Matrix Exponential Analysis and Calculation

Cayley-Hamilton Theorem and Matrix Exponential

The answer key uses the Cayley-Hamilton theorem to find the matrix exponential e^A for a specific 2x2 matrix A.

Explanation

Cayley-Hamilton Theorem Introduction

We start with the Cayley-Hamilton theorem, which states that every square matrix satisfies its own characteristic polynomial. This is a fundamental theorem used to simplify calculations involving powers of matrices and functions of matrices like the matrix exponential.

Matrix A and Characteristic Polynomial

Given matrix A :

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

The characteristic polynomial $\chi(\lambda)$ is calculated as $\det(\lambda I - A)$ which turns out to be $\lambda^2 - 5\lambda - 2$.

Applying Cayley-Hamilton Theorem

By Cayley-Hamilton, A satisfies $\chi(A) = 0$. Substituting A into its characteristic polynomial gives:

$$A^2 - 5A - 2I = 0$$

This equation is verified by explicitly calculating A^2, multiplying A by 5, and adding 2 times the identity matrix I, resulting in the zero matrix. This confirms the theorem's prediction.

Matrix Exponential Explanation

Further, we can compute the matrix exponential e^A. But, note that it is not simply e^t times each element of the matrix.

How This Theorom Helps Compute e^A

To compute e^A when A is a 2x2 matrix, the Cayley-Hamilton theorem can be incredibly helpful. Since we know from Cayley-Hamilton that:

$$A^2 = 5A + 2I$$

we can express higher powers of A in terms of A and I alone. This simplification can be used to express the matrix exponential, which is a series involving powers of A:

$$e^A = I + A + \frac{A^2}{2!} + \frac{A^3}{3!} + \cdots$$

Using the relationship $A^2 = 5A + 2I$, we can rewrite all higher powers of A (like A^3, A^4, etc.) in terms of A and I, making the series more manageable.

In practical computation:

1. Calculate higher powers using the Cayley-Hamilton relationship.
2. Simplify the infinite series using these relationships.

This approach dramatically reduces the complexity of calculating the matrix exponential by eliminating the need to compute very high powers of the matrix directly, which can be computationally expensive and prone to numerical errors.

Python Implementation for Matrix Exponential Calculation

To write a Python function that computes the matrix exponential for a 2x2 matrix using the Cayley-Hamilton theorem, you can follow a systematic approach. First, determine the characteristic polynomial and use it to simplify higher powers of the matrix. Then, compute the exponential of the matrix using its simplified powers series.

Python Code

```
import numpy as np

def matrix_exponential(A):
    # Step 1: Calculate trace and determinant
    trace_A = np.trace(A)
    det_A = np.linalg.det(A)

    # Step 2: Calculate A^2 using Cayley-Hamilton theorem
    I = np.eye(2)  # Identity matrix
    A_squared = trace_A * A - det_A * I

    # Step 3: Compute matrix exponential using the series expansion
    exp_A = I  # Start with the identity matrix
    exp_A += A  # Add A
    exp_A += A_squared / 2  # Add A^2 / 2!

    # Calculate A^3 using A^2 and A
    A_cubed = np.dot(A, A_squared)
    exp_A += A_cubed / 6  # Add A^3 / 3!

    return exp_A

# Example Matrix
A = np.array([[1, 2],
              [3, 4]])

# Calculate matrix exponential
exp_A = matrix_exponential(A)
print("Exponential of A is:")
print(exp_A)
```

Analytical Solution

To provide an analytical solution for $e^{A\Delta t}$, where A is the matrix defined by:

$$A = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{b}{m} \end{bmatrix}$$

we begin by computing the characteristic polynomial and using it to determine the eigenvalues. Based on these eigenvalues, we'll construct the exponential of the matrix.

Steps to Analyze $e^{A\Delta t}$

Characteristic Polynomial

The characteristic polynomial of matrix A is given by:

$$\chi(\lambda) = \lambda^2 - \text{tr}(A)\lambda + \det(A)$$

where:

- $\text{tr}(A) = 0 + (-\frac{b}{m}) = -\frac{b}{m}$

- $\det(A) = -\left(-\frac{k}{m}\right) = \frac{k}{m}$

Thus, the characteristic polynomial becomes:

$$\chi(\lambda) = \lambda^2 + \frac{b}{m}\lambda + \frac{k}{m}$$

Eigenvalues

Solve the quadratic equation:

$$\lambda^2 + \frac{b}{m}\lambda + \frac{k}{m} = 0$$

Using the quadratic formula:

$$\lambda = \frac{-\frac{b}{m} \pm \sqrt{\left(\frac{b}{m}\right)^2 - 4 \cdot \frac{k}{m}}}{2}$$

Simplifying further:

$$\lambda = \frac{-b \pm \sqrt{b^2 - 4km}}{2m}$$

Let $\omega = \sqrt{\frac{b^2-4km}{(2m)^2}}$, the eigenvalues are:

$$\lambda_{1,2} = -\frac{b}{2m} \pm \omega$$

Matrix Exponential

For A with distinct real eigenvalues or a complex conjugate pair, we can express $e^{A\Delta t}$ using the formula involving eigenvectors or directly using the Jordan form if it simplifies calculations.

- If $b^2 - 4km > 0$ (distinct real roots):
  - Use the real eigenvectors and eigenvalues to form $e^{A\Delta t}$.
- If $b^2 - 4km = 0$ (repeated real root):
  - The matrix exponential is calculated using the Jordan form approach for repeated eigenvalues.
- If $b^2 - 4km < 0$ (complex conjugate roots):
  - The eigenvalues are $\alpha \pm i\beta$ where $\alpha = -\frac{b}{2m}$ and $\beta = \omega$.
  - The matrix exponential $e^{A\Delta t}$ can be expressed as:

$$e^{A\Delta t} = e^{\alpha\Delta t}\left(\cos(\beta\Delta t)I + \frac{\sin(\beta\Delta t)}{\beta}(A - \alpha I)\right)$$

  - This results in oscillatory motion dampened by $e^{\alpha\Delta t}$.

Python Code

```
import numpy as np


def matrix_exponential_analytical(A, dt):
    # Extract parameters from A assuming A is structured as specified
    b = -A[1, 1] * A[1, 0]
    k = -A[1, 0]
    m = 1 / A[1, 0]

    # Compute coefficients for the characteristic equation
    alpha = -b / (2 * m)
    omega_squared = (b**2 - 4 * k * m) / (4 * m**2)

    # Initialize the identity matrix
    I = np.eye(2)
```

```
if omega_squared > 0:
    # Distinct real eigenvalues case
    omega = np.sqrt(omega_squared)
    exp_matrix = np.cosh(omega * dt) * I + (np.sinh(omega * dt) / omega) * (A - alpha * I)
elif omega_squared == 0:
    # Repeated real roots case
    exp_matrix = I + dt * (A - alpha * I)
else:
    # Complex conjugate roots case
    omega = np.sqrt(-omega_squared)
    exp_matrix = np.exp(alpha * dt) * (np.cos(omega * dt) * I + (np.sin(omega * dt) / omega) * (A -

# Factor out e^(alpha * dt) for the complex and repeated root cases
exp_matrix *= np.exp(alpha * dt)

return exp_matrix
```