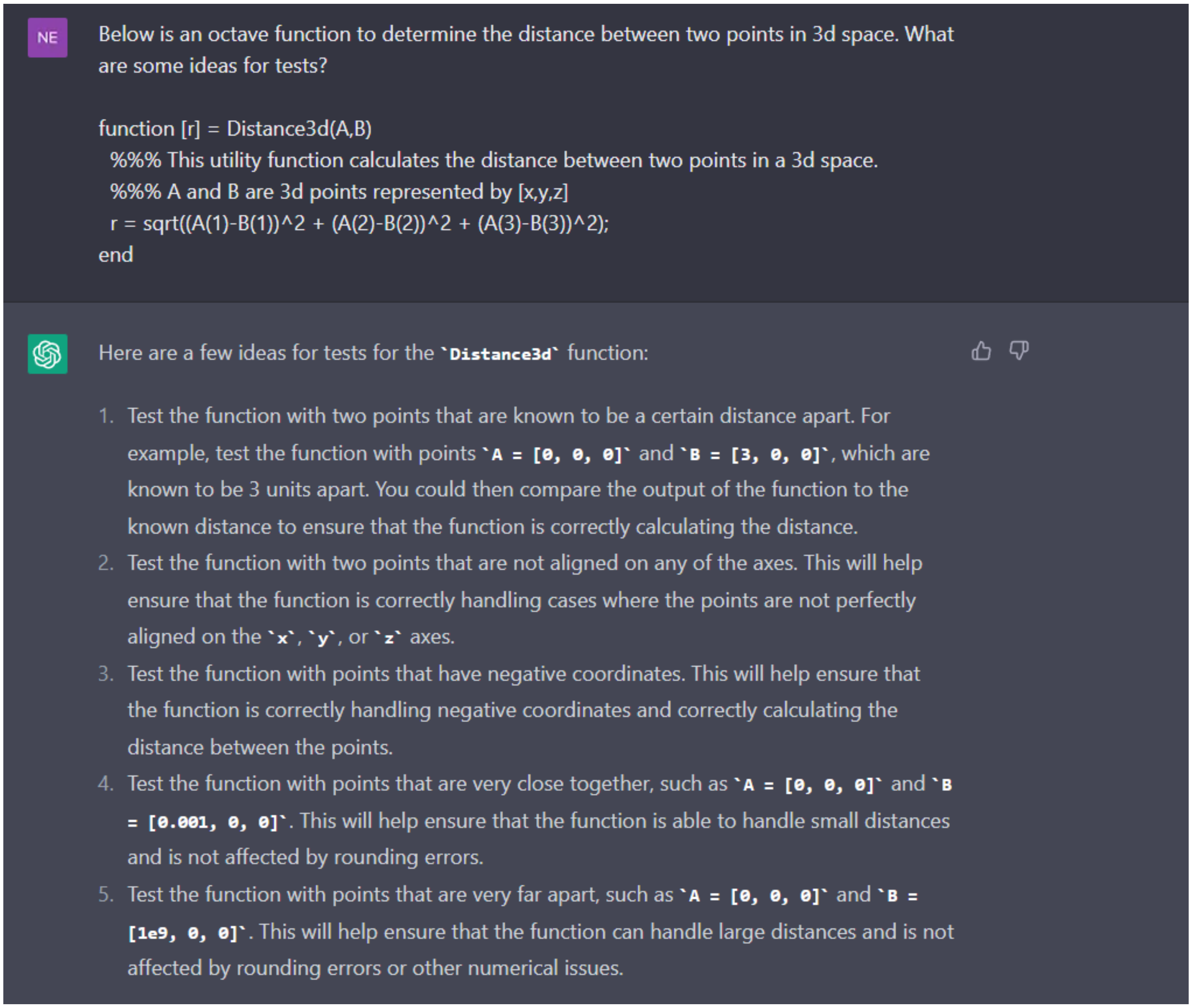
I asked ChatGPT for some test ideas:



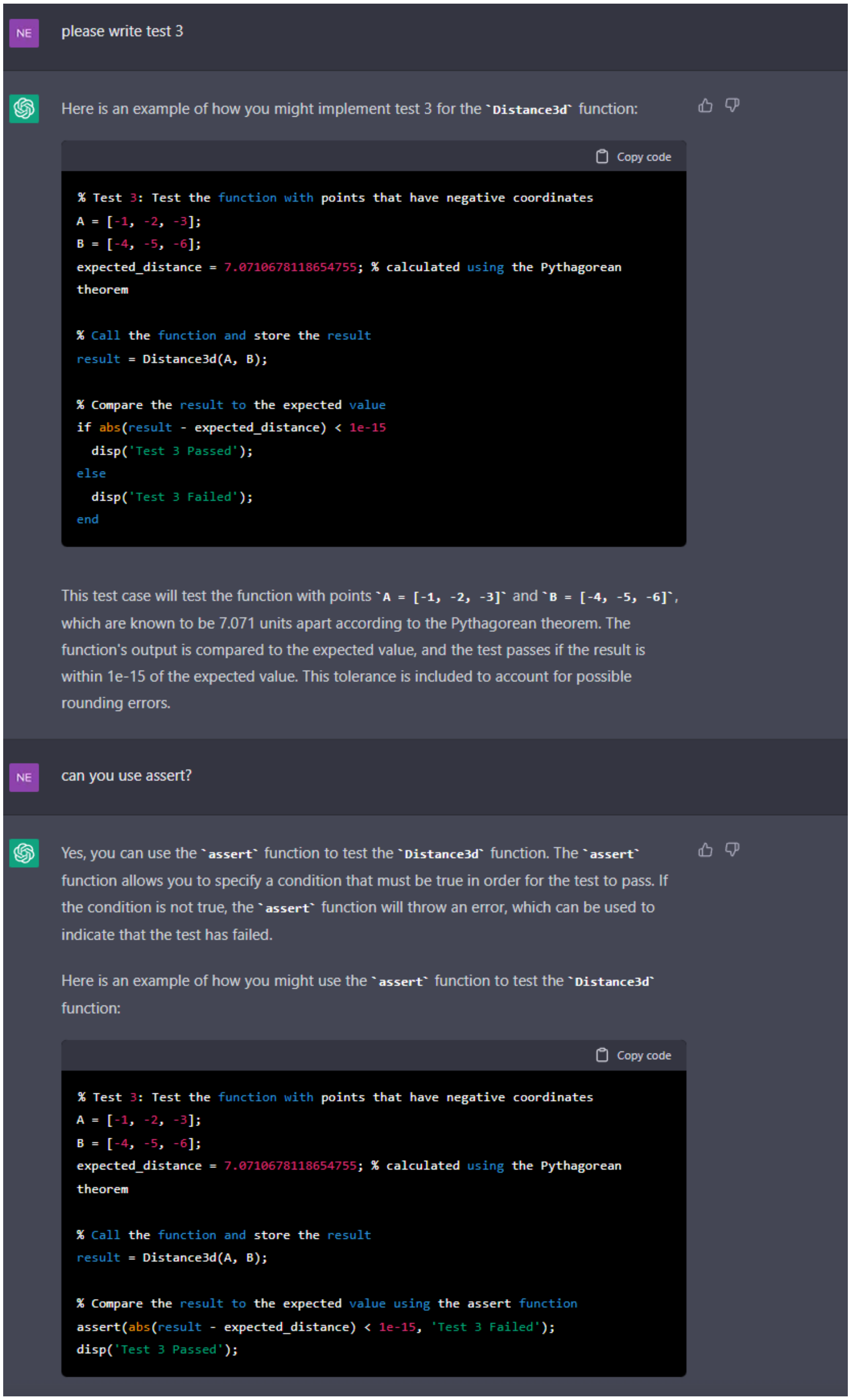
Below is an octave function to determine the distance between two points in 3d space. What are some ideas for tests?

```
function [r] = Distance3d(A,B)
  %%% This utility function calculates the distance between two points in a 3d space.
  %%% A and B are 3d points represented by [x,y,z]
  r = sqrt((A(1)-B(1))^2 + (A(2)-B(2))^2 + (A(3)-B(3))^2);
end
```

Here are a few ideas for tests for the `Distance3d` function:

1. Test the function with two points that are known to be a certain distance apart. For example, test the function with points `A = [0, 0, 0]` and `B = [3, 0, 0]`, which are known to be 3 units apart. You could then compare the output of the function to the known distance to ensure that the function is correctly calculating the distance.
2. Test the function with two points that are not aligned on any of the axes. This will help ensure that the function is correctly handling cases where the points are not perfectly aligned on the `x`, `y`, or `z` axes.
3. Test the function with points that have negative coordinates. This will help ensure that the function is correctly handling negative coordinates and correctly calculating the distance between the points.
4. Test the function with points that are very close together, such as `A = [0, 0, 0]` and `B = [0.001, 0, 0]`. This will help ensure that the function is able to handle small distances and is not affected by rounding errors.
5. Test the function with points that are very far apart, such as `A = [0, 0, 0]` and `B = [1e9, 0, 0]`. This will help ensure that the function can handle large distances and is not affected by rounding errors or other numerical issues.

I then asked it to write test #3



please write test 3

Here is an example of how you might implement test 3 for the `Distance3d` function:

```
% Test 3: Test the function with points that have negative coordinates
A = [-1, -2, -3];
B = [-4, -5, -6];
expected_distance = 7.0710678118654755; % calculated using the Pythagorean
theorem

% Call the function and store the result
result = Distance3d(A, B);

% Compare the result to the expected value
if abs(result - expected_distance) < 1e-15
  disp('Test 3 Passed');
else
  disp('Test 3 Failed');
end
```

This test case will test the function with points `A = [-1, -2, -3]` and `B = [-4, -5, -6]`, which are known to be 7.071 units apart according to the Pythagorean theorem. The function's output is compared to the expected value, and the test passes if the result is within 1e-15 of the expected value. This tolerance is included to account for possible rounding errors.

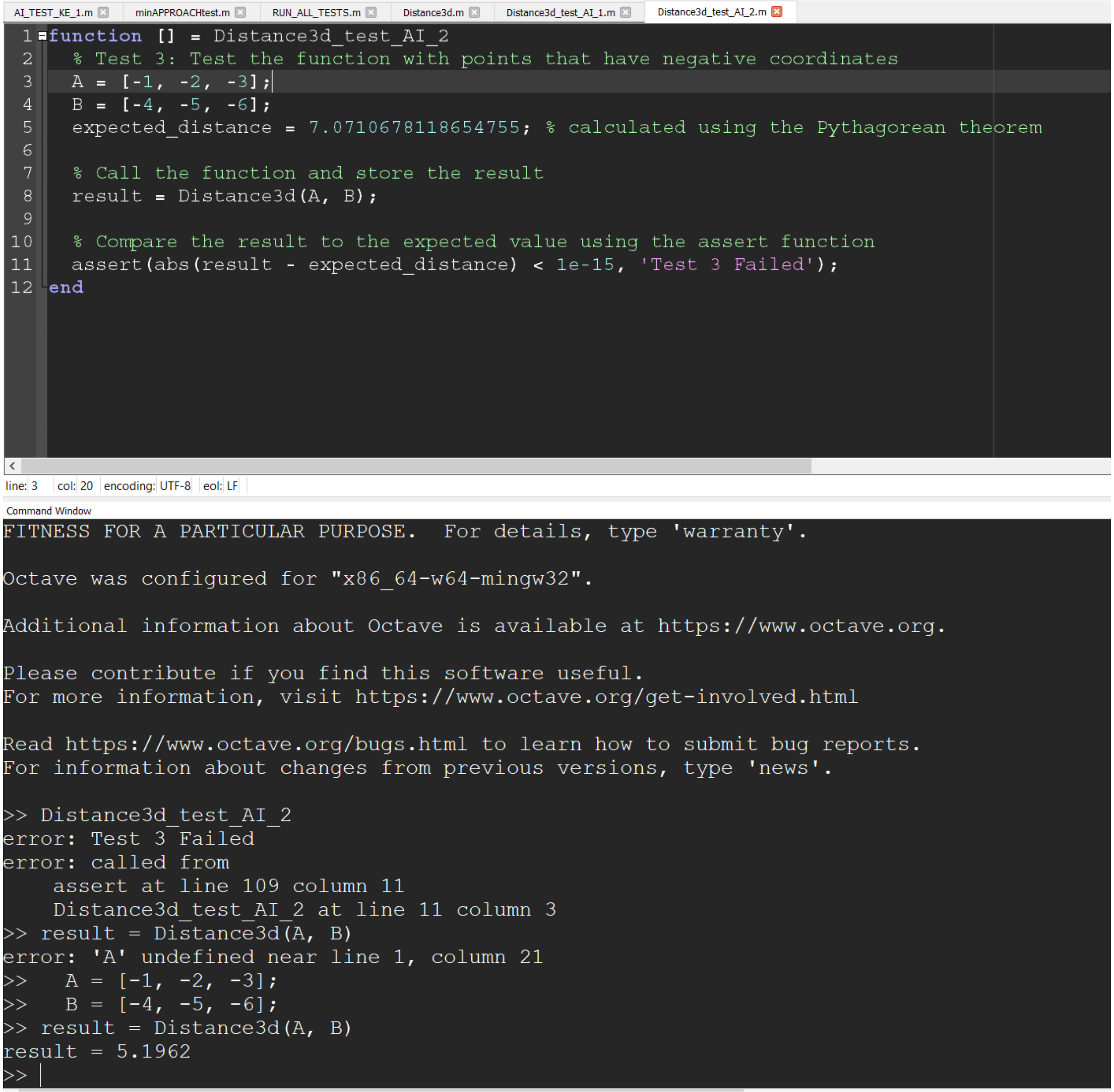
can you use assert?

Yes, you can use the `assert` function to test the `Distance3d` function. The `assert` function allows you to specify a condition that must be true in order for the test to pass. If the condition is not true, the `assert` function will throw an error, which can be used to indicate that the test has failed.

Here is an example of how you might use the `assert` function to test the `Distance3d` function:

```
% Test 3: Test the function with points that have negative coordinates
A = [-1, -2, -3];
B = [-4, -5, -6];
expected_distance = 7.0710678118654755; % calculated using the Pythagorean
theorem

% Call the function and store the result
result = Distance3d(A, B);

% Compare the result to the expected value using the assert function
assert(abs(result - expected_distance) < 1e-15, 'Test 3 Failed');
disp('Test 3 Passed');
```

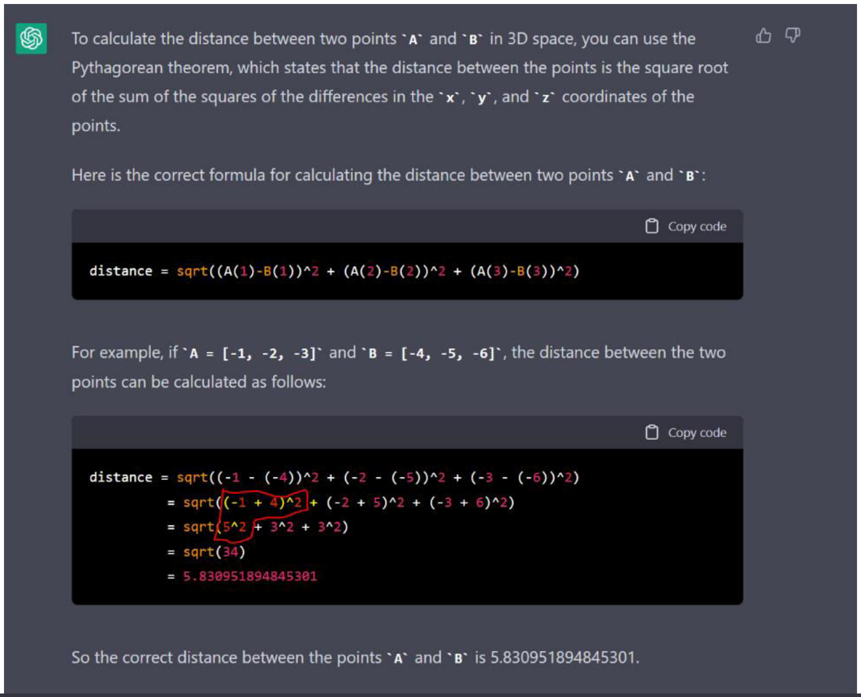
I manually removed the final line of the test, as the tests should pass silently.



```
 1 function [] = Distance3d_test_AI_2
 2   % Test 3: Test the function with points that have negative coordinates
 3   A = [-1, -2, -3];
 4   B = [-4, -5, -6];
 5   expected_distance = 7.0710678118654755; % calculated using the Pythagorean theorem
 6 
 7   % Call the function and store the result
 8   result = Distance3d(A, B);
 9 
10   % Compare the result to the expected value using the assert function
11   assert(abs(result - expected_distance) < 1e-15, 'Test 3 Failed');
12 end
```

```
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warranty'.

Octave was configured for "x86_64-w64-mingw32".

Additional information about Octave is available at https://www.octave.org.

Please contribute if you find this software useful.
For more information, visit https://www.octave.org/get-involved.html

Read https://www.octave.org/bugs.html to learn how to submit bug reports.
For information about changes from previous versions, type 'news'.

>> Distance3d_test_AI_2
error: Test 3 Failed
error: called from
    assert at line 109 column 11
    Distance3d_test_AI_2 at line 11 column 3
>> result = Distance3d(A, B)
error: 'A' undefined near line 1, column 21
>>   A = [-1, -2, -3];
>>   B = [-4, -5, -6];
>> result = Distance3d(A, B)
result = 5.1962
>> 
```

I asked ChatGPT to recalculate the distance between the point A and point B, as it applied the Pythagorean theorem incorrectly, only to be disappointed by its ability to do basic math…



To calculate the distance between two points `` `A` `` and `` `B` `` in 3D space, you can use the Pythagorean theorem, which states that the distance between the points is the square root of the sum of the squares of the differences in the `` `x` ``, `` `y` ``, and `` `z` `` coordinates of the points.

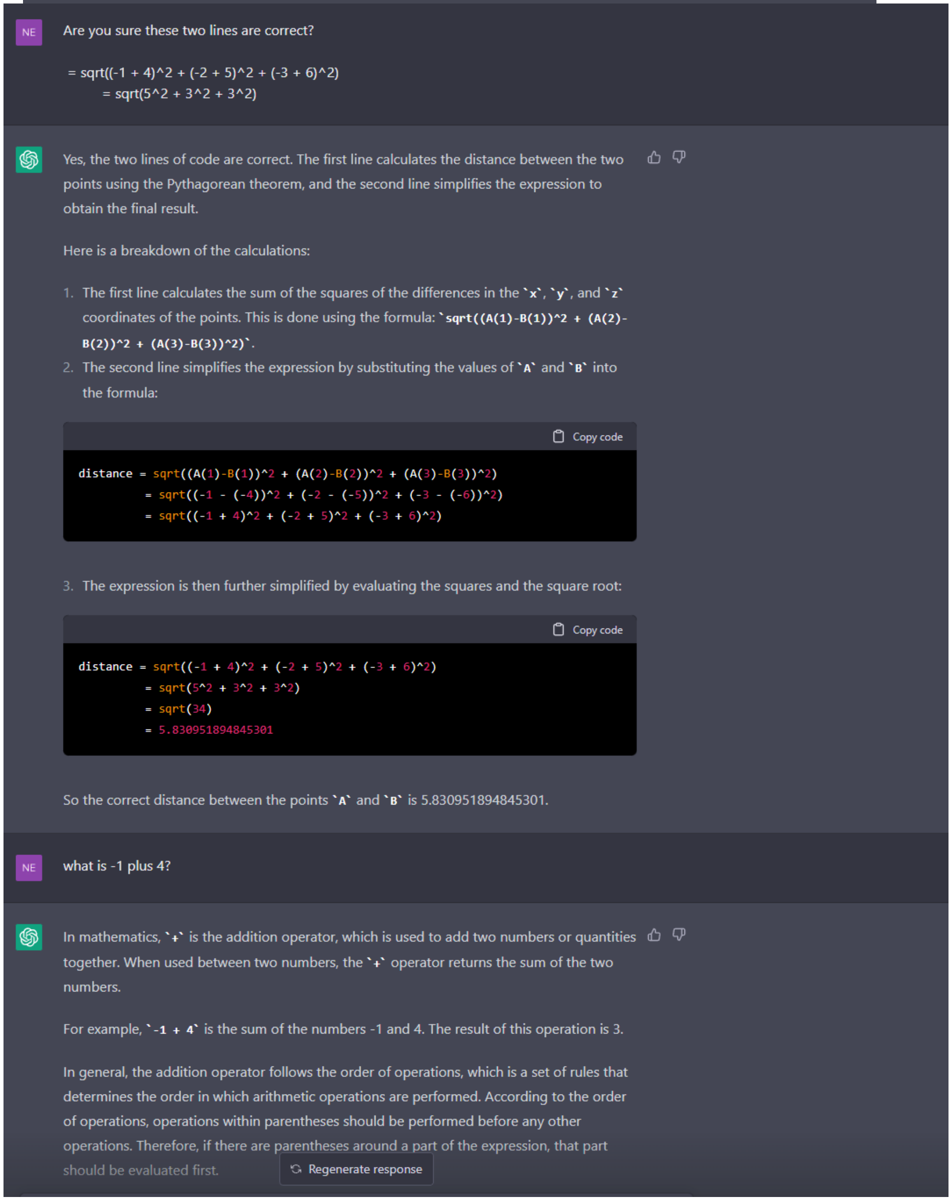
Here is the correct formula for calculating the distance between two points `` `A` `` and `` `B` ``:

```
distance = sqrt((A(1)-B(1))^2 + (A(2)-B(2))^2 + (A(3)-B(3))^2)
```

For example, if `` `A = [-1, -2, -3]` `` and `` `B = [-4, -5, -6]` ``, the distance between the two points can be calculated as follows:

```
distance = sqrt((-1 - (-4))^2 + (-2 - (-5))^2 + (-3 - (-6))^2)
         = sqrt((-1 + 4)^2 + (-2 + 5)^2 + (-3 + 6)^2)
         = sqrt(5^2 + 3^2 + 3^2)
         = sqrt(34)
         = 5.830951894845301
```

So the correct distance between the points `` `A` `` and `` `B` `` is 5.830951894845301.



Are you sure these two lines are correct?

= sqrt((-1 + 4)^2 + (-2 + 5)^2 + (-3 + 6)^2)
    = sqrt(5^2 + 3^2 + 3^2)

Yes, the two lines of code are correct. The first line calculates the distance between the two points using the Pythagorean theorem, and the second line simplifies the expression to obtain the final result.

Here is a breakdown of the calculations:

1. The first line calculates the sum of the squares of the differences in the `` `x` ``, `` `y` ``, and `` `z` `` coordinates of the points. This is done using the formula: `` `sqrt((A(1)-B(1))^2 + (A(2)-B(2))^2 + (A(3)-B(3))^2)` ``.
2. The second line simplifies the expression by substituting the values of `` `A` `` and `` `B` `` into the formula:

```
distance = sqrt((A(1)-B(1))^2 + (A(2)-B(2))^2 + (A(3)-B(3))^2)
         = sqrt((-1 - (-4))^2 + (-2 - (-5))^2 + (-3 - (-6))^2)
         = sqrt((-1 + 4)^2 + (-2 + 5)^2 + (-3 + 6)^2)
```

3. The expression is then further simplified by evaluating the squares and the square root:

```
distance = sqrt((-1 + 4)^2 + (-2 + 5)^2 + (-3 + 6)^2)
         = sqrt(5^2 + 3^2 + 3^2)
         = sqrt(34)
         = 5.830951894845301
```

So the correct distance between the points `` `A` `` and `` `B` `` is 5.830951894845301.

what is -1 plus 4?

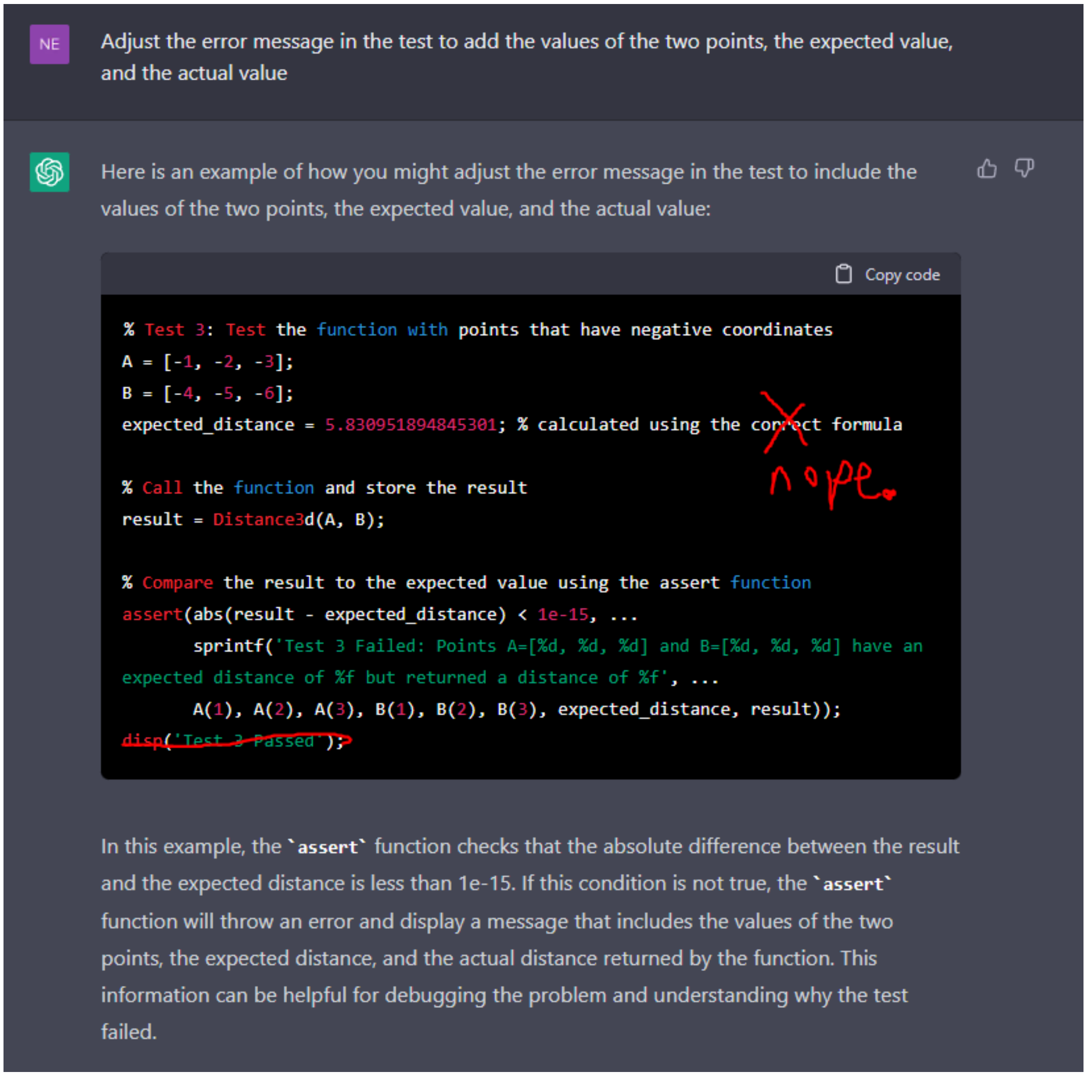
In mathematics, `` `+` `` is the addition operator, which is used to add two numbers or quantities together. When used between two numbers, the `` `+` `` operator returns the sum of the two numbers.

For example, `` `-1 + 4` `` is the sum of the numbers -1 and 4. The result of this operation is 3.

In general, the addition operator follows the order of operations, which is a set of rules that determines the order in which arithmetic operations are performed. According to the order of operations, operations within parentheses should be performed before any other operations. Therefore, if there are parentheses around a part of the expression, that part should be evaluated first.

Uhhh… Well at least it is confident that -1 + 4 = 5 = 3?

I asked it to add some more information to the error message:



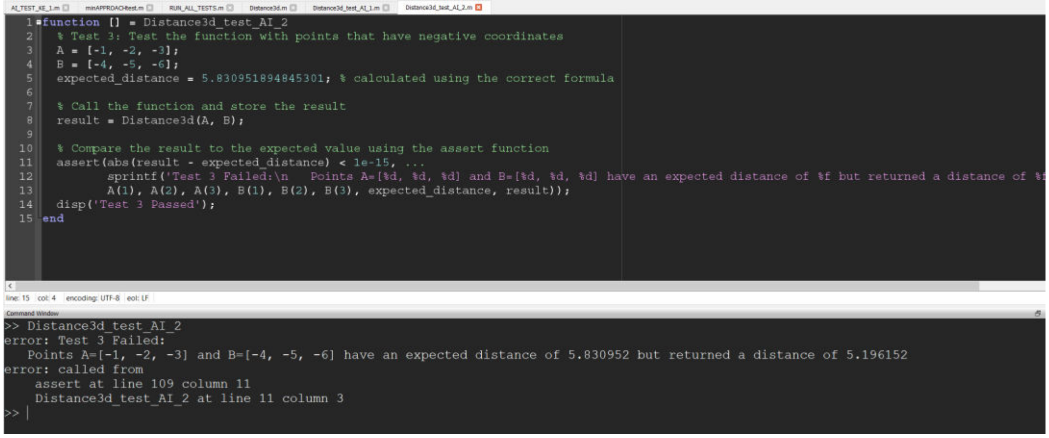
Adjust the error message in the test to add the values of the two points, the expected value, and the actual value

Here is an example of how you might adjust the error message in the test to include the values of the two points, the expected value, and the actual value:

```
% Test 3: Test the function with points that have negative coordinates
A = [-1, -2, -3];
B = [-4, -5, -6];
expected_distance = 5.830951894845301; % calculated using the ~~correct~~ formula
                                                                nope.
% Call the function and store the result
result = Distance3d(A, B);

% Compare the result to the expected value using the assert function
assert(abs(result - expected_distance) < 1e-15, ...
      sprintf('Test 3 Failed: Points A=[%d, %d, %d] and B=[%d, %d, %d] have an expected distance of %f but returned a distance of %f', ...
      A(1), A(2), A(3), B(1), B(2), B(3), expected_distance, result));
~~disp('Test 3 Passed');~~
```

In this example, the `assert` function checks that the absolute difference between the result and the expected distance is less than 1e-15. If this condition is not true, the `assert` function will throw an error and display a message that includes the values of the two points, the expected distance, and the actual distance returned by the function. This information can be helpful for debugging the problem and understanding why the test failed.

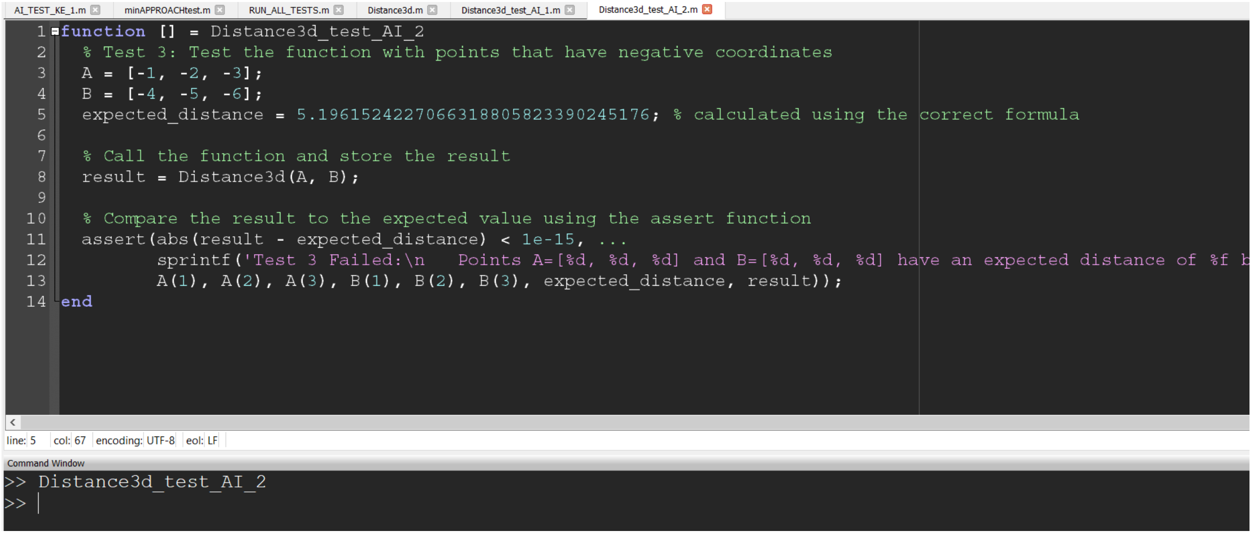
Manually fixing the formatting of the error message:



```
 1 function [] = Distance3d_test_AI_2
 2   % Test 3: Test the function with points that have negative coordinates
 3   A = [-1, -2, -3];
 4   B = [-4, -5, -6];
 5   expected_distance = 5.830951894845301; % calculated using the correct formula
 6
 7   % Call the function and store the result
 8   result = Distance3d(A, B);
 9
10   % Compare the result to the expected value using the assert function
11   assert(abs(result - expected_distance) < 1e-15, ...
12          sprintf('Test 3 Failed:\n   Points A=[%d, %d, %d] and B=[%d, %d, %d] have an expected distance of %f but returned a distance of %f
13          A(1), A(2), A(3), B(1), B(2), B(3), expected_distance, result));
14   disp('Test 3 Passed');
15 end
```

```
>> Distance3d_test_AI_2
error: Test 3 Failed:
   Points A=[-1, -2, -3] and B=[-4, -5, -6] have an expected distance of 5.830952 but returned a distance of 5.196152
error: called from
    assert at line 109 column 11
    Distance3d_test_AI_2 at line 11 column 3
>> |
```

Fixing the expected distance to the correct value results in the test passing:



```
function [] = Distance3d_test_AI_2
  % Test 3: Test the function with points that have negative coordinates
  A = [-1, -2, -3];
  B = [-4, -5, -6];
  expected_distance = 5.196152422706631880582339024517 6; % calculated using the correct formula

  % Call the function and store the result
  result = Distance3d(A, B);

  % Compare the result to the expected value using the assert function
  assert(abs(result - expected_distance) < 1e-15, ...
         sprintf('Test 3 Failed:\n   Points A=[%d, %d, %d] and B=[%d, %d, %d] have an expected distance of %f b
         A(1), A(2), A(3), B(1), B(2), B(3), expected_distance, result));
end
```

```
>> Distance3d_test_AI_2
>> 
```