In a previous chat thread with ChatGPT, I managed to ‘teach’ it what my structure was for my trajectory array, traj. I attempted to continue that thread to create some tests for the min_approach_distance function.

NE

Below is a different function called min_approach_distance . This function takes in a `traj` that describes two bodies , and returns various information about the moment of closest approach of those bodies. Write a test for this function:

To test the `min_approach_distance` function, we can define a struct with two time steps, representing the positions of two bodies at those times, and pass this struct to the `min_approach_distance` function. We can then verify that the output of the function is correct, by comparing it to the expected values.

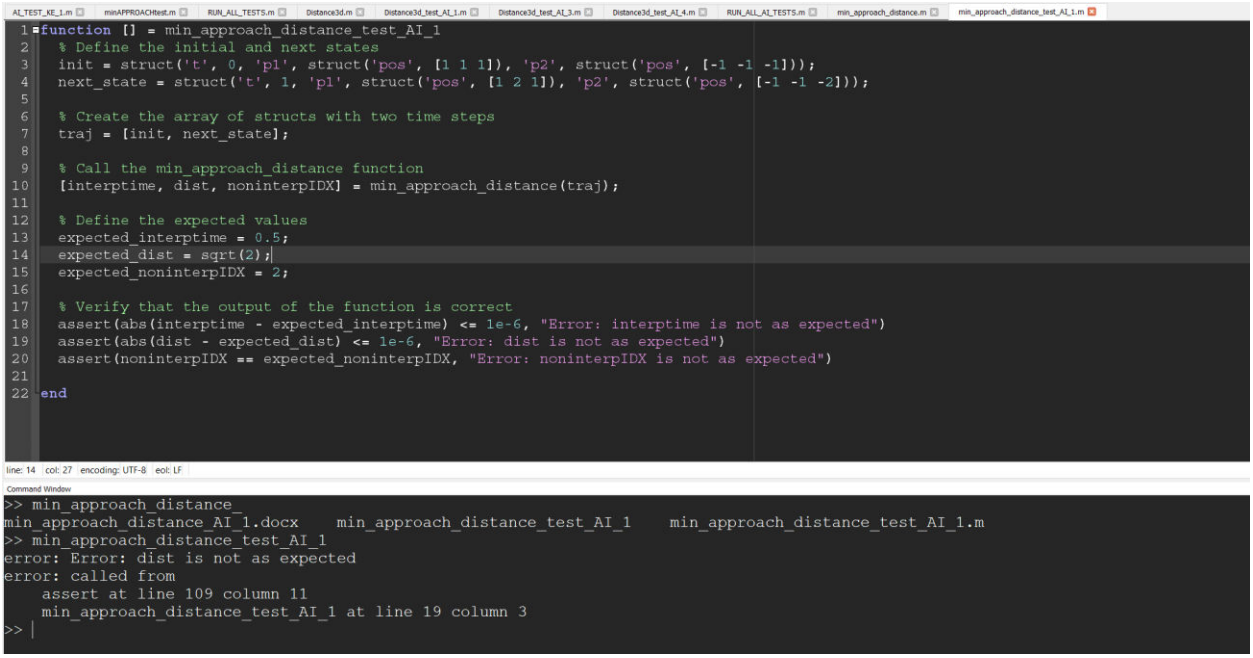
For example, we can define the following test:

Copy code

```
% Define the initial and next states
init = struct('t', 0, 'p1', struct('pos', [1 1 1]), 'p2', struct('pos', [-1 -1
-1]));
next_state = struct('t', 1, 'p1', struct('pos', [1 2 1]), 'p2', struct('pos',
[-1 -1 -2]));

% Create the array of structs with two time steps
traj = [init, next_state];

% Call the min_approach_distance function
[interptime, dist, noninterpIDX] = min_approach_distance(traj);

% Define the expected values
expected_interptime = 0.5;
expected_dist = sqrt(2);
expected_noninterpIDX = 2;

% Verify that the output of the function is correct
assert(abs(interptime - expected_interptime) <= 1e-6, "Error: interptime is not
as expected")
assert(abs(dist - expected_dist) <= 1e-6, "Error: dist is not as expected")
assert(noninterpIDX == expected_noninterpIDX, "Error: noninterpIDX is not as
expected")
```

This test will verify that the `min_approach_distance` function is correctly computing the time of minimum distance, the minimum distance, and the non-interpolated index of the minimum distance, based on the positions of the two bodies at two different times. The expected values are computed based on the positions of the bodies at each time step, and the `assert` function is used to test whether the output of the function is correct.

This is a very difficult task, and I don't expect the code to work immediately.



```
function [] = min_approach_distance_test_AI_1
  % Define the initial and next states
  init = struct('t', 0, 'p1', struct('pos', [1 1 1]), 'p2', struct('pos', [-1 -1 -1]));
  next_state = struct('t', 1, 'p1', struct('pos', [1 2 1]), 'p2', struct('pos', [-1 -1 -2]));

  % Create the array of structs with two time steps
  traj = [init, next_state];

  % Call the min_approach_distance function
  [interptime, dist, noninterpIDX] = min_approach_distance(traj);

  % Define the expected values
  expected_interptime = 0.5;
  expected_dist = sqrt(2);
  expected_noninterpIDX = 2;

  % Verify that the output of the function is correct
  assert(abs(interptime - expected_interptime) <= 1e-6, "Error: interptime is not as expected")
  assert(abs(dist - expected_dist) <= 1e-6, "Error: dist is not as expected")
  assert(noninterpIDX == expected_noninterpIDX, "Error: noninterpIDX is not as expected")

end
```

```
>> min_approach_distance_
min_approach_distance_AI_1.docx    min_approach_distance_test_AI_1    min_approach_distance_test_AI_1.m
>> min_approach_distance_test_AI_1
error: Error: dist is not as expected
error: called from
    assert at line 109 column 11
    min_approach_distance_test_AI_1 at line 19 column 3
>>
```

Sure enough, the test fails, but ChatGPT did make a very solid foundation to start off on.

It creates the following two states manually, without using my next function:

```
init =

  scalar structure containing the fields:

    t = 0
    p1 =

      scalar structure containing the fields:

        pos =

           1   1   1


    p2 =

      scalar structure containing the fields:

        pos =

          -1  -1  -1
```

```
next_state =

  scalar structure containing the fields:

    t = 1
    p1 =

      scalar structure containing the fields:

        pos =

           1   2   1


    p2 =

      scalar structure containing the fields:

        pos =

          -1  -1  -2
```

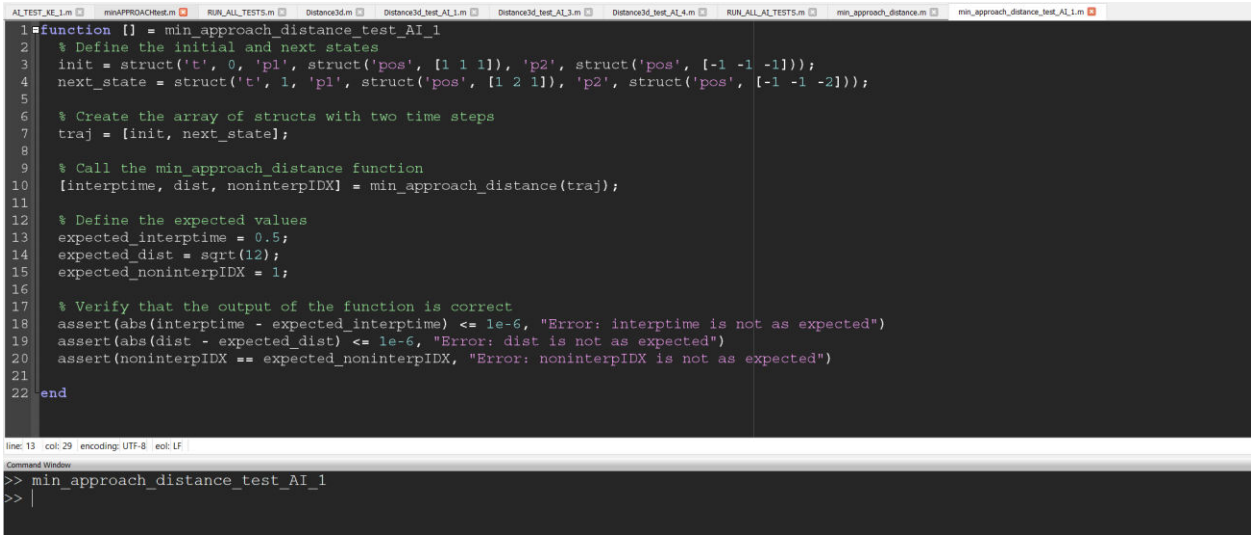
Note that it doesn't even add in a velocity for each body, as it knows that min_approach_distance doesn't use a velocity.

The reason the test fails is simple, it has the incorrect expected values. Note that the interptime is correct, the expected distance is close to accurate (sqrt(12) is correct, not sqrt(2)), and the noninterp index is entirely incorrect.

```
>> [interptime, dist, noninterpIDX] = min_approach_distance(traj)
interptime = 0.5000
dist = 3.4641
noninterpIDX = 1
```

```
% Define the expected values
expected_interptime = 0.5;
expected_dist = sqrt(2);
expected_noninterpIDX = 2;
```

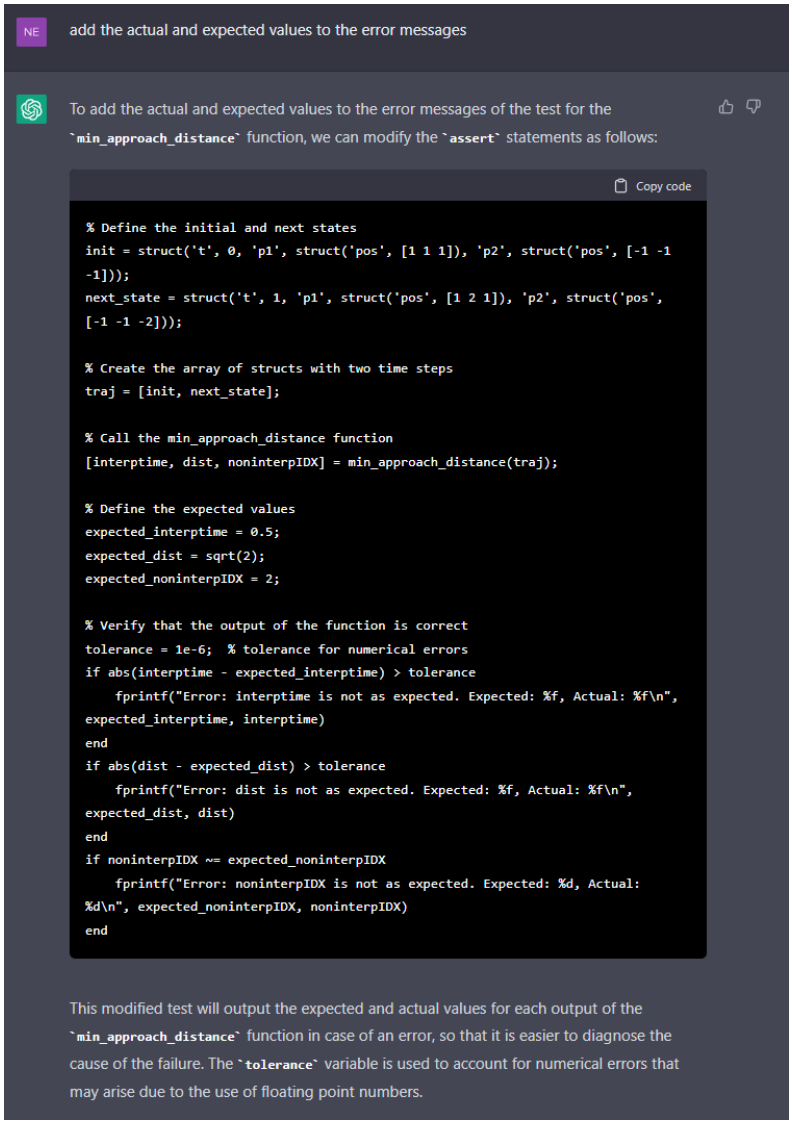
Fixing the code results in a passing test.



```
function [] = min_approach_distance_test_AI_1
  % Define the initial and next states
  init = struct('t', 0, 'p1', struct('pos', [1 1 1]), 'p2', struct('pos', [-1 -1 -1]));
  next_state = struct('t', 1, 'p1', struct('pos', [1 2 1]), 'p2', struct('pos', [-1 -1 -2]));

  % Create the array of structs with two time steps
  traj = [init, next_state];

  % Call the min_approach_distance function
  [interptime, dist, noninterpIDX] = min_approach_distance(traj);

  % Define the expected values
  expected_interptime = 0.5;
  expected_dist = sqrt(12);
  expected_noninterpIDX = 1;

  % Verify that the output of the function is correct
  assert(abs(interptime - expected_interptime) <= 1e-6, "Error: interptime is not as expected")
  assert(abs(dist - expected_dist) <= 1e-6, "Error: dist is not as expected")
  assert(noninterpIDX == expected_noninterpIDX, "Error: noninterpIDX is not as expected")

end
```

```
>> min_approach_distance_test_AI_1
>>
```

As seen in previous examples, I can ask ChatGPT to add the expected and actual values to the error messages:



NE add the actual and expected values to the error messages

To add the actual and expected values to the error messages of the test for the `min_approach_distance` function, we can modify the `assert` statements as follows:

Copy code

```
% Define the initial and next states
init = struct('t', 0, 'p1', struct('pos', [1 1 1]), 'p2', struct('pos', [-1 -1
-1]));
next_state = struct('t', 1, 'p1', struct('pos', [1 2 1]), 'p2', struct('pos',
[-1 -1 -2]));

% Create the array of structs with two time steps
traj = [init, next_state];

% Call the min_approach_distance function
[interptime, dist, noninterpIDX] = min_approach_distance(traj);

% Define the expected values
expected_interptime = 0.5;
expected_dist = sqrt(2);
expected_noninterpIDX = 2;

% Verify that the output of the function is correct
tolerance = 1e-6;  % tolerance for numerical errors
if abs(interptime - expected_interptime) > tolerance
    fprintf("Error: interptime is not as expected. Expected: %f, Actual: %f\n",
expected_interptime, interptime)
end
if abs(dist - expected_dist) > tolerance
    fprintf("Error: dist is not as expected. Expected: %f, Actual: %f\n",
expected_dist, dist)
end
if noninterpIDX ~= expected_noninterpIDX
    fprintf("Error: noninterpIDX is not as expected. Expected: %d, Actual:
%d\n", expected_noninterpIDX, noninterpIDX)
end
```

This modified test will output the expected and actual values for each output of the `min_approach_distance` function in case of an error, so that it is easier to diagnose the cause of the failure. The `tolerance` variable is used to account for numerical errors that may arise due to the use of floating point numbers.

The test still passes after changing just the following lines:

```
17  % Verify that the output of the function is correct
18  tolerance = 1e-6;  % tolerance for numerical errors
19  if abs(interptime - expected_interptime) > tolerance
20      fprintf("Error: interptime is not as expected. Expected: %f, Actual: %f\n", expected_interptime, interptime)
21  end
22  if abs(dist - expected_dist) > tolerance
23      fprintf("Error: dist is not as expected. Expected: %f, Actual: %f\n", expected_dist, dist)
24  end
25  if noninterpIDX ~= expected_noninterpIDX
26      fprintf("Error: noninterpIDX is not as expected. Expected: %d, Actual: %d\n", expected_noninterpIDX, noninterpIDX)
27  end
28
29 end
```

line: 14 col: 28 encoding: UTF-8 eol: LF

Command Window

```
>> min_approach_distance_test_AI_1
>>
```

Asking ChatGPT to use assert instead of fprintf:

NE use assert instead of fprintf

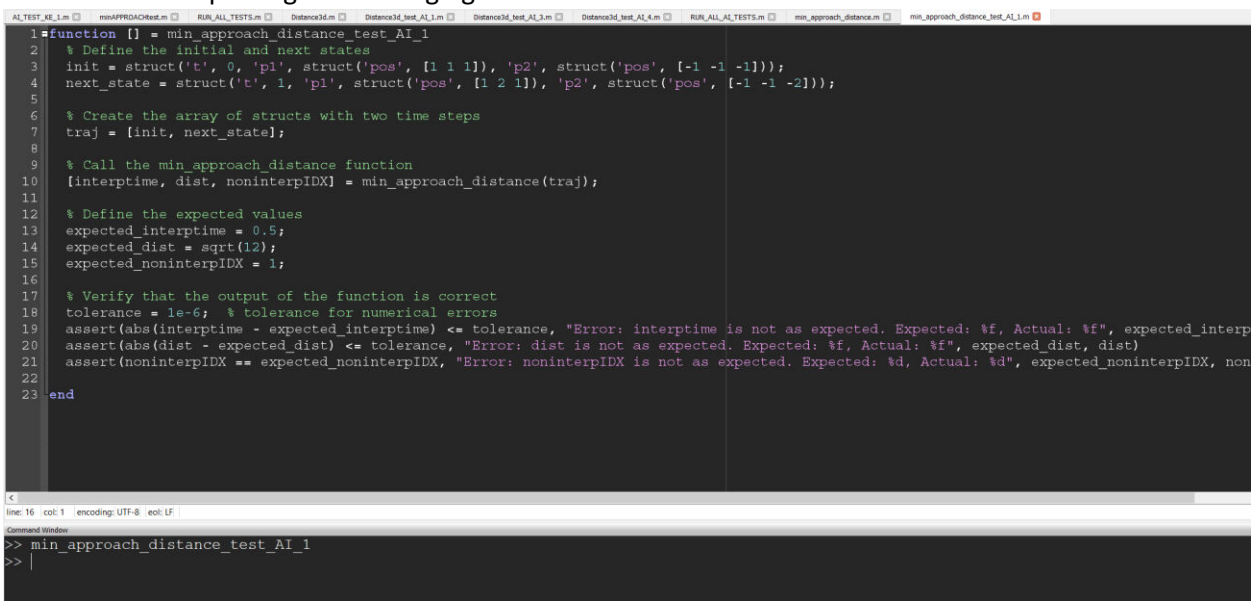
To use the `assert` function instead of `fprintf` to print the expected and actual values in case of an error, we can modify the test for the `min_approach_distance` function as follows:

Copy code

```
% Define the initial and next states
init = struct('t', 0, 'p1', struct('pos', [1 1 1]), 'p2', struct('pos', [-1 -1
-1]));
next_state = struct('t', 1, 'p1', struct('pos', [1 2 1]), 'p2', struct('pos',
[-1 -1 -2]));

% Create the array of structs with two time steps
traj = [init, next_state];

% Call the min_approach_distance function
[interptime, dist, noninterpIDX] = min_approach_distance(traj);

% Define the expected values
expected_interptime = 0.5;
expected_dist = sqrt(2);
expected_noninterpIDX = 2;

% Verify that the output of the function is correct
tolerance = 1e-6;  % tolerance for numerical errors
assert(abs(interptime - expected_interptime) <= tolerance, "Error: interptime is
not as expected. Expected: %f, Actual: %f", expected_interptime, interptime)
assert(abs(dist - expected_dist) <= tolerance, "Error: dist is not as expected.
Expected: %f, Actual: %f", expected_dist, dist)
assert(noninterpIDX == expected_noninterpIDX, "Error: noninterpIDX is not as
expected. Expected: %d, Actual: %d", expected_noninterpIDX, noninterpIDX)
```

This modified test will use the `assert` function to print the expected and actual values in case of an error, and will also include the tolerance for numerical errors. The `assert` function will raise an error and terminate the program if the condition it tests is not met.

The test remains passing after changing the correct lines.



```
function [] = min_approach_distance_test_AI_1
  % Define the initial and next states
  init = struct('t', 0, 'p1', struct('pos', [1 1 1]), 'p2', struct('pos', [-1 -1 -1]));
  next_state = struct('t', 1, 'p1', struct('pos', [1 2 1]), 'p2', struct('pos', [-1 -1 -2]));

  % Create the array of structs with two time steps
  traj = [init, next_state];

  % Call the min_approach_distance function
  [interptime, dist, noninterpIDX] = min_approach_distance(traj);

  % Define the expected values
  expected_interptime = 0.5;
  expected_dist = sqrt(12);
  expected_noninterpIDX = 1;

  % Verify that the output of the function is correct
  tolerance = 1e-6;  % tolerance for numerical errors
  assert(abs(interptime - expected_interptime) <= tolerance, "Error: interptime is not as expected. Expected: %f, Actual: %f", expected_interp
  assert(abs(dist - expected_dist) <= tolerance, "Error: dist is not as expected. Expected: %f, Actual: %f", expected_dist, dist)
  assert(noninterpIDX == expected_noninterpIDX, "Error: noninterpIDX is not as expected. Expected: %d, Actual: %d", expected_noninterpIDX, non

end
```

```
>> min_approach_distance_test_AI_1
>>
```

Overall, although ChatGPT made mistakes with the expected values and inaccuracies in the error messages and its usage of fprintf instead of assert, it is still extremely impressive that it was able to create such a useful amount of text. Overall, I would say 90-95% of the code it provided was correct, which is an extremely impressive figure given the difficulty of the problem I presented it with.