/\* Define the matrix \*/

A: matrix([1, 2], [2, 3]);

/\* Calculate eigenvalues \*/

eigenvalues: eigenvalues(A);

/\* Calculate eigenvectors \*/

eigenvectors: eigenvectors(A);

/\* Display results \*/

print("Eigenvalues: ", eigenvalues);

print("Eigenvectors: ", eigenvectors);

/\* Define the matrix \*/

A: matrix([1, 2], [2, 3]);

/\* Characteristic polynomial \*/

char\_poly: charpoly(A, λ);

/\* Substitute the matrix A into the characteristic polynomial \*/

cayley\_hamilton\_expr: subst(A, λ, char\_poly);

/\* Simplify the expression \*/

simplified\_expr: ratsimp(cayley\_hamilton\_expr);

/\* Display the result \*/

print("Characteristic Polynomial: ", char\_poly);

print("Cayley-Hamilton Expression (Should be zero): ", simplified\_expr);

/\* Define vectors u and v \*/

u: [1, 2, 3];

v: [4, 5, 6];

/\* Compute the dot product of u and v \*/

dot\_prod: dotproduct(u, v);

/\* Display the result \*/

print("Dot Product: ", dot\_prod);

/\* Compute the cross product of u and v \*/

cross\_prod: crossproduct(u, v);

/\* Display the result \*/

print("Cross Product: ", cross\_prod);

/\* Define vectors u and v \*/

u: [1, 2, 3];

v: [4, 5, 6];

/\* Define scalar c \*/

c: 3;

/\* Check closure under addition: u + v \*/

addition\_closure: u + v;

/\* Check closure under scalar multiplication: c \* u \*/

scalar\_multiplication\_closure: c \* u;

/\* Check for zero vector \*/

zero\_vector: vector([0, 0, 0]);

/\* Display results \*/

print("Closure under addition (u + v): ", addition\_closure);

print("Closure under scalar multiplication (c \* u): ", scalar\_multiplication\_closure);

print("Zero vector: ", zero\_vector);

/\* Define matrices A and B \*/

A: matrix([1, 2, 3], [4, 5, 6]);

B: matrix([6, 5, 4], [3, 2, 1]);

/\* Get the dimensions of matrices A and B \*/

[m, n]: dimensions(A);

/\* Initialize the result matrix C with zeros \*/

C: zeromatrix(m, n);

/\* Perform matrix addition using loops \*/

for i: 1 thru m do (

for j: 1 thru n do (

C[i, j]: A[i, j] + B[i, j]

)

);

/\* Display the result matrix C \*/

print("Result of A + B: ", C);

/\* Define the matrix \*/

A: matrix([1, 2], [2, 3]);

/\* Calculate eigenvalues \*/

eigenvalues: eigenvalues(A);

/\* Calculate eigenvectors \*/

eigenvectors: eigenvectors(A);

/\* Display results \*/

print("Eigenvalues: ", eigenvalues);

print("Eigenvectors: ", eigenvectors);

/\* Define the matrix \*/

A: matrix([1, 2], [2, 3]);

/\* Characteristic polynomial \*/

char\_poly: charpoly(A, λ);

/\* Substitute the matrix A into the characteristic polynomial \*/

cayley\_hamilton\_expr: subst(A, λ, char\_poly);

/\* Simplify the expression \*/

simplified\_expr: ratsimp(cayley\_hamilton\_expr);

/\* Display the result \*/

print("Characteristic Polynomial: ", char\_poly);

print("Cayley-Hamilton Expression (Should be zero): ", simplified\_expr);

/\* Define vectors u and v \*/

u: [1, 2, 3];

v: [4, 5, 6];

/\* Compute the dot product of u and v \*/

dot\_prod: dotproduct(u, v);

/\* Display the result \*/

print("Dot Product: ", dot\_prod);

/\* Compute the cross product of u and v \*/

cross\_prod: crossproduct(u, v);

/\* Display the result \*/

print("Cross Product: ", cross\_prod);

/\* Define vectors u and v \*/

u: [1, 2, 3];

v: [4, 5, 6];

/\* Define scalar c \*/

c: 3;

/\* Check closure under addition: u + v \*/

addition\_closure: u + v;

/\* Check closure under scalar multiplication: c \* u \*/

scalar\_multiplication\_closure: c \* u;

/\* Check for zero vector \*/

zero\_vector: vector([0, 0, 0]);

/\* Display results \*/

print("Closure under addition (u + v): ", addition\_closure);

print("Closure under scalar multiplication (c \* u): ", scalar\_multiplication\_closure);

print("Zero vector: ", zero\_vector);

/\* Define matrices A and B \*/

A: matrix([1, 2, 3], [4, 5, 6]);

B: matrix([6, 5, 4], [3, 2, 1]);

/\* Get the dimensions of matrices A and B \*/

[m, n]: dimensions(A);

/\* Initialize the result matrix C with zeros \*/

C: zeromatrix(m, n);

/\* Perform matrix addition using loops \*/

for i: 1 thru m do (

for j: 1 thru n do (

C[i, j]: A[i, j] + B[i, j]

)

);

/\* Display the result matrix C \*/

print("Result of A + B: ", C);

/\* Define vectors \*/

v1: matrix([1], [2], [3]);

v2: matrix([4], [5], [6]);

/\* Define scalars \*/

a: 2;

b: 3;

/\* Form the linear combination \*/

linear\_combination: a \* v1 + b \* v2;

/\* Expand and display the result \*/

expand(linear\_combination);

/\* Define the vectors \*/

v1: matrix([1], [2]);

v2: matrix([3], [4]);

/\* Define the coefficients with the constraint that their sum is 1 \*/

lambda1: 0.4;

lambda2: 1 - lambda1; /\* Ensures that lambda1 + lambda2 = 1 \*/

/\* Form the convex combination \*/

convex\_combination: lambda1 \* v1 + lambda2 \* v2;

/\* Expand and display the result \*/

expand(convex\_combination);

/\* Load the linear algebra package \*/

load("linearalgebra");

/\* Define the vectors \*/

v1: matrix([1], [2], [3]);

v2: matrix([4], [5], [6]);

v3: matrix([7], [8], [9]);

/\* Form a matrix with the vectors as columns \*/

M: matrix([1, 4, 7], [2, 5, 8], [3, 6, 9]);

/\* Check the determinant \*/

determinant(M);

/\* Alternatively, check the rank \*/

rank(M);

/\* Load the linear algebra package \*/

load("linearalgebra");

/\* Define the matrix \*/

A: matrix([6, 2, 1], [2, 3, 1], [1, 1, 1]);

/\* Calculate determinants of leading principal minors \*/

minor1: determinant(submatrix(A, 1, 1));

minor2: determinant(submatrix(A, 1..2, 1..2));

minor3: determinant(A);

/\* Check if all minors are positive \*/

all\_positive: (minor1 > 0) and (minor2 > 0) and (minor3 > 0);

1..2 specifies a range of indices from 1 to 2, and it's useful for selecting parts of a matrix.

/\* Load the linear algebra package \*/

load("linearalgebra");

/\* Define the matrix \*/

A: matrix([4, 1], [2, 3]);

/\* Find eigenvalues and eigenvectors \*/

eigenvalues\_A: eigenvalues(A);

eigenvectors\_A: eigenvectors(A);

/\* Construct the matrix P and the diagonal matrix D \*/

/\* Assume eigenvectors\_A returns eigenvectors [v1, v2] and eigenvalues\_A returns [lambda1, lambda2] \*/

P: matrix([v1\_1, v2\_1], [v1\_2, v2\_2]); /\* Replace with actual eigenvector components \*/

D: diag(lambda1, lambda2); /\* Replace with actual eigenvalues \*/

/\* Verify the diagonalization \*/

is(equal(A, P . D . invert(P)));

/\* Load the linear algebra package \*/

load("linearalgebra");

/\* Define the matrix \*/

A: matrix([2, 1], [1, 2]);

/\* Define the variable for lambda \*/

lambda: variable;

/\* Define the identity matrix \*/

I: identity(2); /\* 2x2 identity matrix \*/

/\* Calculate the characteristic polynomial \*/

char\_poly: determinant(A - lambda \* I);

/\* Expand the characteristic polynomial \*/

char\_poly\_expanded: expand(char\_poly);

/\* Substitute the matrix A into the characteristic polynomial \*/

pA: subst(A, lambda, char\_poly\_expanded);

/\* Verify the Cayley-Hamilton theorem \*/

result: is(equal(pA, 0));

Define function

/\* Load the linear algebra package \*/

load("linearalgebra");

/\* Define the function to add two matrices \*/

add\_matrices(A, B) := A + B;

/\* Define two matrices \*/

M1: matrix([1, 2], [3, 4]); /\* 2x2 matrix \*/

M2: matrix([5, 6], [7, 8]); /\* 2x2 matrix \*/

/\* Call the function to add the matrices \*/

result: add\_matrices(M1, M2);

/\* Display the result \*/

result; /\* Should return the matrix [[6, 8], [10, 12]] \*/

Block of function

/\* Load the linear algebra package \*/

load("linearalgebra");

/\* Define a simple multi-line function to add two matrices \*/

add\_matrices(A, B) := block(

C: A + B, /\* Add the matrices \*/

return(C) /\* Return the result \*/

);

/\* Define two matrices \*/

M1: matrix([1, 2], [3, 4]); /\* 2x2 matrix \*/

M2: matrix([5, 6], [7, 8]); /\* 2x2 matrix \*/

/\* Call the function to add the matrices \*/

result: add\_matrices(M1, M2);

/\* Display the result \*/

result; /\* Should return [[6, 8], [10, 12]] \*/

/\* Define a function to classify numbers \*/

classify\_number(n) := block(

if n > 0 then "Positive"

elseif n < 0 then "Negative"

else "Zero"

);

/\* Test the function \*/

result1: classify\_number(5); /\* Returns "Positive" \*/

result2: classify\_number(-3); /\* Returns "Negative" \*/

result3: classify\_number(0); /\* Returns "Zero \*/

/\* Display results \*/

[result1, result2, result3];

If else

load(vect);

/\* Gradient of a scalar field \*/

f : x^2 + y^2 + z^2;

grad(f);

/\* Divergence of a vector field \*/

F : [x^2, y^2, z^2];

div(F);

/\* Curl of a vector field \*/

curl(F);

f : x^2 + y^2 + z^2;

grad\_f : [diff(f, x), diff(f, y), diff(f, z)];

F : [x^2, y^2, z^2];

div\_F : diff(F[1], x) + diff(F[2], y) + diff(F[3], z);

/\* Define the vector field components \*/

F\_x : x^2;

F\_y : y^2;

F\_z : z^2;

/\* Define the matrix for the curl operation \*/

M : matrix(

['i, 'j, 'k], /\* First row with unit vectors \*/

[diff(1, x), diff(1, y), diff(1, z)], /\* Second row with partial derivatives \*/

[F\_x, F\_y, F\_z] /\* Third row with vector field components \*/

);

/\* Calculate the determinant to get the curl \*/

curl\_F : determinant(M);