Dynamic Mode Decomposition (DMD) is a powerful analysis tool for studying the dynamics of complex systems, particularly those that can be modeled by linear dynamical systems or exhibit periodic or quasi-periodic behavior. It's widely used in fluid dynamics, signal processing, and more recently, in machine learning and data science for extracting spatio-temporal features from data. The DMD algorithm decomposes a high-dimensional dataset into modes, each associated with a specific frequency and growth/decay rate, which can help in understanding the underlying dynamics of the system.

Here's a detailed step-by-step guide to the DMD algorithm:

1. Data Collection

- **Objective**: Collect sequential snapshots of the system's state over time.
- **Process**: Collect snapshot pairs $(x(tk),x(t'k))$ for $k=1,\ldots,m$, where $t'k=tk+\Delta t$ and Δt is small enough to resolve the highest frequencies in the dynamics. These snapshots are then arranged into two data matrices, X and X′: $X=[x(t1)\ x(t2)\ \cdots\ x(tm)]$;$X'=[x(t1')\ x(t2')\ \cdots\ x(tm')]$. Linear operator A that maps X to X′: $X'\approx AX$ Mathematically, A is defined as the solution that minimizes the Frobenius norm of the difference between X′ and AX, which can be computed as: $A=X'X\dagger$ where $X\dagger$ is the pseudo-inverse of X.

2. Singular Value Decomposition (SVD)

- **Objective**: Decompose the matrix X to reduce the dimensionality of the problem, making the computation more efficient and robust to noise.
- **Process**: Perform SVD on the matrix X such that $X=U\Sigma V*$, where U and V are unitary matrices, and Σ is a diagonal matrix containing the singular values.

3. Reduced Order Model Construction

- **Objective**: Create a reduced-order model that captures the most significant dynamics of the system.
- **Process**:
  1. Compute the projection of Y onto the modes in U, obtaining $U*Y$.
  2. Calculate the matrix Atilde that best maps X to Y in the reduced space: $A\text{tilde}=U*YV\Sigma-1$.

4. Eigenvalue Decomposition

- **Objective**: Extract dynamic modes and their corresponding growth rates and frequencies.
- **Process**: Perform an eigenvalue decomposition of Atilde: $A\text{tilde}W=W\Lambda$, where Λ contains eigenvalues and W the corresponding eigenvectors.

5. Dynamic Modes Reconstruction

- **Objective**: Convert the reduced-order eigenvectors back into the high-dimensional space to obtain the dynamic modes.
- **Process**: The dynamic modes are given by $\Phi=YV\Sigma-1W$, where Φ are the DMD modes in the original high-dimensional space.

6. Mode Analysis

- **Objective**: Analyze the dynamic modes and their associated eigenvalues to understand the system's dynamics.
- **Process**:
    1. Each eigenvalue λ in Λ provides information about the growth rate and frequency of its corresponding mode in Φ.
    2. The real part of λ indicates the growth or decay rate, while the imaginary part indicates the oscillation frequency.

7. Reconstruction and Forecasting

- **Objective**: Use the DMD modes and eigenvalues to reconstruct the system's dynamics or predict future states.
- **Process**: The system's state at any time t can be approximated by a linear combination of the DMD modes, weighted by their corresponding eigenvalues raised to the power of t, considering the initial conditions.

Summary

DMD offers a way to decompose complex datasets into components that reveal the inherent dynamics of the system. By identifying dominant modes and their behaviors, DMD facilitates both the understanding of the system's structure and the prediction of its future states. This makes DMD an invaluable tool in fields where understanding the time evolution of systems is crucial.

Appendix

```
import numpy as np
import matplotlib.pyplot as plt

#Define the functions
def f1(xx, tt):
   y_1 = 2 * np.cos(xx) * np.exp(1j * tt)
   return y_1

def f2(xx, tt):
   y_2 = np.sin(xx) * np.exp(3j * tt)
   return y_2


#Define time and space discretizations
xi = np.linspace(-10, 10, 401)
t = np.linspace(0, 15, 201)
dt = t[1] - t[0]
tt, xx = np.meshgrid(t,xi)
X = f1(xx, tt) + f2(xx, tt)

print(X.shape)

plt.figure(figsize=(6, 4))
plt.contourf(tt, xx, np.real(X), 20, cmap='RdGy')
plt.colorbar()
plt.xlabel('t')
plt.ylabel('x')
plt.title('Contour plot of X')
plt.show()

X_1 = X[:, :-1]
X_2 = X[:, 1:]


print(X_1.shape, X_2.shape)

U, S, VT = np.linalg.svd(X_1,full_matrices=0)
V=VT.conj().T
S=np.diag(S)
print(U.shape,S.shape, V.shape)


plt.figure(figsize=(4, 4))
plt.plot(U[:, 0], label='U[:, 0]')
plt.plot(U[:, 1], label='U[:, 1]')
plt.legend(loc='upper left')
plt.show()

plt.figure(figsize=(4, 4))
plt.plot(np.diag(S[:10]), 'o-')
plt.xlabel('Index')
plt.ylabel('Singular Value')
plt.title('First 10 Singular Values of X')
plt.show()

print(np.diag(S[:4]))


plt.figure(figsize=(4, 4))
plt.plot(V[:,0], label='V[:,0]')
plt.plot(V[:,1], label='V[:,1]')
plt.legend(loc='upper left')
plt.show()
```

Take only the first two modes

```
r =2
Ur = U[:,:r]
Sr = S[:r,:r]
Vr = V[:,:r]
print(Ur.shape, Sr.shape, Vr.shape)

print(X_2.shape)

print(Ur[:2,:2])

A_tilde =  (Ur.conj().T) @ X_2 @ Vr @ np.linalg.inv(Sr)

print(A_tilde)

Lambda, W = np.linalg.eig(A_tilde)

#Lambda = np.diag(Lambda)
print(Lambda)

# Plot the eigenvalues in the complex plane
plt.figure(figsize=(4, 4))
plt.scatter(Lambda.real, Lambda.imag)
```

```
# Plot unit circle
theta = np.linspace(0, 2*np.pi, 100)
plt.plot(np.cos(theta), np.sin(theta), linestyle='--', color='r', label='Unit Circle')

plt.axis('equal')
plt.xlim(-1.5, 1.5)
plt.ylim(-1.5, 1.5)
plt.xlabel('Real')
plt.ylabel('Imaginary')
plt.title('Complex Eigenvalues')
plt.legend(loc='upper left')
plt.show()

print(W)

Phi = X_2 @ Vr @ np.linalg.inv(Sr) @ W

print(Phi[:2,:])

Omega = np.log(Lambda)/dt

print(Omega.shape, Omega)
print("*******************")
print("Notice the imaginary parts", np.imag(Omega))


amp = np.linalg.lstsq(Phi,X_1[:,0],rcond=None)[0]

print(amp.shape[0], amp)

t_exp = np.arange(X.shape[1]) * dt
temp = np.repeat(Omega.reshape(-1,1), t_exp.size, axis=1)
dynamics = np.exp(temp * t_exp) * amp.reshape(amp.shape[0], -1)
print(t_exp.shape, temp.shape, dynamics.shape)
print(X.shape[1])
print(t_exp.size)

plt.figure(figsize=(4, 4))
plt.plot(t_exp, dynamics[0, :], '-', label='dynamics[0, :]')
plt.plot(t_exp, dynamics[1, :], '-', label='dynamics[1, :]')
plt.xlabel('t')
plt.ylabel('Dynamics')
plt.legend()
plt.title('Dynamics of the reduced model')
plt.show()
```

```
X_dmd = Phi @ dynamics
print(X_dmd.shape)

plt.figure(figsize=(15, 4))
plt.subplot(1, 3, 1)
plt.contourf(tt, xx, np.real(X), 20, cmap='RdGy')
plt.colorbar()
plt.xlabel('x')
plt.ylabel('t')
plt.title('Contour plot of X')

plt.subplot(1, 3, 2)
plt.contourf(tt, xx, np.real(X_dmd), 20, cmap='RdGy')
plt.colorbar()
plt.xlabel('x')
plt.ylabel('t')
plt.title('Contour plot of X_dmd')

X_diff = np.real(X) - np.real(X_dmd)
plt.subplot(1, 3, 3)
plt.contourf(tt, xx, X_diff , cmap='RdGy')
plt.colorbar()
plt.xlabel('x')
plt.ylabel('t')
plt.title('Contour plot of Error')


plt.show()
```