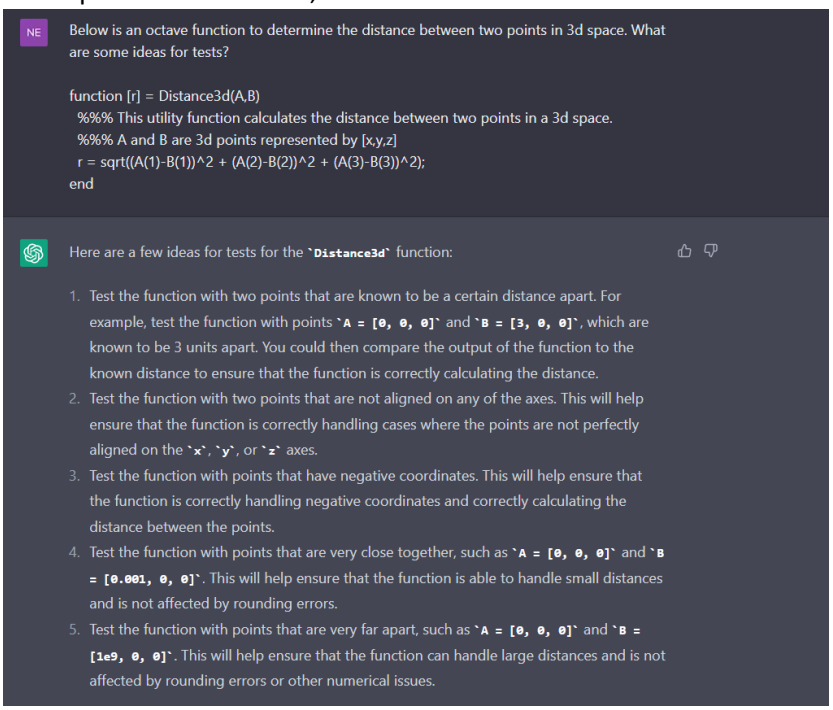
In the previous document, I asked ChatGPT for some test ideas:



NE

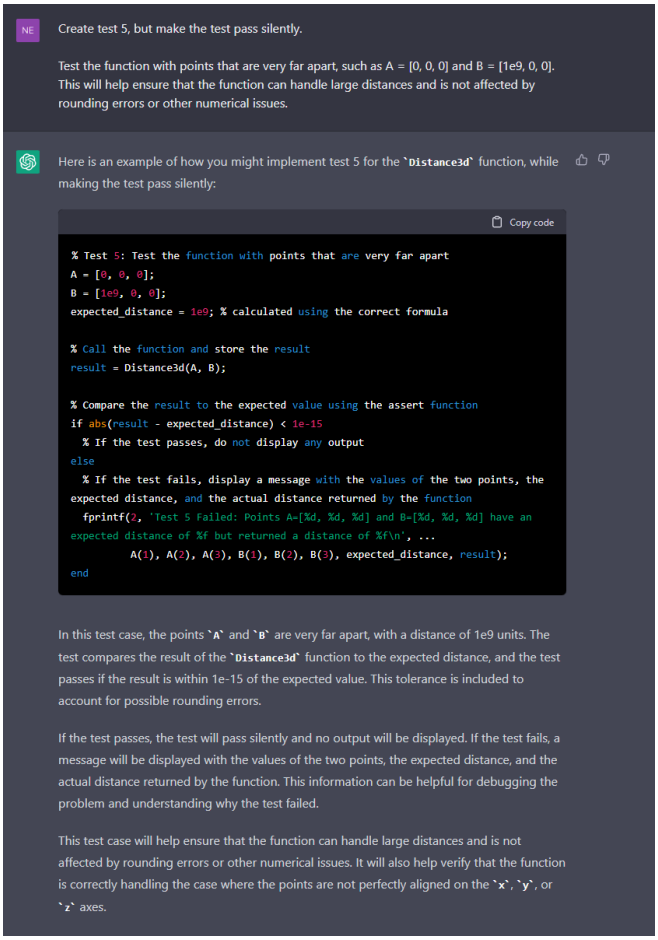
Below is an octave function to determine the distance between two points in 3d space. What are some ideas for tests?

```
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 #5, but to make it pass silently.



NE

Create test 5, but make the test pass silently.

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.

Here is an example of how you might implement test 5 for the `Distance3d` function, while making the test pass silently:

Copy code

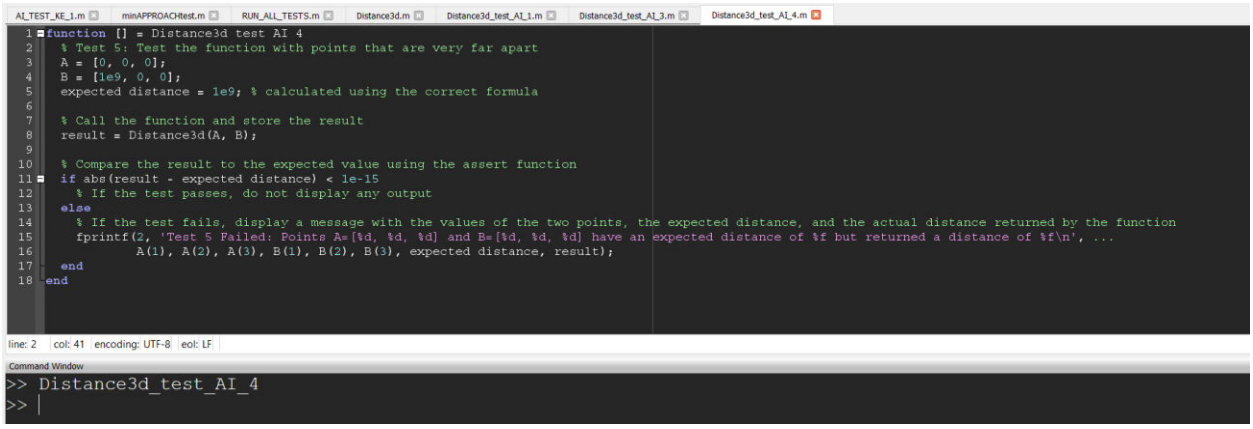
```
% Test 5: Test the function with points that are very far apart
A = [0, 0, 0];
B = [1e9, 0, 0];
expected_distance = 1e9; % calculated using the correct formula

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

% Compare the result to the expected value using the assert function
if abs(result - expected_distance) < 1e-15
  % If the test passes, do not display any output
else
  % If the test fails, display a message with the values of the two points, the
expected distance, and the actual distance returned by the function
  fprintf(2, 'Test 5 Failed: Points A=[%d, %d, %d] and B=[%d, %d, %d] have an
expected distance of %f but returned a distance of %f\n', ...
          A(1), A(2), A(3), B(1), B(2), B(3), expected_distance, result);
end
```

In this test case, the points `A` and `B` are very far apart, with a distance of 1e9 units. The test compares the result of the `Distance3d` function to the expected distance, 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.

If the test passes, the test will pass silently and no output will be displayed. If the test fails, a message will be displayed with 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.

This test case will help ensure that the function can handle large distances and is not affected by rounding errors or other numerical issues. It will also help verify that the function is correctly handling the case where the points are not perfectly aligned on the `x`, `y`, or `z` axes.

The test passes, but it doesn't use assert:



Asking ChatGpt to use assert:

NE

please use the assert function

Here is an example of how you might use the `assert` function to implement test 5 for the `Distance3d` function, while making the test pass silently:

Copy code

```
% Test 5: Test the function with points that are very far apart
A = [0, 0, 0];
B = [1e9, 0, 0];
expected_distance = 1e9; % 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
try
  assert(abs(result - expected_distance) < 1e-15);
catch
  % If the test fails, display a message with the values of the two points, the expected distance, and the actual distance returned by the function
  fprintf(2, 'Test 5 Failed: Points A=[%d, %d, %d] and B=[%d, %d, %d] have an expected distance of %f but returned a distance of %f\n', ...
          A(1), A(2), A(3), B(1), B(2), B(3), expected_distance, result);
end
```

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. The error is caught by the `catch` block, which displays a message with 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.

If the test passes, the `assert` function will not throw an error and the test will pass silently, with no output displayed. This allows you to make the test pass silently while still using the `assert` function to handle test failures.

Asking ChatGpt to not use try/catch:

NE

Don't use try and catch

Here is an example of how you might use the `assert` function to implement test 5 for the `Distance3d` function, while making the test pass silently and without using the `try` and `catch` statements:

Copy code

```
% Test 5: Test the function with points that are very far apart
A = [0, 0, 0];
B = [1e9, 0, 0];
expected_distance = 1e9; % 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, ...
       'Test 5 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);

% If the test passes, do not display any output
```

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 with 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.

If the test passes, the `assert` function will not throw an error and the test will pass silently, with no output displayed. This allows you to make the test pass silently while still using the `assert` function to handle test failures.

I hope this helps. Let me know if you have any further questions.

The test passes, although I again added a new line to the error message for readability.