```
import numpy as np
from scipy.linalg import expm, norm
from scipy.sparse import diags, linalg as spla
import matplotlib.pyplot as plt
from time import time
from scipy.sparse.linalg import expm_multiply

# Krylov subspace approximation of e^{tA}v
def krylov_expm_multiply(A, v, m=20, t=1.0):
    """
    Inputs:
    A (ndarray): input matrix
    v (ndarray): input vector
    m (int): dimension of Krylov subspace
    t (float): time parameter

    Returns:
    ndarray: Approximation of e^{tA}v
    """
    n = len(v)
    V = np.zeros((n, m+1))
    H = np.zeros((m+1, m))
    V[:,0] = v / norm(v)

    # Arnoldi iteration
    for j in range(m):
        w = A @ V[:,j]
        for i in range(j+1):
            H[i,j] = np.dot(V[:,i], w)
            w -= H[i,j] * V[:,i]
        H[j+1,j] = norm(w)
        # terminate early if break down
        if H[j+1,j] < 1e-12:
            m = j+1
            H = H[:m+1,:m]
            V = V[:,:m+1]
            break
        V[:,j+1] = w / H[j+1,j]

    # compute matrix exponential
    expH = expm(t * H[:m,:m])
    return norm(v) * (V[:,:m] @ expH[:,0])

# this is taylor series approximation e^{tA}v
def taylor_expm_multiply(A, v, t=1.0, k=10):
    """
    Inputs:
    A (ndarray): input matrix
    v (ndarray): input vector
    t (float): time parameter
    k (int): mumber of terms of Taylor series

    Return:
    ndarray: Approximation of e^{tA}v
    """
    result = np.zeros_like(v)
    term = v.copy()
    for i in range(k):
        result += term
        term = (t * A @ term) / (i+1)
    return result

# change the krylov_dims for different matrices
def compare_methods(A, v, t=1.0, krylov_dims=range(20, 100, 10)):
    """
    Inputs:
    A (ndarray): test matrix
    v (ndarray): test vector
    t (float): time parameter
    krylov_dims: Krylov subspace dimensions to test
```

```
    Returns:
    dict: Results containing errors and timings
    """
    # reference solution (using SciPy's expm_multiply)
    print("reference solution using expm_multiply")
    ref = expm_multiply(A, v, start=0.0, stop=t, num=2)[-1]

    # change the krylov_dims for different matrices
    results = {
        'krylov_errors': [],
        'krylov_dims': list(krylov_dims),
        'taylor_errors': [],
        'taylor_terms': range(20, 100, 10),
        'methods': {}
    }

    # Krylov subspace method comparison
    print("\nKrylov subspace method")
    for m in krylov_dims:
        approx = krylov_expm_multiply(A, v, m, t)
        err = norm(approx - ref) / norm(ref)
        results['krylov_errors'].append(err)
        print(f"m={m}: relative error = {err:.2e}")

    # Taylor series comparison
    print("\nTesting Taylor series method...")
    for k in results['taylor_terms']:
        approx = taylor_expm_multiply(A, v, t, k)
        err = norm(approx - ref) / norm(ref)
        results['taylor_errors'].append(err)
        print(f"k={k}: relative error = {err:.2e}")

    # Direct expm (only for small matrices)
    if A.shape[0] <= 1000:
        print("\nDirect expm method...")
        try:
            full_expm = expm(t * A)
            approx = full_expm @ v
            err = norm(approx - ref) / norm(ref)
            results['methods']['expm'] = err
            print(f"Direct expm: relative error = {err:.2e}")
        except:
            print("expm failed (possibly too large matrix)")

    return results

def plot_results(results, title=""):
    """Plot comparison results."""
    plt.figure(figsize=(12, 6))

    # Krylov error convergence
    plt.subplot(1, 2, 1)
    plt.semilogy(results['krylov_dims'], results['krylov_errors'], 'o-')
    plt.xlabel('Krylov subspace dimension (m)')
    plt.ylabel('Relative error')
    plt.title('Krylov Method Convergence')
    plt.grid(True)

    # Taylor error convergence
    plt.subplot(1, 2, 2)
    plt.semilogy(results['taylor_terms'], results['taylor_errors'], 's-')
    plt.xlabel('Number of Taylor terms')
    plt.ylabel('Relative error')
    plt.title('Taylor Series Convergence')
    plt.grid(True)

    plt.suptitle(title)
    plt.tight_layout()
    plt.show()
```

```
# ====================
# Test Cases
# ====================

def test_small_dense():
    """Small dense matrix test case."""
    np.random.seed(42)
    n = 100
    # scale this for stability
    A = np.random.randn(n, n) / n
    v = np.random.randn(n)
    t = 1.0

    print(krylov_expm_multiply(A, v, m=20, t=1.0))

    print("Small Dense Matrix Test")
    results = compare_methods(A, v, t)
    plot_results(results, "Small Dense Matrix")

def test_symmetric_negative_definite():
    """Symmetric Negative Definite Matrix test case."""
    n = 100
    # Creating a symmetric negative definite matrix by setting negative values on the dia
gonal
    A = -np.diag(np.linspace(1, n, n))  # Negative diagonal with increasing values
    v = np.random.randn(n)  # Random vector
    t = 0.1  # Time parameter

    print("\nSymmetric Negative Definite Matrix Test")
    results = compare_methods(A, v, t)
    plot_results(results, f"Symmetric Negative Definite Matrix")


def test_sparse_2d_laplacian():
    """2D Laplacian matrix from finite difference discretization."""
    n = 30
    N = n * n
    h2 = (1.0 / (n + 1))**2

    # 1D Laplacian
    main_diag = 4.0 * np.ones(n)
    off_diag = -1.0 * np.ones(n - 1)
    T = diags([off_diag, main_diag, off_diag], [-1, 0, 1], shape=(n, n))

    I = diags([np.ones(n)], [0])
    A = (np.kron(I.toarray(), T.toarray()) + np.kron(T.toarray(), I.toarray())) / h2
    # make this negative definite
    A = -A

    v = np.random.randn(N)
    # small t for stability
    t = 0.01
    print("\nSparse 2D Laplacian Matrix Test")
    results = compare_methods(A, v, t)
    plot_results(results, f"Sparse 2D Laplacian Matrix")


if __name__ == "__main__":
    # test_small_dense()
    # test_symmetric_negative_definite()
    # test_sparse_2d_laplacian()
    pass
```