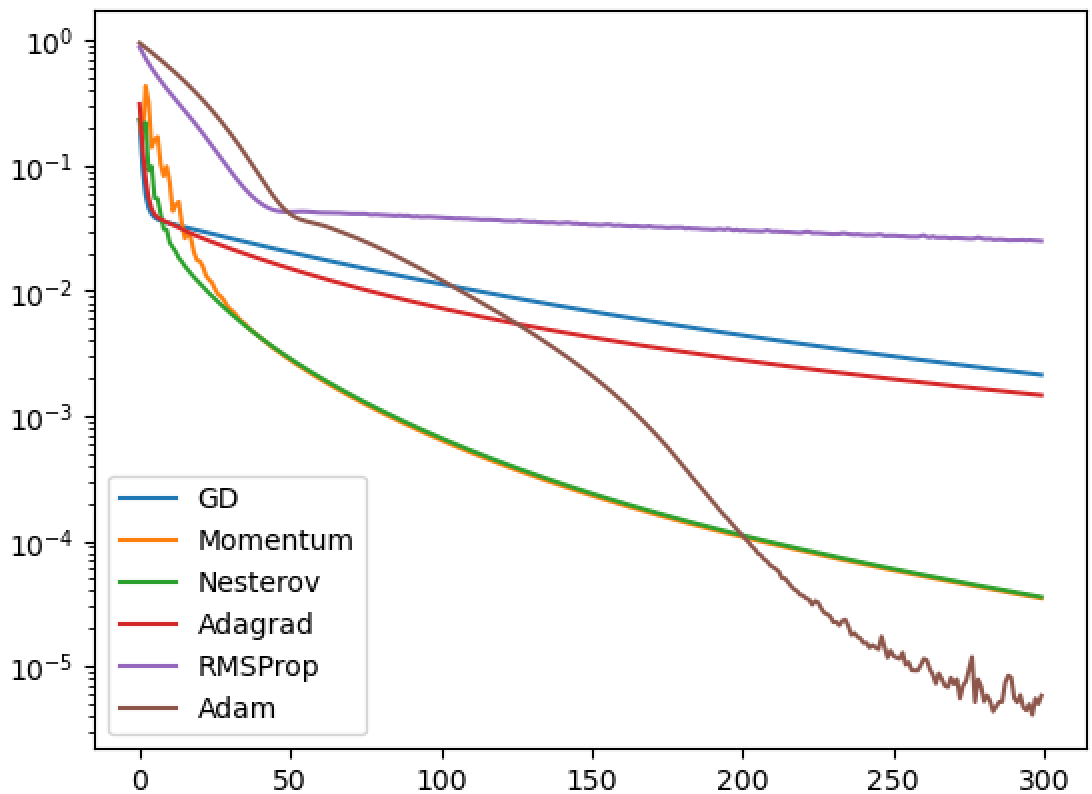
Problem A

In [66]:
```
import numpy as np
import numpy.linalg as npl
import matplotlib.pyplot as plt

np.random.seed(0)

d=100             # Number of features
n=200             # Sample size
IT=300            # Number of iterations

kappa=100         # Condition number
is_diag_cov=False # Whether features are aligned with the eigenspectrum

# Create input features based on kappa and is_diag_cov
COV=np.concatenate([np.sqrt(kappa)*np.ones(int(d/4)),np.ones(int(3*d/4))])
X0=np.random.randn(n,d)*COV
U=npl.svd(np.random.randn(d,d))[0]
X=X0 if is_diag_cov else X0.dot(U)

# Assign labels based on planted weights b
b=np.ones(d) # Planted all ones weights
y=X.dot(b)

### Gradient-based algorithms ###

### Algo 1: Gradient Descent ###
w=np.zeros(d)
errGD=np.zeros(IT)
eta=1/npl.svd(X)[1][0]**2
for it in range(IT):
    w+=eta*X.T.dot(y-X.dot(w))
    errGD[it]=npl.norm(y-X.dot(w))**2/npl.norm(y)**2
plt.semilogy(errGD)


### Algo 2: Momentum ###
w=np.zeros(d)
vel=np.zeros(d)
alpha=0.8
errMOM=np.zeros(IT)
eta=1/npl.svd(X)[1][0]**2
for it in range(IT):
    vel=alpha*vel+eta*X.T.dot(y-X.dot(w))
    w+=vel
    errMOM[it]=npl.norm(y-X.dot(w))**2/npl.norm(y)**2
plt.semilogy(errMOM)


### Algo 3: Nesterov ###
w=np.zeros(d)
vel=np.zeros(d)
alpha=0.8
errNest=np.zeros(IT)
#### Complete the algo below ###
for it in range(IT):
    vel = alpha * vel + eta * X.T.dot(y - X.dot(w))
    w_nesterov = w + vel
```

```
    vel = alpha * vel + eta * X.T.dot(y - X.dot(w_nesterov))
    w = w_nesterov
    errNest[it] = npl.norm(y - X.dot(w)) ** 2 / npl.norm(y) ** 2
plt.semilogy(errNest)


### Algo 4: Adagrad ###
w=np.zeros(d)
r=np.zeros(d)
delta=1
errAdaGrad=np.zeros(IT)
eta=1e4/npl.svd(X)[1][0]**2
#### Complete the algo below ###
for it in range(IT):
    gradient = X.T.dot(y - X.dot(w))
    r += gradient ** 2
    w += (eta / (np.sqrt(r + delta))) * gradient
    errAdaGrad[it] = npl.norm(y - X.dot(w)) ** 2 / npl.norm(y) ** 2
plt.semilogy(errAdaGrad)


### Algo 5: RMSProp with Nesterov ###
w=np.zeros(d)
vel=np.zeros(d)
r=np.zeros(d)
rho=0.8
alpha=0.8
errRMSProp=np.zeros(IT)
eta=5e2/npl.svd(X)[1][0]**2
#### Complete the algo below ###
eps = 1e-8
for it in range(IT):
    vel = alpha * vel + eta * X.T.dot(y - X.dot(w))
    w_nesterov = w + alpha * vel
    gradient = X.T.dot(y - X.dot(w_nesterov))
    r = rho * r + (1 - rho) * gradient ** 2
    w -= (eta / (np.sqrt(r) + eps)) * gradient
    errRMSProp[it] = npl.norm(y - X.dot(w)) ** 2 / npl.norm(y) ** 2
plt.semilogy(errRMSProp)


### Algo 6: Adam ###
w=np.zeros(d)
vel=np.zeros(d)
r=np.zeros(d)
alpha=0.8
rho=0.8
delta=1
errAdam=np.zeros(IT)
eta=4e2/npl.svd(X)[1][0]**2
#### Complete the algo below ###
for it in range(IT):
    gradient = X.T.dot(y - X.dot(w))
    vel = alpha * vel + (1 - alpha) * gradient
    r = rho * r + (1 - rho) * gradient ** 2
    vel_hat = vel / (1 - alpha ** (it+1))
    r_hat = r / (1 - rho ** (it+1))
```

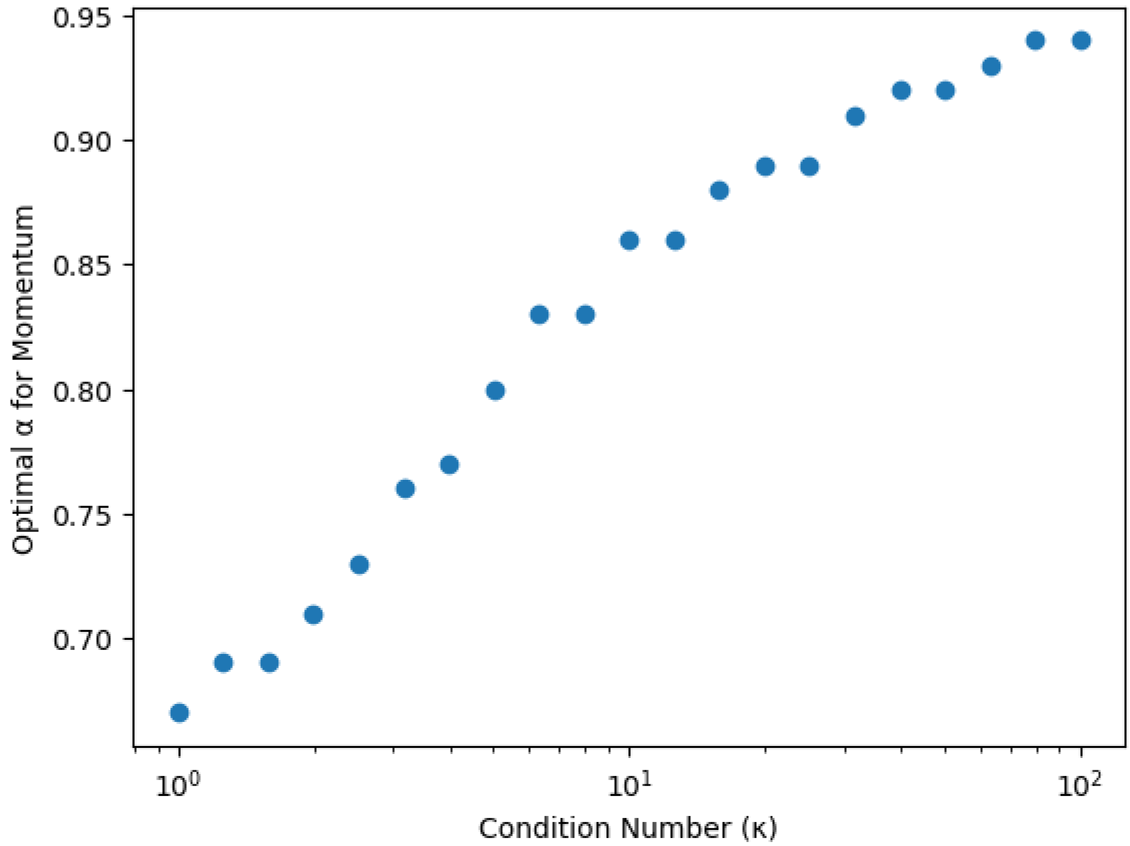
```
    w += (eta / (np.sqrt(r_hat) + delta)) * vel_hat
    errAdam[it] = npl.norm(y - X.dot(w)) ** 2 / npl.norm(y) ** 2
plt.semilogy(errAdam)

plt.legend(['GD','Momentum','Nesterov','Adagrad','RMSProp','Adam'])
```

Out[66]: <matplotlib.legend.Legend at 0x29e645bd0d0>



Problem C

In [80]:
```
optimal_alphas = []
is_diag_cov=False
for kappa in np.logspace(0.0, 2.0, num=21, base=10.0):
    COV = np.concatenate([np.sqrt(kappa) * np.ones(int(d / 4)), np.ones(int(3 
    X0=np.random.randn(n,d)*COV
    U=npl.svd(np.random.randn(d,d))[0]
    X=X0 if is_diag_cov else X0.dot(U)

    b=np.ones(d) # Planted all ones weights
    y=X.dot(b)
    valuemin = 999
    for alpha in np.arange(0.5, 1.1, 0.01):
        w=np.zeros(d)
        vel=np.zeros(d)
        errMOM=np.zeros(IT)
        eta=1/npl.svd(X)[1][0]**2
        for it in range(IT):
            vel=alpha*vel+eta*X.T.dot(y-X.dot(w))
            w+=vel
            errMOM[it]=npl.norm(y-X.dot(w))**2/npl.norm(y)**2
    # find the minimum index
        value = np.min(errMOM)
        if value < valuemin:
            optimal_alpha = alpha
            valuemin = value
    optimal_alphas.append(optimal_alpha)
plt.scatter(np.logspace(0.0, 2.0, num=21, base=10.0), optimal_alphas, marker='
plt.xscale('log')
plt.xlabel('Condition Number (κ)')
plt.ylabel('Optimal α for Momentum')
plt.show()
```

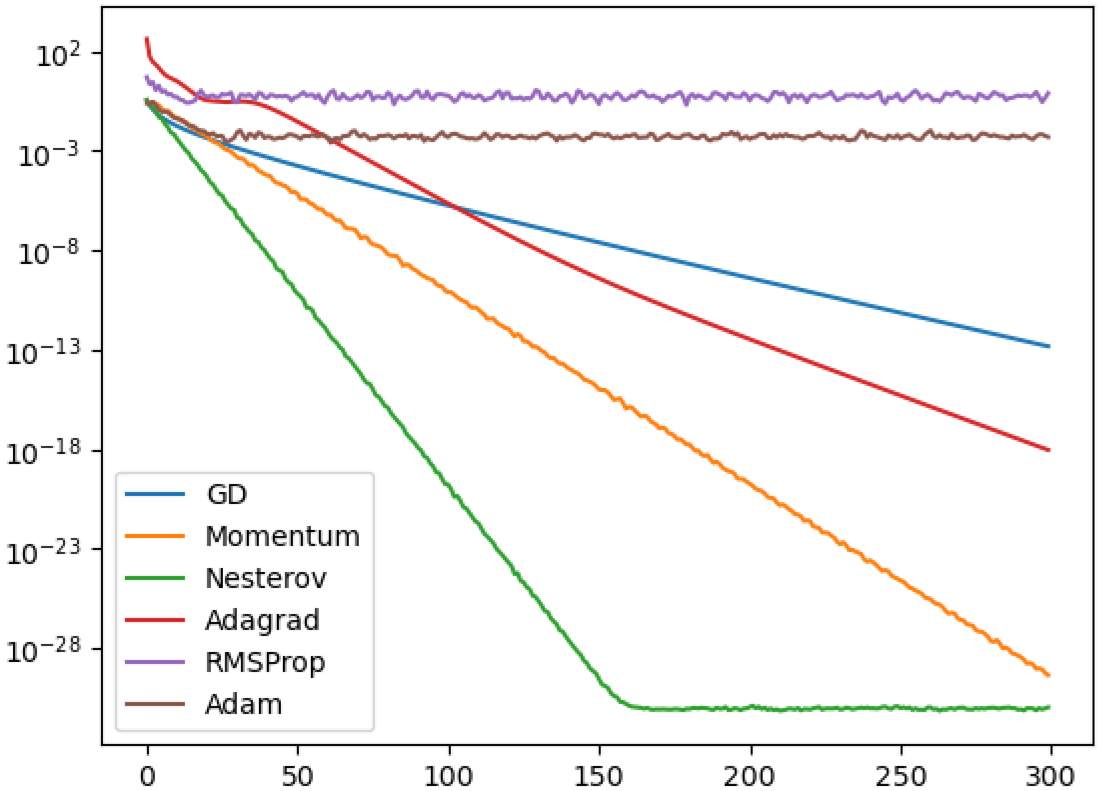

Optimal α for Momentum
Condition Number (κ)
0.95
0.90
0.85
0.80
0.75
0.70
10^0
10^1
10^2

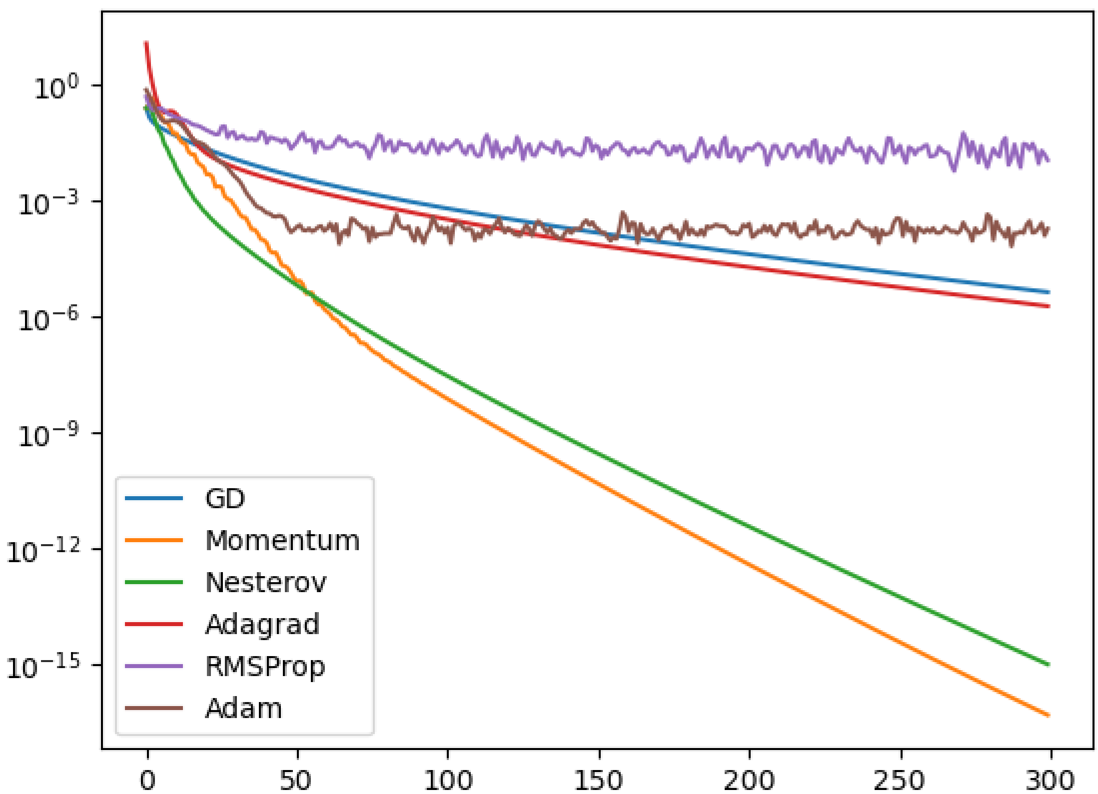
Problem d

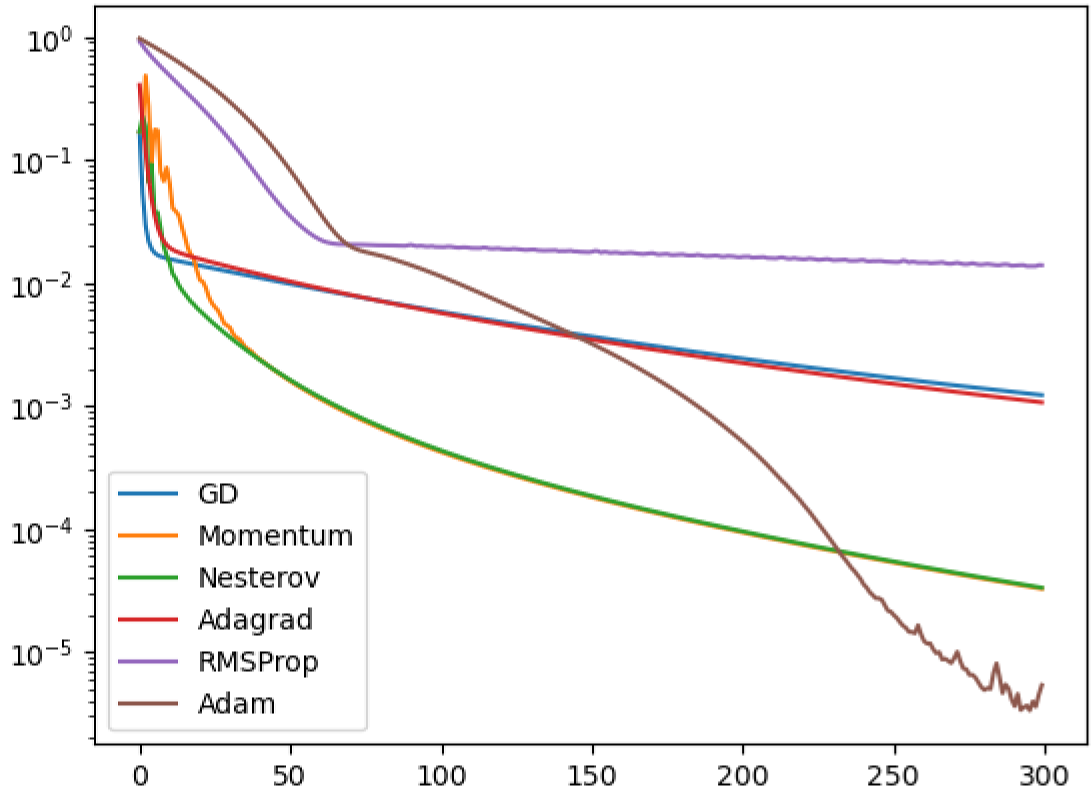
In [81]:
```
d=100             # Number of features
n=200             # Sample size
IT=300            # Number of iterations

kappa=100         # Condition number
is_diag_cov=False # Whether features are aligned with the eigenspectrum
for i, kappa in enumerate([1, 10, 100]):

    # Create input features based on kappa and is_diag_cov
    COV=np.concatenate([np.sqrt(kappa)*np.ones(int(d/4)),np.ones(int(3*d/4))])
    X0=np.random.randn(n,d)*COV
    U=npl.svd(np.random.randn(d,d))[0]
    X=X0 if is_diag_cov else X0.dot(U)

    # Assign labels based on planted weights b
    b=np.ones(d) # Planted all ones weights
    y=X.dot(b)
#     axes[i].set_title(f'Condition Number (κ) = {kappa}')
#     axes[i].set_xlabel('Iterations')
#     axes[i].set_ylabel('Error')
    ### Gradient-based algorithms ###

    ### Algo 1: Gradient Descent ###
    w=np.zeros(d)
    errGD=np.zeros(IT)
    eta=1/npl.svd(X)[1][0]**2
    for it in range(IT):
        w+=eta*X.T.dot(y-X.dot(w))
        errGD[it]=npl.norm(y-X.dot(w))**2/npl.norm(y)**2
    plt.semilogy(errGD)


    ### Algo 2: Momentum ###
    w=np.zeros(d)
    vel=np.zeros(d)
    alpha=0.8
    errMOM=np.zeros(IT)
    eta=1/npl.svd(X)[1][0]**2
    for it in range(IT):
        vel=alpha*vel+eta*X.T.dot(y-X.dot(w))
        w+=vel
        errMOM[it]=npl.norm(y-X.dot(w))**2/npl.norm(y)**2
    plt.semilogy(errMOM)


    ### Algo 3: Nesterov ###
    w=np.zeros(d)
    vel=np.zeros(d)
    alpha=0.8
    errNest=np.zeros(IT)
    #### Complete the algo below ###
    for it in range(IT):
        vel = alpha * vel + eta * X.T.dot(y - X.dot(w))
        w_nesterov = w + vel
        vel = alpha * vel + eta * X.T.dot(y - X.dot(w_nesterov))
        w = w_nesterov
        errNest[it] = npl.norm(y - X.dot(w)) ** 2 / npl.norm(y) ** 2
```

```
plt.semilogy(errNest)


### Algo 4: Adagrad ###
w=np.zeros(d)
r=np.zeros(d)
delta=1
errAdaGrad=np.zeros(IT)
eta=1e4/npl.svd(X)[1][0]**2
#### Complete the algo below ###
for it in range(IT):
    gradient = X.T.dot(y - X.dot(w))
    r += gradient ** 2
    w += (eta / (np.sqrt(r + delta))) * gradient
    errAdaGrad[it] = npl.norm(y - X.dot(w)) ** 2 / npl.norm(y) ** 2
plt.semilogy(errAdaGrad)


### Algo 5: RMSProp with Nesterov ###
w=np.zeros(d)
vel=np.zeros(d)
r=np.zeros(d)
rho=0.8
alpha=0.8
errRMSProp=np.zeros(IT)
eta=5e2/npl.svd(X)[1][0]**2
#### Complete the algo below ###
eps = 1e-8
for it in range(IT):
    vel = alpha * vel + eta * X.T.dot(y - X.dot(w))
    w_nesterov = w + alpha * vel
    gradient = X.T.dot(y - X.dot(w_nesterov))
    r = rho * r + (1 - rho) * gradient ** 2
    w -= (eta / (np.sqrt(r) + eps)) * gradient
    errRMSProp[it] = npl.norm(y - X.dot(w)) ** 2 / npl.norm(y) ** 2
plt.semilogy(errRMSProp)


### Algo 6: Adam ###
w=np.zeros(d)
vel=np.zeros(d)
r=np.zeros(d)
alpha=0.8
rho=0.8
delta=1
errAdam=np.zeros(IT)
eta=4e2/npl.svd(X)[1][0]**2
#### Complete the algo below ###
for it in range(IT):
    gradient = X.T.dot(y - X.dot(w))
    vel = alpha * vel + (1 - alpha) * gradient
    r = rho * r + (1 - rho) * gradient ** 2
    vel_hat = vel / (1 - alpha ** (it+1))
    r_hat = r / (1 - rho ** (it+1))
    w += (eta / (np.sqrt(r_hat) + delta)) * vel_hat
    errAdam[it] = npl.norm(y - X.dot(w)) ** 2 / npl.norm(y) ** 2
plt.semilogy(errAdam)
```

```
plt.legend(['GD', 'Momentum', 'Nesterov', 'Adagrad', 'RMSProp', 'Adam'])
plt.show()
```





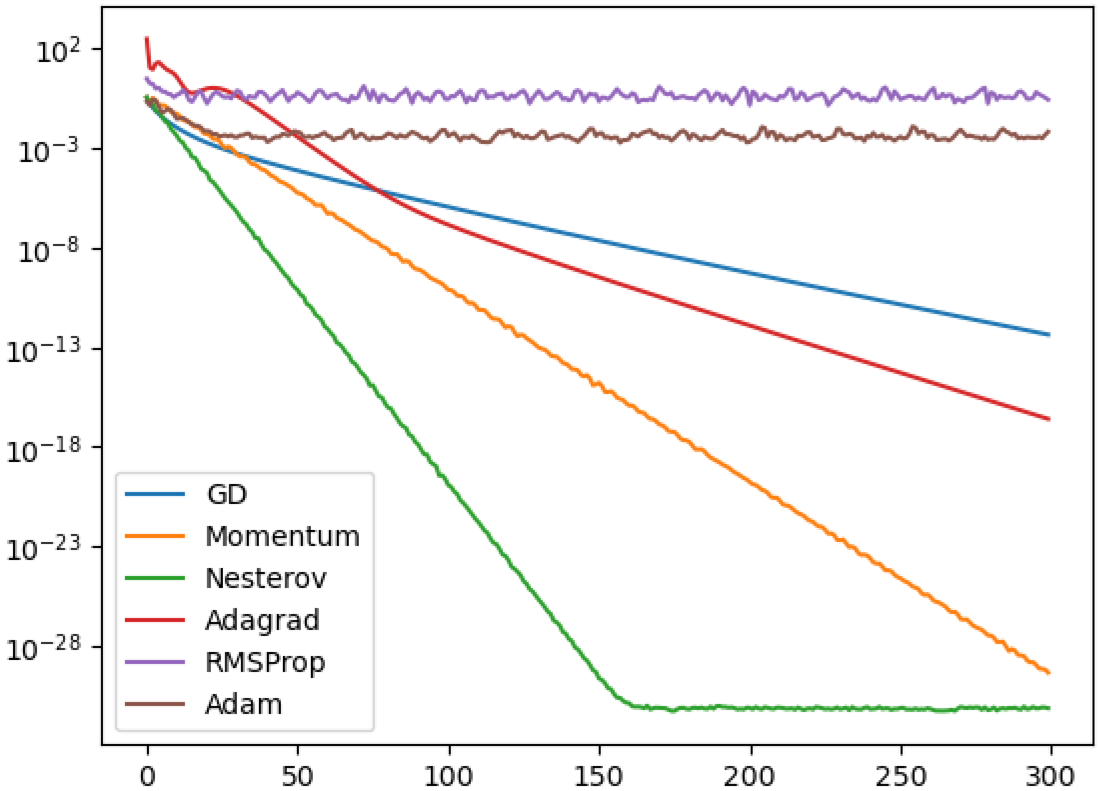

10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
0
50
100
150
200
250
300
GD
Momentum
Nesterov
Adagrad
RMSProp
Adam

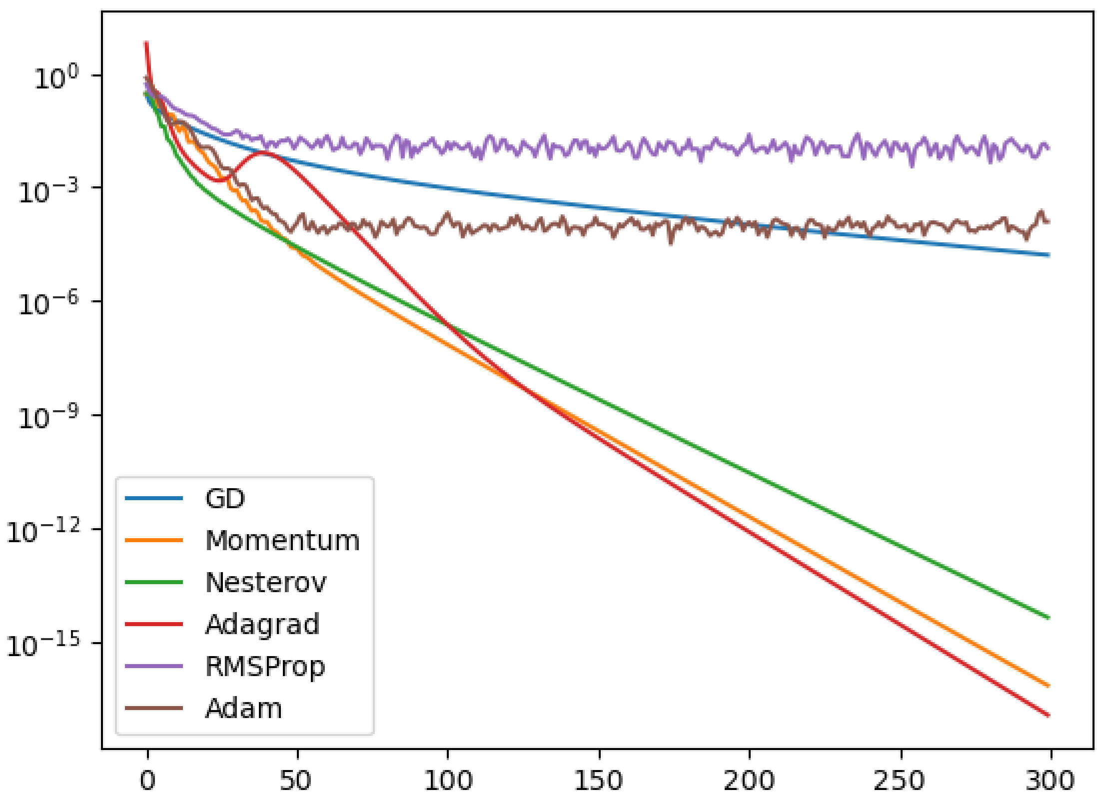
Problem e

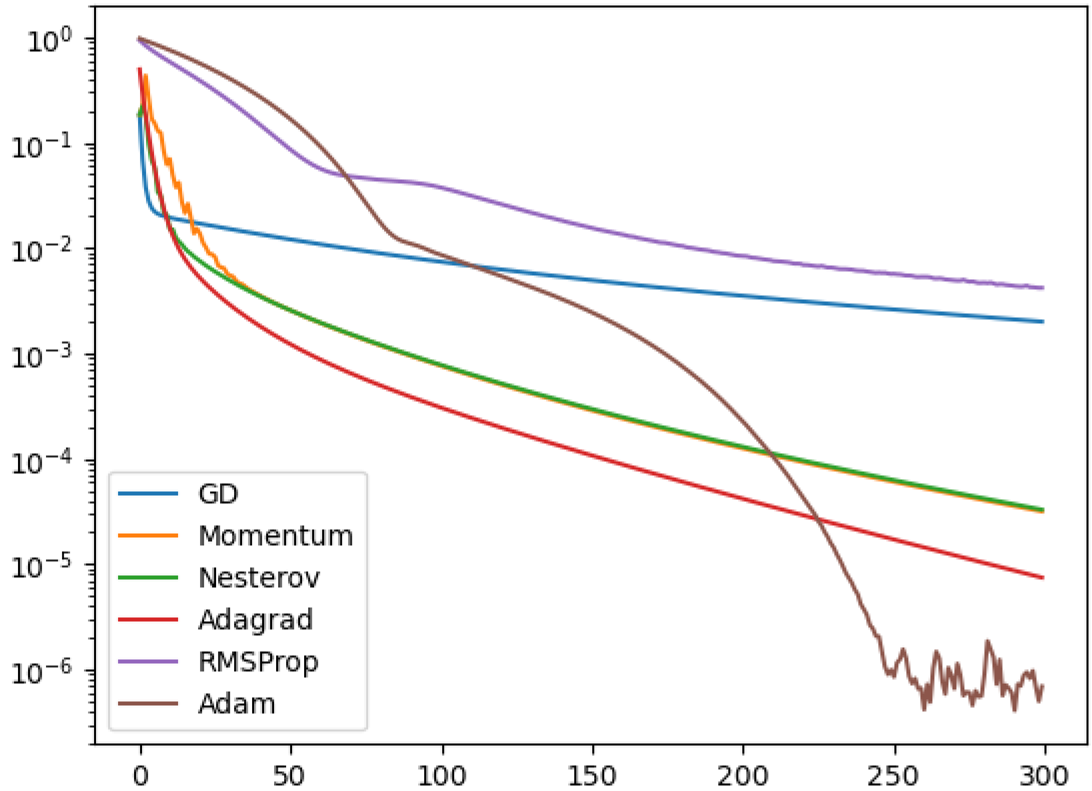
In [82]:
```
d=100             # Number of features
n=200             # Sample size
IT=300            # Number of iterations

kappa=100         # Condition number
is_diag_cov=True # Whether features are aligned with the eigenspectrum
for i, kappa in enumerate([1, 10, 100]):

    # Create input features based on kappa and is_diag_cov
    COV=np.concatenate([np.sqrt(kappa)*np.ones(int(d/4)),np.ones(int(3*d/4))])
    X0=np.random.randn(n,d)*COV
    U=npl.svd(np.random.randn(d,d))[0]
    X=X0 if is_diag_cov else X0.dot(U)

    # Assign labels based on planted weights b
    b=np.ones(d) # Planted all ones weights
    y=X.dot(b)
#     axes[i].set_title(f'Condition Number (κ) = {kappa}')
#     axes[i].set_xlabel('Iterations')
#     axes[i].set_ylabel('Error')
    ### Gradient-based algorithms ###

    ### Algo 1: Gradient Descent ###
    w=np.zeros(d)
    errGD=np.zeros(IT)
    eta=1/npl.svd(X)[1][0]**2
    for it in range(IT):
        w+=eta*X.T.dot(y-X.dot(w))
        errGD[it]=npl.norm(y-X.dot(w))**2/npl.norm(y)**2
    plt.semilogy(errGD)


    ### Algo 2: Momentum ###
    w=np.zeros(d)
    vel=np.zeros(d)
    alpha=0.8
    errMOM=np.zeros(IT)
    eta=1/npl.svd(X)[1][0]**2
    for it in range(IT):
        vel=alpha*vel+eta*X.T.dot(y-X.dot(w))
        w+=vel
        errMOM[it]=npl.norm(y-X.dot(w))**2/npl.norm(y)**2
    plt.semilogy(errMOM)


    ### Algo 3: Nesterov ###
    w=np.zeros(d)
    vel=np.zeros(d)
    alpha=0.8
    errNest=np.zeros(IT)
    #### Complete the algo below ###
    for it in range(IT):
        vel = alpha * vel + eta * X.T.dot(y - X.dot(w))
        w_nesterov = w + vel
        vel = alpha * vel + eta * X.T.dot(y - X.dot(w_nesterov))
        w = w_nesterov
        errNest[it] = npl.norm(y - X.dot(w)) ** 2 / npl.norm(y) ** 2
```

```
plt.semilogy(errNest)


### Algo 4: Adagrad ###
w=np.zeros(d)
r=np.zeros(d)
delta=1
errAdaGrad=np.zeros(IT)
eta=1e4/npl.svd(X)[1][0]**2
#### Complete the algo below ###
for it in range(IT):
    gradient = X.T.dot(y - X.dot(w))
    r += gradient ** 2
    w += (eta / (np.sqrt(r + delta))) * gradient
    errAdaGrad[it] = npl.norm(y - X.dot(w)) ** 2 / npl.norm(y) ** 2
plt.semilogy(errAdaGrad)


### Algo 5: RMSProp with Nesterov ###
w=np.zeros(d)
vel=np.zeros(d)
r=np.zeros(d)
rho=0.8
alpha=0.8
errRMSProp=np.zeros(IT)
eta=5e2/npl.svd(X)[1][0]**2
#### Complete the algo below ###
eps = 1e-8
for it in range(IT):
    vel = alpha * vel + eta * X.T.dot(y - X.dot(w))
    w_nesterov = w + alpha * vel
    gradient = X.T.dot(y - X.dot(w_nesterov))
    r = rho * r + (1 - rho) * gradient ** 2
    w -= (eta / (np.sqrt(r) + eps)) * gradient
    errRMSProp[it] = npl.norm(y - X.dot(w)) ** 2 / npl.norm(y) ** 2
plt.semilogy(errRMSProp)


### Algo 6: Adam ###
w=np.zeros(d)
vel=np.zeros(d)
r=np.zeros(d)
alpha=0.8
rho=0.8
delta=1
errAdam=np.zeros(IT)
eta=4e2/npl.svd(X)[1][0]**2
#### Complete the algo below ###
for it in range(IT):
    gradient = X.T.dot(y - X.dot(w))
    vel = alpha * vel + (1 - alpha) * gradient
    r = rho * r + (1 - rho) * gradient ** 2
    vel_hat = vel / (1 - alpha ** (it+1))
    r_hat = r / (1 - rho ** (it+1))
    w += (eta / (np.sqrt(r_hat) + delta)) * vel_hat
    errAdam[it] = npl.norm(y - X.dot(w)) ** 2 / npl.norm(y) ** 2
plt.semilogy(errAdam)
```

```python
    plt.legend(['GD', 'Momentum', 'Nesterov', 'Adagrad', 'RMSProp', 'Adam'])
    plt.show()
```







is_diag_cov did affect the results.

If is_diag_cov is aligned (no feature correlation), algorithms likely handle the optimization task more efficiently due to a simpler covariance structure.

In the presence of feature correlation (when is_diag_cov is False), the optimization task becomes more challenging. The complex covariance structure requires algorithms to adapt, potentially impacting stability and convergence speed.

Problem f

In [98]:

```
d=100             # Number of features
n=200             # Sample size
IT=300            # Number of iterations

kappa=100         # Condition number
is_diag_cov=False # Whether features are aligned with the eigenspectrum


# Create input features based on kappa and is_diag_cov
COV=np.concatenate([np.sqrt(kappa)*np.ones(int(d/4)),np.ones(int(3*d/4))])
X0=np.random.randn(n,d)*COV
U=npl.svd(np.random.randn(d,d))[0]
X=X0 if is_diag_cov else X0.dot(U)

# Assign labels based on planted weights b
b=np.ones(d) # Planted all ones weights
y=X.dot(b)

### Algo 6: Adam ###
w=np.zeros(d)
vel=np.zeros(d)
r=np.zeros(d)
alpha=0
rho=0.8
delta=1
errAdam=np.zeros(IT)
eta=4e2/npl.svd(X)[1][0]**2
#### Complete the algo below ###
valuemin = 999
optimal_rhos = []
for rho in np.arange(0.1, 1.01, 0.01):
    for it in range(IT):
        gradient = X.T.dot(y - X.dot(w))
        vel = alpha * vel + (1 - alpha) * gradient
        r = rho * r + (1 - rho) * gradient ** 2
        vel_hat = vel / (1 - alpha ** (it+1))
        r_hat = r / (1 - rho ** (it+1))
        w += (eta / (np.sqrt(r_hat) + delta)) * vel_hat
        errAdam[it] = npl.norm(y - X.dot(w)) ** 2 / npl.norm(y) ** 2

    value = np.min(errAdam)
    if value < valuemin:
        optimal_rho = rho
        valuemin = value
print(f'optimal rho: {optimal_rho}')
#plt.scatter(np.arange(0.1, 1.01, 0.01), optimal_rhos, marker='o')
```

```
optimal rho: 0.9499999999999995
```

In [99]:

```python
d=100              # Number of features
n=200              # Sample size
IT=300             # Number of iterations

kappa=100          # Condition number
is_diag_cov=False # Whether features are aligned with the eigenspectrum


# Create input features based on kappa and is_diag_cov
COV=np.concatenate([np.sqrt(kappa)*np.ones(int(d/4)),np.ones(int(3*d/4))])
X0=np.random.randn(n,d)*COV
U=npl.svd(np.random.randn(d,d))[0]
X=X0 if is_diag_cov else X0.dot(U)

# Assign labels based on planted weights b
b=np.ones(d) # Planted all ones weights
y=X.dot(b)

### Algo 6: Adam ###
w=np.zeros(d)
vel=np.zeros(d)
r=np.zeros(d)
alpha=0.8
rho=0
delta=1
errAdam=np.zeros(IT)
eta=4e2/npl.svd(X)[1][0]**2
#### Complete the algo below ###
valuemin = 999
optimal_alphas = []
for alpha in np.arange(0.1, 1.01, 0.01):
    for it in range(IT):
        gradient = X.T.dot(y - X.dot(w))
        vel = alpha * vel + (1 - alpha) * gradient
        r = rho * r + (1 - rho) * gradient ** 2
        vel_hat = vel / (1 - alpha ** (it+1))
        r_hat = r / (1 - rho ** (it+1))
        w += (eta / (np.sqrt(r_hat) + delta)) * vel_hat
        errAdam[it] = npl.norm(y - X.dot(w)) ** 2 / npl.norm(y) ** 2

    value = np.min(errAdam)
    if value < valuemin:
        optimal_alpha = alpha
        valuemin = value
print(f'optimal alpha: {optimal_alpha}')
#plt.scatter(np.arange(0.1, 1.01, 0.01), optimal_alphas, marker='o')
```

```
optimal alpha: 0.11
```

In [101]:
```
d=100            # Number of features
n=200            # Sample size
IT=300           # Number of iterations

kappa=100        # Condition number
is_diag_cov=False # Whether features are aligned with the eigenspectrum


# Create input features based on kappa and is_diag_cov
COV=np.concatenate([np.sqrt(kappa)*np.ones(int(d/4)),np.ones(int(3*d/4))])
X0=np.random.randn(n,d)*COV
U=npl.svd(np.random.randn(d,d))[0]
X=X0 if is_diag_cov else X0.dot(U)

# Assign labels based on planted weights b
b=np.ones(d) # Planted all ones weights
y=X.dot(b)

### Algo 6: Adam ###
w=np.zeros(d)
vel=np.zeros(d)
r=np.zeros(d)
alpha=0.8
rho=0
delta=1
errAdam=np.zeros(IT)
eta=4e2/npl.svd(X)[1][0]**2
#### Complete the algo below ###
valuemin = 999
optimal_alphas = []
for rho in np.arange(0.1, 1.01, 0.01):
    for alpha in np.arange(0.1, 1.01, 0.01):
        for it in range(300):
            gradient = X.T.dot(y - X.dot(w))
            vel = alpha * vel + (1 - alpha) * gradient
            r = rho * r + (1 - rho) * gradient ** 2
            vel_hat = vel / (1 - alpha ** (it+1))
            r_hat = r / (1 - rho ** (it+1))
            w += (eta / (np.sqrt(r_hat) + delta)) * vel_hat
            errAdam[it] = npl.norm(y - X.dot(w)) ** 2 / npl.norm(y) ** 2

        value = np.min(errAdam)
        if value < valuemin:
            optimal_rho = rho
            optimal_alpha = alpha
            valuemin = value
print(f'optimal alpha: {optimal_alpha} optimal rho: {optimal_rho}')
#plt.scatter(np.arange(0.1, 1.01, 0.01), optimal_alphas, marker='o')
```

```
optimal alpha: 0.4099999999999998 optimal rho: 0.1
```

From my implementation, I found that alpha (beta 1) is tend to be large while rho (beta 2) is tend to be small. When Combined together, rho stayed to be as small as possible but alpha went down to 0.41, which means that rho is much important than alpha.

Problem g

Adam is more consistent. Adam often exhibits good overall performance, but empirical testing and tuning are essential to identify the most suitable optimizer for a given scenario. Additionally, the choice may also be influenced by considerations such as convergence speed and computational efficiency.