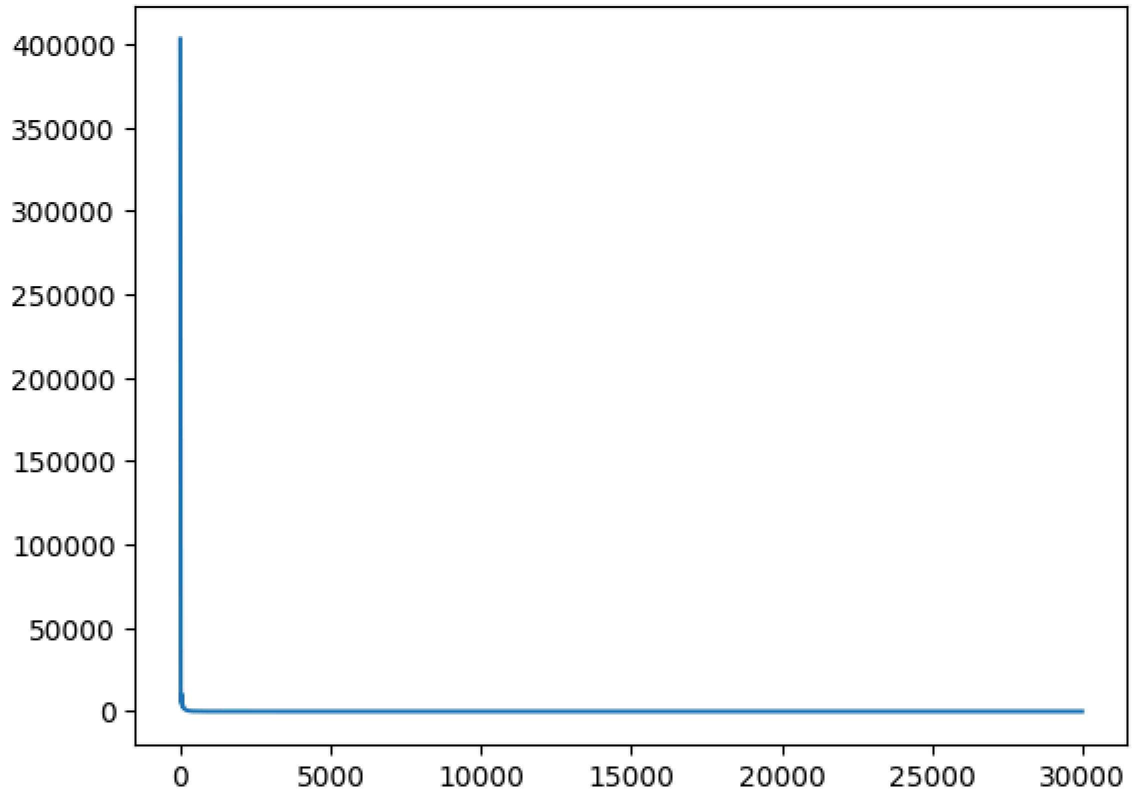
Implemente a função com passo `la=1./L` com `L=30000`.

In [7]:
```
#Escreva código aqui
def accel_gd(J, df, x0, la=1, numb_iter=100):
    U, V = X0[0].copy(), X0[1].copy()
    Y_U, Y_V = U.copy(), V.copy()
    t = 1
    grad_norms = list()

    for i in range(numb_iter):
        dU, dV = df((Y_U, Y_V), Y)

        grad_norm = J(dU) + J(dV)
        grad_norms.append(float(grad_norm))

        X_U = Y_U - la * dU
        X_V = Y_V - la * dV

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

        Y_U = X_U + ((t - 1) / t_next) * (X_U - U)
        Y_V = X_V + ((t - 1) / t_next) * (X_V - V)

        U, V = X_U, X_V
        t = t_next

    return grad_norms, (U, V)
```

In [8]:
```
# Nesterov accelerated gradient descent
L = 30000
```

```
f2 = accel_gd(J, df, X0, 1./L, numb_iter=N)
plt.plot(f2[0])
```

Out[8]: `[<matplotlib.lines.Line2D at 0x7f79ccaf7a90>]`



Exercício 4: Adagrad-Norm

Construa uma função `ad_grad_norm(J, df, x0, b0=0.5, eta=1, numb_iter=100)` que toma como entrada as funções `J()`, `df`, o ponto inicial `x0`, parametros positivos `b0` e `eta` e o número de iterações `numb_iter` e implementa o método Adagrad-Norm iniciando de `x0`:

$$X_{k+1} := X_k - \frac{\eta}{\sqrt{b_0^2 + \sum_{j=1}^{k} \|\nabla f(X_j)\|_2^2}} \nabla f(X_k).$$

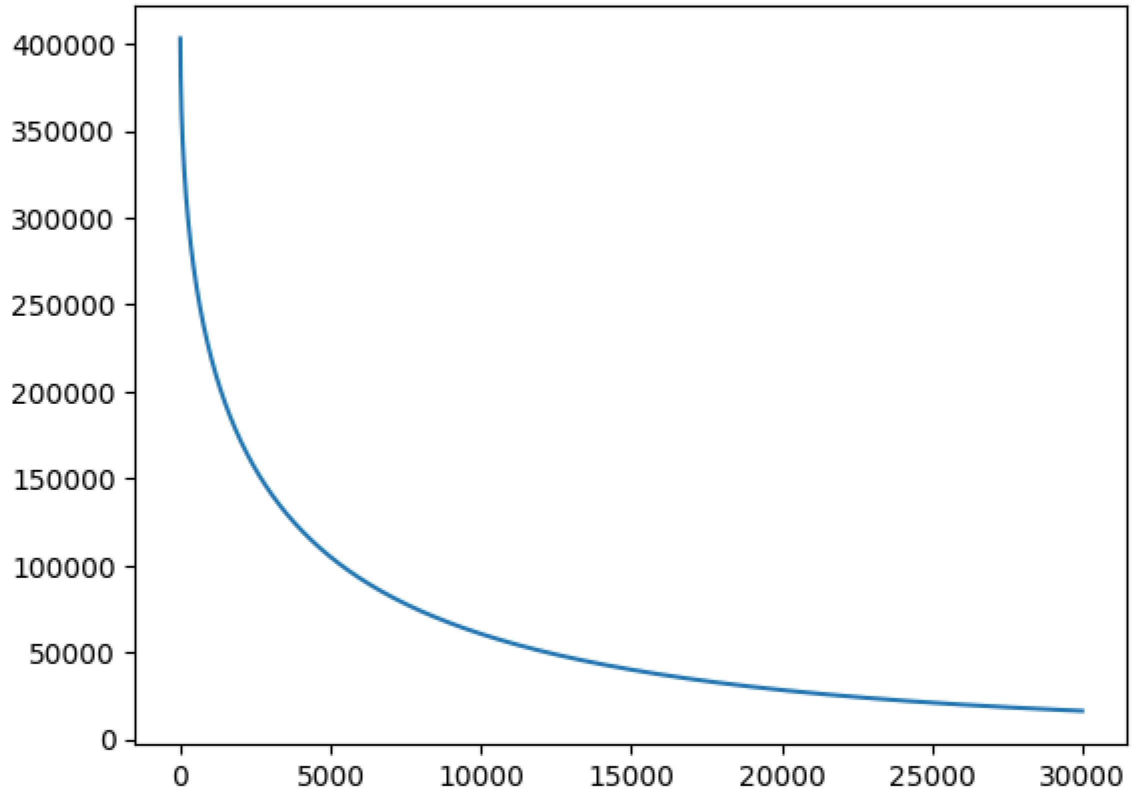
Esta função deve retornar a sequência de valores da função `J(df(x))` em cada um dos iterados `x`, isto é, a sequência das normas dos gradientes de X_k ao longo da trajetória do método. A função também deve retornar o último iterado.

Implemente a função com `b0=0.5` e `eta=1`.

In [9]:
```
#Escreva código aqui
def ad_grad_norm(J, df, X0, b0=0.5, eta=1.0, numb_iter=100):
    U, V = X0[0].copy(), X0[1].copy()
    grad_norms = list()
    grad_norm_sum = 0

    for i in range(numb_iter):
        dU, dV = df((U, V), Y)

        grad_norm = J(dU) + J(dV)
        grad_norms.append(float(grad_norm))

        grad_norm_sum += grad_norm ** 2

        step_size = eta / np.sqrt(b0 ** 2 + grad_norm_sum)

        U -= step_size * dU
        V -= step_size * dV

    return grad_norms, (U, V)
```

In [10]:
```
# Adagrad-Norm
f3 = ad_grad_norm(J, df, X0, b0=0.5, eta=1, numb_iter=N)
plt.plot(f3[0])
```

Out[10]: `[<matplotlib.lines.Line2D at 0x7f79c85273d0>]`



Exercício 5: Adam

Construa uma função `adam(J, df, x0, alpha, beta1, beta2, epsilon, numb_iter)` que toma como entrada as funções `J()`, `df`, o ponto inicial `x0`, parametros positivos `alpha`, `beta1`, `beta2`, `epsilon` e o número de iterações `numb_iter` e implementa o método Adam iniciando de `x0`, $m_0 = 0$, $v_0 = 0$ e $k = 0$: para cada $j$ézima coordenada:

$$
\begin{aligned}
m_{k+1}[j] &:= \beta_1 \cdot m_k[j] + (1-\beta_1)\cdot \nabla f(X_k)[j],\\
v_{k+1}[j] &:= \beta_2 \cdot v_k[j] + (1-\beta_2)\cdot (\nabla f(X_k)[j])^2,\\
\hat{m}_{k+1}[j] &:= \frac{1}{1-\beta_1^{k+1}} m_{k+1}[j],\\
\hat{v}_{k+1}[j] &:= \frac{1}{1-\beta_2^{k+1}} v_{k+1}[j],\\
X_{k+1}[j] &:= X_k[j] - \frac{\alpha}{\sqrt{\hat{v}_{k+1}[j]}+\epsilon}\hat{m}_{k+1}[j].
\end{aligned}
$$

Esta função deve retornar a sequência de valores da função `J(df(x))` em cada um dos iterados `x`, isto é, a sequência das normas dos gradientes de X_k ao longo da trajetória do método. A função também deve retornar o último iterado.

Implemente a função com `alpha=0.001`, `beta1=0.9`, `beta2=0.999`, `epsilon=10**(-8)`.

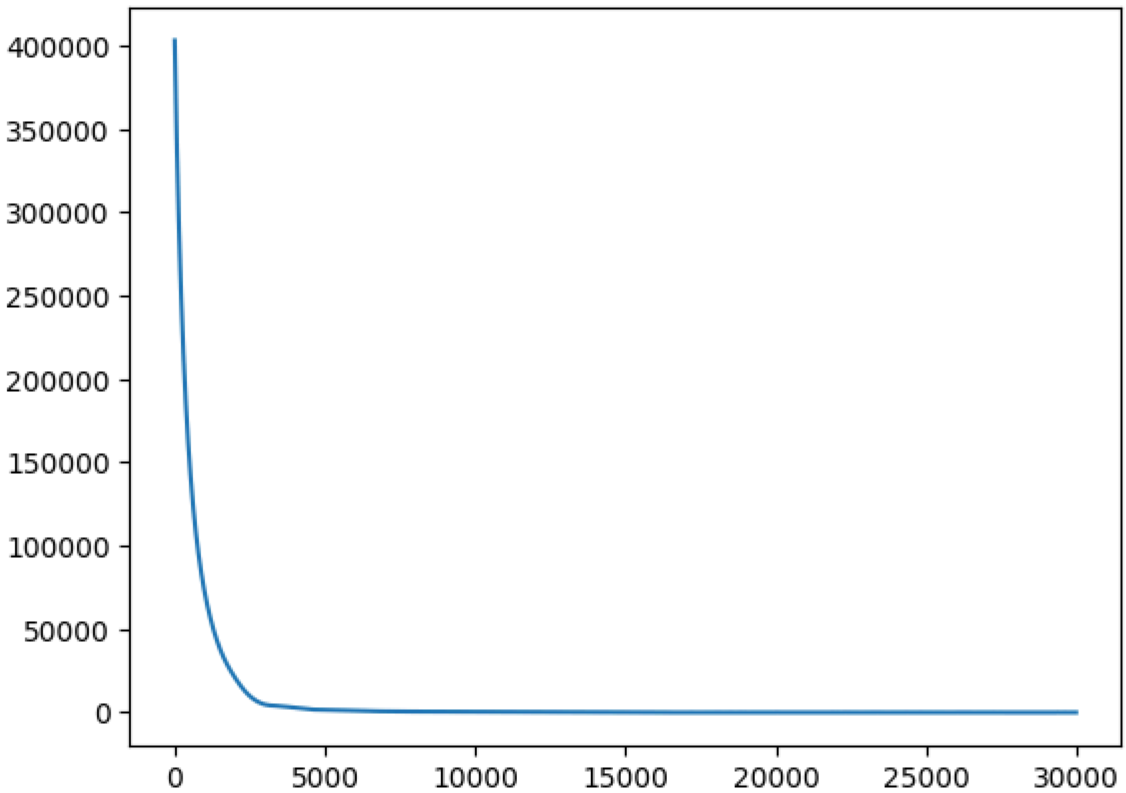
In [11]:

```python
#Escreva código aqui
def adam(J, df, X0, alpha=0.001, beta1=0.9, beta2=0.999, epsilon=1e-8, numb_iter=100):
    U, V = X0[0].copy(), X0[1].copy()
    m_U, m_V = np.zeros_like(U), np.zeros_like(V)
    v_U, v_V = np.zeros_like(U), np.zeros_like(V)
    grad_norms = list()

    for k in range(1, numb_iter + 1):
        dU, dV = df((U, V), Y)

        m_U = beta1 * m_U + (1 - beta1) * dU
        m_V = beta1 * m_V + (1 - beta1) * dV

        v_U = beta2 * v_U + (1 - beta2) * dU ** 2
        v_V = beta2 * v_V + (1 - beta2) * dV ** 2

        m_U_chapeu = m_U / (1 - beta1 ** k)
        m_V_chapeu = m_V / (1 - beta1 ** k)

        v_U_chapeu = v_U / (1 - beta2 ** k)
        v_V_chapeu = v_V / (1 - beta2 ** k)

        U -= alpha * m_U_chapeu / (np.sqrt(v_U_chapeu) + epsilon)
        V -= alpha * m_V_chapeu / (np.sqrt(v_V_chapeu) + epsilon)
```

```python
        grad_norm = J(dU) + J(dV)
        grad_norms.append(float(grad_norm))

    return grad_norms, (U, V)
```

In [12]:
```python
# Adam
f4 = adam(J, df, X0, alpha=0.001, beta1=0.9, beta2=0.999, epsilon=10**(-8), numb_iter=N)
plt.plot(f4[0])
```

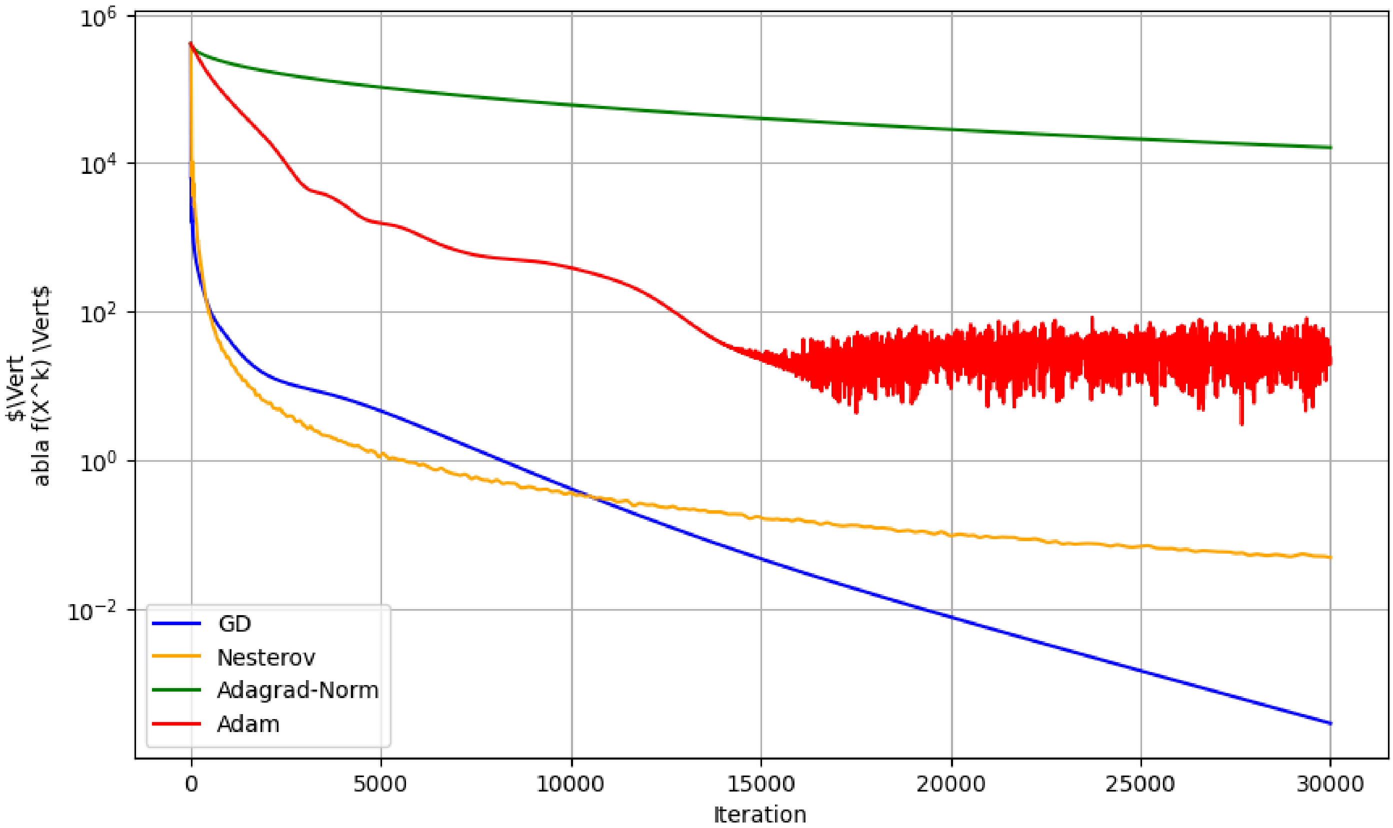
Out[12]:
```
[<matplotlib.lines.Line2D at 0x7f79cb312bd0>]
```



Exercício 6:

Implemente num mesmo gráfico os erros $\|\nabla f(X_k)\|$ de cada método em função no número de iterações.

In [13]:
```python
#Escreva código aqui
plt.figure(figsize=(10, 6))
plt.plot(f1[0], label='GD', color='blue')
plt.plot(f2[0], label='Nesterov', color='orange')
plt.plot(f3[0], label='Adagrad-Norm', color='green')
plt.plot(f4[0], label='Adam', color='red')

plt.yscale('log')
plt.xlabel('Iteration')
plt.ylabel("$\Vert \nabla f(X^k) \Vert$")
plt.legend()
plt.grid(True)
plt.show()
```



Exercício 7:

Experimente com os hyper-parâmetros de Adagrad-Norm e Adam para ver se eles podem chegar perto ou superar a performance de GD e Nesterov. Plote o gráfico como no Exercício 6.

In [15]:
```python
# Adam
import optuna
import cupy as np

# Função exemplo de perda (norma do gradiente)
def J(gradient):
    return np.linalg.norm(gradient)

# Função objetivo para Adagrad-Norm
def objective_adagrad(trial):
    # Hiperparâmetros a serem otimizados
    b0 = trial.suggest_float("b0", 0.1, 1.0, log=True)
    eta = trial.suggest_float("eta", 0.01, 1.0, log=True)
    numb_iter = 5000

    # Defina df (gradiente) e ponto inicial X0
    df = lambda X, Y: (2 * X[0], 2 * X[1])  # Exemplo de função de gradiente
    X0 = (np.array([5.0, 5.0]), np.array([5.0, 5.0]))
    Y = None  # Parâmetro adicional, se necessário

    # Execute Adagrad-Norm com os parâmetros testados
    grad_norms, final_X = ad_grad_norm(J, df, X0, b0=b0, eta=eta, numb_iter=numb_iter)

    # Retorne o valor final da norma do gradiente para o Optuna
    return grad_norms[-1]

# Função objetivo para Adam
def objective_adam(trial):
    # Hiperparâmetros a serem otimizados
    alpha = trial.suggest_float("alpha", 1e-5, 1e-2, log=True)
    beta1 = trial.suggest_float("beta1", 0.8, 0.99)
    beta2 = trial.suggest_float("beta2", 0.9, 0.9999)
    epsilon = trial.suggest_float("epsilon", 1e-8, 1e-6, log=True)
    numb_iter = 5000

    # Defina df (gradiente) e ponto inicial X0
    df = lambda X, Y: (2 * X[0], 2 * X[1])  # Exemplo de função de gradiente
    X0 = (np.array([5.0, 5.0]), np.array([5.0, 5.0]))
    Y = None  # Parâmetro adicional, se necessário

    # Execute Adam com os parâmetros testados
    grad_norms, final_X = adam(J, df, X0, alpha=alpha, beta1=beta1, beta2=beta2, epsilon=epsilon

    # Retorne o valor final da norma do gradiente para o Optuna
```

```
    return grad_norms[-1]

# Executar a otimização para Adagrad-Norm
study_adagrad = optuna.create_study(direction="minimize")
study_adagrad.optimize(objective_adagrad, n_trials=30)

# Executar a otimização para Adam
study_adam = optuna.create_study(direction="minimize")
study_adam.optimize(objective_adam, n_trials=30)

print("Melhores hiperparâmetros para Adagrad-Norm:", study_adagrad.best_params)
print("Melhores hiperparâmetros para Adam:", study_adam.best_params)
```

```
[I 2024-11-11 21:58:53,549] A new study created in memory with name: no-name-9bea94c8-2928-4ba6-a1c2-b0ed228b8e7e
[I 2024-11-11 21:58:56,117] Trial 0 finished with value: 12.965024364888086 and parameters: {'b0': 0.9990865887081177, 'eta': 0.06286484354195433}. Best is trial 0 with value: 12.965024364888086.
[I 2024-11-11 21:58:58,325] Trial 1 finished with value: 9.29680113057297e-25 and parameters: {'b0': 0.3805605093713223, 'eta': 0.8189586703788901}. Best is trial 1 with value: 9.29680113057297e-25.
[I 2024-11-11 21:59:00,698] Trial 2 finished with value: 1.629668017104445e-10 and parameters: {'b0': 0.244568541927596, 'eta': 0.5290983447362997}. Best is trial 1 with value: 9.29680113057297e-25.
[I 2024-11-11 21:59:03,223] Trial 3 finished with value: 0.7389116671991548 and parameters: {'b0': 0.896767160616109 6, 'eta': 0.18104469165170678}. Best is trial 1 with value: 9.29680113057297e-25.
[I 2024-11-11 21:59:05,422] Trial 4 finished with value: 0.14495931407910828 and parameters: {'b0': 0.5298960260871683, 'eta': 0.2238011546744606}. Best is trial 1 with value: 9.29680113057297e-25.
[I 2024-11-11 21:59:07,755] Trial 5 finished with value: 14.806133494561822 and parameters: {'b0': 0.5339330436703172, 'eta': 0.053988669456259915}. Best is trial 1 with value: 9.29680113057297e-25.
[I 2024-11-11 21:59:10,087] Trial 6 finished with value: 5.37998443366766 and parameters: {'b0': 0.6253484852241372, 'eta': 0.11022416875201203}. Best is trial 1 with value: 9.29680113057297e-25.
[I 2024-11-11 21:59:12,478] Trial 7 finished with value: 24.27728282191116 and parameters: {'b0': 0.21968402257398936, 'eta': 0.014696286342385534}. Best is trial 1 with value: 9.29680113057297e-25.
[I 2024-11-11 21:59:14,837] Trial 8 finished with value: 2.1378225523954888e-14 and parameters: {'b0': 0.10826482998700185, 'eta': 0.619642697799502}. Best is trial 1 with value: 9.29680113057297e-25.
[I 2024-11-11 21:59:17,092] Trial 9 finished with value: 20.952020717604015 and parameters: {'b0': 0.538202269677630 1, 'eta': 0.02759609404043204}. Best is trial 1 with value: 9.29680113057297e-25.
[I 2024-11-11 21:59:19,238] Trial 10 finished with value: 4.3925746961180303e-29 and parameters: {'b0': 0.3314707245592656, 'eta': 0.8909042290309462}. Best is trial 10 with value: 4.3925746961180303e-29.
[I 2024-11-11 21:59:21,403] Trial 11 finished with value: 3.0982226460936636e-35 and parameters: {'b0': 0.3150310981391883, 'eta': 0.9856900865169511}. Best is trial 11 with value: 3.0982226460936636e-35.
[I 2024-11-11 21:59:23,777] Trial 12 finished with value: 3.090468389753408e-05 and parameters: {'b0': 0.17085177106581503, 'eta': 0.37729153957253236}. Best is trial 11 with value: 3.0982226460936636e-35.
[I 2024-11-11 21:59:25,984] Trial 13 finished with value: 6.746064375755311e-36 and parameters: {'b0': 0.33999122847110275, 'eta': 0.9954518913247333}. Best is trial 13 with value: 6.746064375755311e-36.
[I 2024-11-11 21:59:28,183] Trial 14 finished with value: 0.0020131563627138627 and parameters: {'b0': 0.2630712521439227, 'eta': 0.3103659816515463}. Best is trial 13 with value: 6.746064375755311e-36.
[I 2024-11-11 21:59:30,339] Trial 15 finished with value: 1.5642477990484573e-33 and parameters: {'b0': 0.14947556417327085, 'eta': 0.9602056294727696}. Best is trial 13 with value: 6.746064375755311e-36.
[I 2024-11-11 21:59:32,496] Trial 16 finished with value: 0.00022979778844396038 and parameters: {'b0': 0.3713379155217437, 'eta': 0.34662290553329206}. Best is trial 13 with value: 6.746064375755311e-36.
[I 2024-11-11 21:59:34,925] Trial 17 finished with value: 3.15591137345299 and parameters: {'b0': 0.4269542199592423, 'eta': 0.13259488970260488}. Best is trial 13 with value: 6.746064375755311e-36.
[I 2024-11-11 21:59:37,178] Trial 18 finished with value: 7.916657714516456e-15 and parameters: {'b0': 0.19026041716454675, 'eta': 0.6290029187497674}. Best is trial 13 with value: 6.746064375755311e-36.
[I 2024-11-11 21:59:39,526] Trial 19 finished with value: 25.156530681958994 and parameters: {'b0': 0.29382421787151236, 'eta': 0.011400971320077889}. Best is trial 13 with value: 6.746064375755311e-36.
[I 2024-11-11 21:59:41,840] Trial 20 finished with value: 12.73477440796099 and parameters: {'b0': 0.7594485410405417, 'eta': 0.06401818843598875}. Best is trial 13 with value: 6.746064375755311e-36.
[I 2024-11-11 21:59:44,054] Trial 21 finished with value: 3.1408709802789535e-35 and parameters: {'b0': 0.1407581899310437, 'eta': 0.9855938959115744}. Best is trial 13 with value: 6.746064375755311e-36.
[I 2024-11-11 21:59:46,241] Trial 22 finished with value: 4.971697369098377e-09 and parameters: {'b0': 0.11308472009581612, 'eta': 0.49067295649321646}. Best is trial 13 with value: 6.746064375755311e-36.
[I 2024-11-11 21:59:49,259] Trial 23 finished with value: 1.5752613890144115e-32 and parameters: {'b0': 0.1344497358942469, 'eta': 0.944941096000944}. Best is trial 13 with value: 6.746064375755311e-36.
[I 2024-11-11 21:59:51,406] Trial 24 finished with value: 0.059534142723644254 and parameters: {'b0': 0.21813080747710514, 'eta': 0.24417593662907594}. Best is trial 13 with value: 6.746064375755311e-36.
[I 2024-11-11 21:59:54,194] Trial 25 finished with value: 5.689145260031012e-07 and parameters:
```

```
{'b0': 0.3029368010403351, 'eta': 0.43246797752688854}. Best is trial 13 with value: 6.7460643757
55311e-36.
[I 2024-11-11 21:59:56,366] Trial 26 finished with value: 4.0243646954073225e-15 and parameters:
{'b0': 0.4644464547754229, 'eta': 0.6353173582231669}. Best is trial 13 with value: 6.74606437575
5311e-36.
[I 2024-11-11 21:59:58,828] Trial 27 finished with value: 3.602498640342476e-19 and parameters:
{'b0': 0.13460029786015232, 'eta': 0.7173121338792655}. Best is trial 13 with value: 6.7460643757
55311e-36.
[I 2024-11-11 22:00:01,239] Trial 28 finished with value: 0.0021023267508871154 and parameters:
{'b0': 0.1686088472293914, 'eta': 0.309601837704892}. Best is trial 13 with value: 6.746064375755
311e-36.
[I 2024-11-11 22:00:03,556] Trial 29 finished with value: 10.92015627397105 and parameters: {'b
0': 0.6671246146102565, 'eta': 0.0735242700290691}. Best is trial 13 with value: 6.74606437575531
1e-36.
[I 2024-11-11 22:00:03,557] A new study created in memory with name: no-name-75067cc7-d6a1-4f14-a
36f-c8afccabd68d
[I 2024-11-11 22:00:08,712] Trial 0 finished with value: 27.04289533680407 and parameters: {'alph
a': 4.388075472208628e-05, 'beta1': 0.9828437404253445, 'beta2': 0.9252217592753216, 'epsilon':
1.504437148799621e-07}. Best is trial 0 with value: 27.04289533680407.
[I 2024-11-11 22:00:14,480] Trial 1 finished with value: 7.360003371472163e-05 and parameters:
{'alpha': 0.0011750331656050351, 'beta1': 0.9530444984844055, 'beta2': 0.9487083404353772, 'epsil
on': 4.57541014534091e-08}. Best is trial 1 with value: 7.360003371472163e-05.
[I 2024-11-11 22:00:19,798] Trial 2 finished with value: 8.95248848235117e-05 and parameters: {'a
lpha': 0.0012765757935183878, 'beta1': 0.8768653240532875, 'beta2': 0.9566279530222325, 'epsilo
n': 8.609623978137356e-07}. Best is trial 1 with value: 7.360003371472163e-05.
[I 2024-11-11 22:00:24,579] Trial 3 finished with value: 25.506079341102883 and parameters: {'alp
ha': 9.825739695975187e-05, 'beta1': 0.872704103176246, 'beta2': 0.9320067572699047, 'epsilon':
2.8920921860176536e-08}. Best is trial 1 with value: 7.360003371472163e-05.
[I 2024-11-11 22:00:29,145] Trial 4 finished with value: 20.503108888213376 and parameters: {'alp
ha': 0.0002752453299417087, 'beta1': 0.8761633495552111, 'beta2': 0.9223585548162035, 'epsilon':
1.8613113089478563e-07}. Best is trial 1 with value: 7.360003371472163e-05.
[I 2024-11-11 22:00:33,984] Trial 5 finished with value: 19.190746590535348 and parameters: {'alp
ha': 0.00032185987213696055, 'beta1': 0.8326410953686046, 'beta2': 0.943501716769433, 'epsilon':
1.600235983950922e-07}. Best is trial 1 with value: 7.360003371472163e-05.
[I 2024-11-11 22:00:38,437] Trial 6 finished with value: 27.582425271559224 and parameters: {'alp
ha': 2.4822076877844972e-05, 'beta1': 0.8561663303332501, 'beta2': 0.9689242599899143, 'epsilon':
8.172324771016469e-07}. Best is trial 1 with value: 7.360003371472163e-05.
[I 2024-11-11 22:03:15,990] Trial 7 finished with value: 26.17457772108318 and parameters: {'alph
a': 7.451033865094521e-05, 'beta1': 0.9895249506459491, 'beta2': 0.914541893296228, 'epsilon': 1.
0163920605236761e-07}. Best is trial 1 with value: 7.360003371472163e-05.
[I 2024-11-11 22:00:49,451] Trial 8 finished with value: 0.0006574624804212371 and parameters:
{'alpha': 0.00590573751452528, 'beta1': 0.9028343562926366, 'beta2': 0.9425969204637129, 'epsilo
n': 4.3643083018617485e-08}. Best is trial 1 with value: 7.360003371472163e-05.
[I 2024-11-11 22:00:54,548] Trial 9 finished with value: 9.070192790832174e-05 and parameters:
{'alpha': 0.0042760230651679955, 'beta1': 0.8370555419103491, 'beta2': 0.9724975885334548, 'epsil
on': 1.4586776462300734e-08}. Best is trial 1 with value: 7.360003371472163e-05.
[I 2024-11-11 22:00:59,007] Trial 10 finished with value: 5.764173014620367 and parameters: {'alp
ha': 0.0009638154830304697, 'beta1': 0.9324256092242462, 'beta2': 0.9986959633769248, 'epsilon':
4.143687605461224e-08}. Best is trial 1 with value: 7.360003371472163e-05.
[I 2024-11-11 22:01:04,009] Trial 11 finished with value: 0.00022927898182446182 and parameters:
{'alpha': 0.001618280858215221, 'beta1': 0.9348175035733166, 'beta2': 0.9642674576582101, 'epsilo
n': 9.726857758579453e-07}. Best is trial 1 with value: 7.360003371472163e-05.
[I 2024-11-11 22:01:09,079] Trial 12 finished with value: 0.0002498939421796355 and parameters:
{'alpha': 0.0012358769234403712, 'beta1': 0.8001008315815494, 'beta2': 0.9591442930414285, 'epsil
on': 3.667330878386026e-07}. Best is trial 1 with value: 7.360003371472163e-05.
[I 2024-11-11 22:01:14,126] Trial 13 finished with value: 16.40034860463824 and parameters: {'alp
ha': 0.0004201414904229003, 'beta1': 0.9191633406978651, 'beta2': 0.9010773500528392, 'epsilon':
4.0813607943249906e-07}. Best is trial 1 with value: 7.360003371472163e-05.
[I 2024-11-11 22:01:18,746] Trial 14 finished with value: 5.68513707524683e-05 and parameters:
{'alpha': 0.0023166082797773098, 'beta1': 0.9584705343003076, 'beta2': 0.9822140428888487, 'epsil
on': 6.778878837714148e-08}. Best is trial 14 with value: 5.68513707524683e-05.
[I 2024-11-11 22:01:23,487] Trial 15 finished with value: 3.836832346476007e-05 and parameters:
{'alpha': 0.0032098626888935225, 'beta1': 0.9599427932216873, 'beta2': 0.9879924569084911, 'epsil
on': 6.502468067048768e-08}. Best is trial 15 with value: 3.836832346476007e-05.
[I 2024-11-11 22:01:28,712] Trial 16 finished with value: 0.0006380049063566763 and parameters:
{'alpha': 0.00951187999490623, 'beta1': 0.9599543034032708, 'beta2': 0.9885659375062383, 'epsilo
n': 8.003685948808881e-08}. Best is trial 15 with value: 3.836832346476007e-05.
[I 2024-11-11 22:01:33,851] Trial 17 finished with value: 1.4788419163400032e-05 and parameters:
{'alpha': 0.0030554534107903658, 'beta1': 0.9629913320617801, 'beta2': 0.9818063308654452, 'epsil
on': 1.3875602933773957e-08}. Best is trial 17 with value: 1.4788419163400032e-05.
[I 2024-11-11 22:01:39,532] Trial 18 finished with value: 7.624053749870734e-21 and parameters:
{'alpha': 0.003561520697305306, 'beta1': 0.9699377405600118, 'beta2': 0.9952462429355582, 'epsilo
n': 1.03789960307954e-08}. Best is trial 18 with value: 7.624053749870734e-21.
[I 2024-11-11 22:01:43,992] Trial 19 finished with value: 4.056527263041254e-20 and parameters:
{'alpha': 0.009830477869515445, 'beta1': 0.9771571329334247, 'beta2': 0.999849282691034, 'epsilo
n': 1.126472414570448e-08}. Best is trial 18 with value: 7.624053749870734e-21.
[I 2024-11-11 22:01:48,744] Trial 20 finished with value: 3.8283606442000984e-19 and parameters:
{'alpha': 0.009970840658329008, 'beta1': 0.9792314137803209, 'beta2': 0.9996642815479237, 'epsilo
n': 1.025021957268027e-08}. Best is trial 18 with value: 7.624053749870734e-21.
```

```
[I 2024-11-11 22:01:54,196] Trial 21 finished with value: 6.802439367522098e-23 and parameters: {'alpha': 0.00953286309877464, 'beta1': 0.9751895738806128, 'beta2': 0.9973048759821315, 'epsilon': 1.0450060227505438e-08}. Best is trial 21 with value: 6.802439367522098e-23.
[I 2024-11-11 22:01:59,123] Trial 22 finished with value: 0.0005204413526472371 and parameters: {'alpha': 0.005824796567306812, 'beta1': 0.9395663913188501, 'beta2': 0.9918795658287536, 'epsilon': 2.1043640653687828e-08}. Best is trial 21 with value: 6.802439367522098e-23.
[I 2024-11-11 22:02:03,967] Trial 23 finished with value: 0.00010933767301859687 and parameters: {'alpha': 0.00985788761321355, 'beta1': 0.9751114501906613, 'beta2': 0.9781401755274095, 'epsilon': 2.0502562076531724e-08}. Best is trial 21 with value: 6.802439367522098e-23.
[I 2024-11-11 22:02:08,981] Trial 24 finished with value: 10.96943758521049 and parameters: {'alpha': 0.0006414278801507466, 'beta1': 0.9115551485977011, 'beta2': 0.9963583258201336, 'epsilon': 1.1358433511104384e-08}. Best is trial 21 with value: 6.802439367522098e-23.
[I 2024-11-11 22:02:13,721] Trial 25 finished with value: 27.98650706343533 and parameters: {'alpha': 1.0529696017737266e-05, 'beta1': 0.9745542411111396, 'beta2': 0.9755936035004382, 'epsilon': 2.042841421819846e-08}. Best is trial 21 with value: 6.802439367522098e-23.
[I 2024-11-11 22:02:18,460] Trial 26 finished with value: 0.0003676603033582775 and parameters: {'alpha': 0.00454540940496791, 'beta1': 0.9436425200910612, 'beta2': 0.9873559008019263, 'epsilon': 2.8531968891179676e-08}. Best is trial 21 with value: 6.802439367522098e-23.
[I 2024-11-11 22:02:23,951] Trial 27 finished with value: 2.598282726264337e-37 and parameters: {'alpha': 0.002213165890881288, 'beta1': 0.9241423503890107, 'beta2': 0.9934642439790442, 'epsilon': 1.6129459988505686e-08}. Best is trial 27 with value: 2.598282726264337e-37.
[I 2024-11-11 22:02:29,674] Trial 28 finished with value: 4.7499720246118595e-07 and parameters: {'alpha': 0.0020832433099577806, 'beta1': 0.9229019740760813, 'beta2': 0.991436978067277, 'epsilon': 1.644852738272956e-08}. Best is trial 27 with value: 2.598282726264337e-37.
[I 2024-11-11 22:02:35,514] Trial 29 finished with value: 10.481270497923564 and parameters: {'alpha': 0.0006356150017566652, 'beta1': 0.9002566912562378, 'beta2': 0.9830412667475061, 'epsilon': 2.866491433097435e-08}. Best is trial 27 with value: 2.598282726264337e-37.
Melhores hiperparâmetros para Adagrad-Norm: {'b0': 0.33999122847110275, 'eta': 0.9954518913247333}
Melhores hiperparâmetros para Adam: {'alpha': 0.002213165890881288, 'beta1': 0.9241423503890107, 'beta2': 0.9934642439790442, 'epsilon': 1.6129459988505686e-08}
```

In [16]:

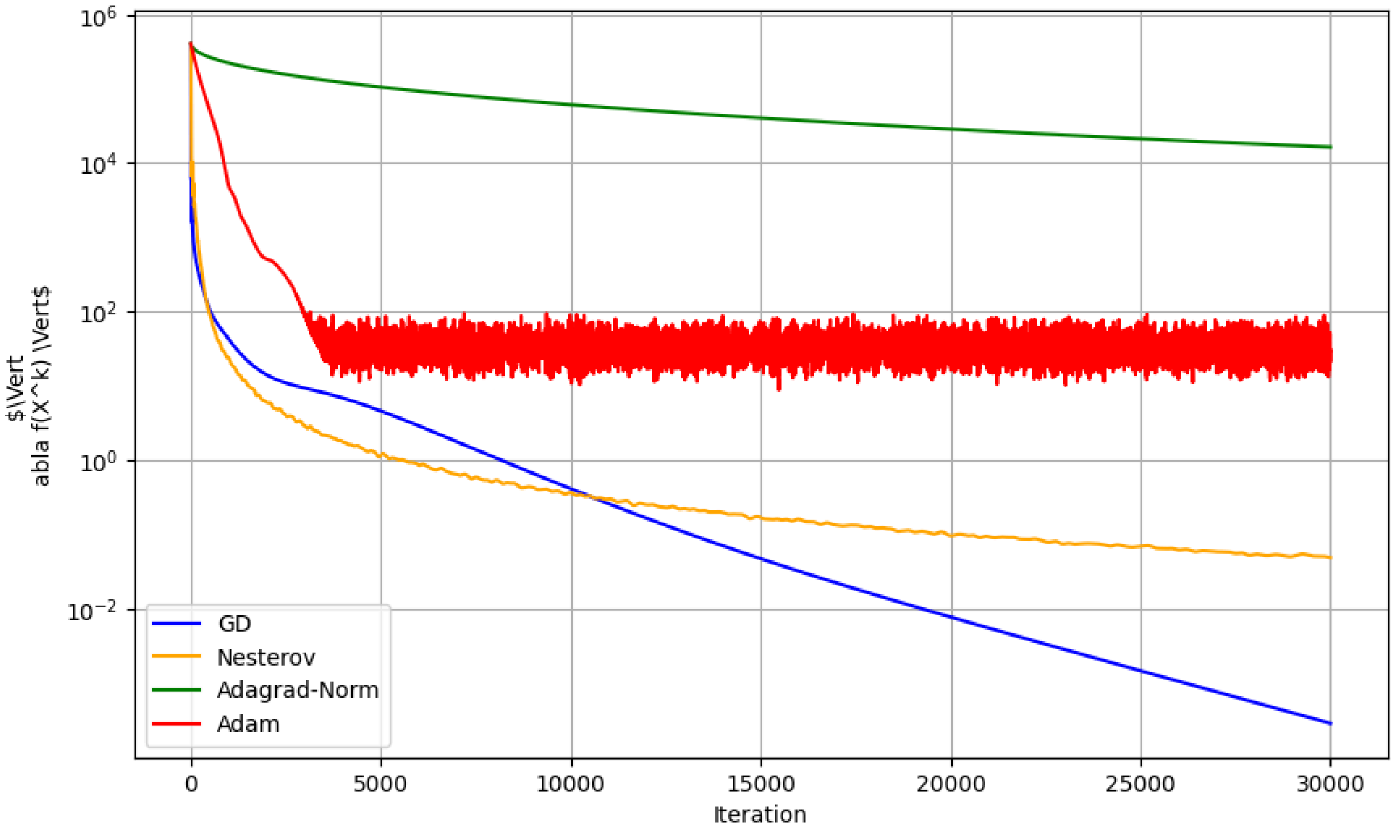
```python
import json

best_params_adagrad = study_adagrad.best_params
best_params_adam = study_adam.best_params

params = {
    "Adagrad-Norm": best_params_adagrad,
    "Adam": best_params_adam
}

with open("best_hyperparameters.json", "w") as f:
    json.dump(params, f, indent=4)
```

In [20]:

```python
f3_new = ad_grad_norm(J, df, X0, b0=params["Adagrad-Norm"]["b0"], eta=params["Adagrad-Norm"]["eta
f4_new = adam(J, df, X0, alpha=params["Adam"]["alpha"], beta1=params["Adam"]["beta1"], beta2=para
```

In [21]:

```python
plt.figure(figsize=(10, 6))
plt.plot(f1[0], label='GD', color='blue')
plt.plot(f2[0], label='Nesterov', color='orange')
plt.plot(f3_new[0], label='Adagrad-Norm', color='green')
plt.plot(f4_new[0], label='Adam', color='red')

plt.yscale('log')
plt.xlabel('Iteration')
plt.ylabel("$\Vert \nabla f(X^k) \Vert$")
plt.legend()
plt.grid(True)
plt.show()
```



10^6
10^4
10^2
10^0
10^{-2}
$\Vert
abla f(X^k) \Vert$
0
5000
10000
15000
20000
25000
30000
Iteration
GD
Nesterov
Adagrad-Norm
Adam