Michael Cowan

Computational Physics

Problem Set 1

Chapter 1

12. In this problem we modify the given **orthog** program to accept input vectors of any length. I accomplished this by, instead of summing the products of the individual elements within a loop, using the fact that performing a dot product is the same as multiplying a row matrix by a column matrix. The second input matrix has to be transposed to a column matrix because it is inputted as a row matrix. I also had the program compare the dimensions of the two input matrices prior to transposition and multiplication to ensure that we were not trying to multiply vectors of different dimensions.

I tested the program with a few different inputs in order to make sure it handled all of the different cases correctly: first, vectors of different dimensions; second, vectors that were orthogonal; third, vectors that were not orthogonal. All three cases gave the expected results.

Code (italicized lines were copied directly from the given **orthog.m** file):

1. *a = input('Enter the first vector: '); %input the values of the two vectors*
2. *b = input('Enter the second vector: ');*
3. da=length(a); %find the number of elements in the two vectors
4. db=length(b);
5. if(da==db) %first case: the vectors are the same size
6. a\_dot\_b=a\*b'; %find the dot product by multiplying a by the transpose of b
7. *if( a\_dot\_b == 0 ) %check for orthogonality*
8. *disp('Vectors are orthogonal');*
9. *else*
10. *disp('Vectors are NOT orthogonal');*
11. *fprintf('Dot product = %g \n',a\_dot\_b); %give the value of the dot product if a and b are not orthogonal*
12. *end*
13. else %second case: the vectors are not the same size
14. disp('Vectors are different sizes');
15. end

Output:

Case: input vectors are of different dimensions

Enter the first vector: [1,2]

Enter the second vector: [1,2,3]

Vectors are different sizes

Case: input vectors are orthogonal

Enter the first vector: [3,2,0]

Enter the second vector: [0,0,-2]

Vectors are orthogonal

Case: input vectors are not orthogonal

Enter the first vector: [2,1,-4,3]

Enter the second vector: [0,-1,5,2]

Vectors are NOT orthogonal

Dot product = -15

13. In this problem, we write a program that accepts the input of two three-dimensional vectors and then finds a unit vector that is orthogonal to the first two. Since a cross product is orthogonal to the original two vectors, I found the elements of the cross product of the two input vectors and then divided each one by the total magnitude. I also included several if-else statements to eliminate some special cases: vectors that were not three dimensional, vectors of magnitude 0, and vectors that are parallel (or anti-parallel).

Code (the only part of the given program that was used was the two input lines (in italics)):

1. *a = input('Enter the first vector: '); %input the values of the two vectors*
2. *b = input('Enter the second vector: ');*
3. da=length(a); %find the number of elements in the two vectors
4. db=length(b);
5. if(da==3 && db==3) %first case: both vectors are three dimensional
6. if(sqrt(a(1)^2+a(2)^2+a(3)^2)==0 || sqrt(b(1)^2+b(2)^2+b(3)^2)==0) %check for a 0 vector
7. disp('The magnitude of one or both vectors is 0');
8. else
9. c1=a(2)\*b(3)-a(3)\*b(2); %calculate the elements of vector c
10. c2=a(3)\*b(1)-a(1)\*b(3);
11. c3=a(1)\*b(2)-a(2)\*b(1);
12. magc=sqrt(c1^2+c2^2+c3^2); %calculate the magnitude of c
13. if(magc==0) %if the vectors are parallel the cross product is 0
14. disp('Vectors are parallel so orthogonal unit vector can not be calculated');
15. else
16. c=[c1/magc,c2/magc,c3/magc]; %construct the unit vector
17. disp('Vector c is a unit vector orthogonal to both a and b'); %give the value of c
18. disp('c=');
19. disp(c);
20. end
21. end
22. else %second case: one or both of the vectors are not three dimensional
23. disp('At least one of the vectors is not three dimensional');
24. end

Output:

Case: one vector is 0 length

Enter the first vector: [2,3,-1]

Enter the second vector: [0,0,0]

The magnitude of one or both vectors is 0

Case: vectors are parallel

Enter the first vector: [1,2,3]

Enter the second vector: [2,4,6]

Vectors are parallel so orthogonal unit vector can not be calculated

Case: one vector is not three dimensional

Enter the first vector: [2,3]

Enter the second vector: [1,3,4]

At least one of the vectors is not three dimensional

Case: all the criteria are met and the orthogonal unit vector is calculated

Enter the first vector: [1,2,-1]

Enter the second vector: [0,3,1]

Vector c is a unit vector orthogonal to both a and b

c=

0.84515 -0.16903 0.50709

14. We again modify the given **orthog** program so that if the second vector is not orthogonal to the first, we calculate a new vector that is orthogonal to the first, has the same magnitude as the second, and is in the same plane as both. I used the Gram-Schmidt procedure given in class on Jan. 26

Code (italicized lines were copied directly from the given **orthog.m** file):

1. *a = input('Enter the first vector: ');*
2. *b = input('Enter the second vector: ');*
3. da=length(a); %find the number of elements in the two vectors
4. db=length(b);
5. if(da==3 && db==3) %check that both vectors are three dimensional
6. maga=sqrt(a(1)^2+a(2)^2+a(3)^2);
7. magb=sqrt(b(1)^2+b(2)^2+b(3)^2);
8. if(maga==0 || magb==0) %check for a 0 vector
9. disp('The magnitude of one or both vectors is 0');
10. else
11. *%\* Evaluate the dot product as sum over products of elements*
12. *a\_dot\_b = 0;*
13. *for i=1:3*
14. *a\_dot\_b = a\_dot\_b + a(i)\*b(i);*
15. *end*
16. *if( a\_dot\_b == 0 ) %check for orthogonality*
17. *disp('Vectors are orthogonal');*
18. elseif(a(1)/b(1)==a(2)/b(2) && a(2)/b(2)==a(3)/b(3) && a(1)/b(1)==a(3)/b(3)) %check that the vectors are not parallel
19. disp('Vectors are parallel so orthogonal vector can not calculated');
20. else
21. u1=a/maga; %calculate the first unit vector
22. v2=b-(b\*u1')\*u1; %calculate the second unit vector, orthogonal to the first
23. magv2=sqrt(v2(1)^2+v2(2)^2+v2(3)^2);
24. u2=v2/magv2;
25. c=u2\*magb; %result points in the direction of the second unit vector and has the same magnitude as b
26. disp('Vector c is a vector orthogonal to and in the same plane as a that has the same magnitude as b');
27. %give the value of c
28. disp('c=');
29. disp(c);
30. end
31. end
32. else %one or both of the vectors is not three dimensional
33. disp('At least one of the vectors is not three dimensional');
34. end

Output:

Case: one vector is 0 length

Enter the first vector: [2,5,-2]

Enter the second vector: [0,0,0]

The magnitude of one or both vectors is 0

Case: vectors are parallel

Enter the first vector: [2,3,4]

Enter the second vector: [-4,-6,-8]

Vectors are parallel so orthogonal vector can not calculated

Case: vectors are already orthogonal

Enter the first vector: [0,0,4]

Enter the second vector: [3,2,0]

Vectors are orthogonal

Case: one vector is not three dimensional

Enter the first vector: [2,3]

Enter the second vector: [1,4,7]

At least one of the vectors is not three dimensional

Case: all the criteria are met and the orthogonal vector is calculated

Enter the first vector: [3,0,0]

Enter the second vector: [4,3,0]

Vector c is a vector orthogonal to and in the same plane as a that has the same magnitude as b

c=

0 5 0

15. In this program, we test the ability of the Langrange polynomial interpolation algorithm to approximate the values of the zeroth-order Bessel function. The **interp** program calls the **intrpf** function, and the output is a plot of the interpolated polynomial. We can then read the values of the graph to find the approximate values of the Bessel function and compare them to the known values given in a table.

Code (interp, followed by the intrpf function; both codes come directly from the author’s website except the part I added (in italics)):

1. % interp - Program to interpolate data using Lagrange
2. % polynomial to fit quadratic to three data points
3. clear all; help interp; % Clear memory and print header
4. %\* Initialize the data points to be fit by quadratic
5. disp('Enter data points as x,y pairs (e.g., [1 2])');
6. for i=1:3
7. temp = input('Enter data point: ');
8. x(i) = temp(1);
9. y(i) = temp(2);
10. end
11. %\* Establish the range of interpolation (from x\_min to x\_max)
12. xr = input('Enter range of x values as [x\_min x\_max]: ');
13. %\* Find yi for the desired interpolation values xi using
14. % the function intrpf
15. nplot = 100; % Number of points for interpolation curve
16. for i=1:nplot
17. xi(i) = xr(1) + (xr(2)-xr(1))\*(i-1)/(nplot-1);
18. yi(i) = intrpf(xi(i),x,y); % Use intrpf function to interpolate
19. end
20. %\* Plot the curve given by (xi,yi) and mark original data points
21. plot(x,y,'\*',xi,yi,'-');
22. xlabel('x');
23. ylabel('y');
24. title('Three point interpolation');
25. legend('Data points','Interpolation ');
26. *disp('J0(0.3)=');*
27. *disp(intrpf(0.3,x,y));*
28. *disp('J0(0.9)=');*
29. *disp(intrpf(0.9,x,y));*
30. *disp('J0(1.1)=');*
31. *disp(intrpf(1.1,x,y));*
32. *disp('J0(1.5)=');*
33. *disp(intrpf(1.5,x,y));*
34. *disp('J0(1.9)=');*
35. *disp(intrpf(1.9,x,y));*
36. *disp('J0(2.0)=');*
37. *disp(intrpf(2.0,x,y));*
38. function yi = intrpf(xi,x,y)
39. % Function to interpolate between data points
40. % using Lagrange polynomial (quadratic)
41. % Inputs
42. % x Vector of x coordinates of data points (3 values)
43. % y Vector of y coordinates of data points (3 values)
44. % xi The x value where interpolation is computed
45. % Output
46. % yi The interpolation polynomial evaluated at xi
47. %\* Calculate yi = p(xi) using Lagrange polynomial
48. yi = (xi-x(2))\*(xi-x(3))/((x(1)-x(2))\*(x(1)-x(3)))\*y(1) ...
49. + (xi-x(1))\*(xi-x(3))/((x(2)-x(1))\*(x(2)-x(3)))\*y(2) ...
50. + (xi-x(1))\*(xi-x(2))/((x(3)-x(1))\*(x(3)-x(2)))\*y(3);
51. return;

Output:

Enter data points as x,y pairs (e.g., [1 2])

Enter data point: [0,1.0]

Enter data point: [0.5,0.9385]

Enter data point: [1.0,0.7652]

Enter range of x values as [x\_min x\_max]: [0,2.0]

J0(0.3)=

0.97652

J0(0.9)=

0.80880

J0(1.1)=

0.71712

J0(1.5)=

0.48010

J0(2.0)=

0.083200

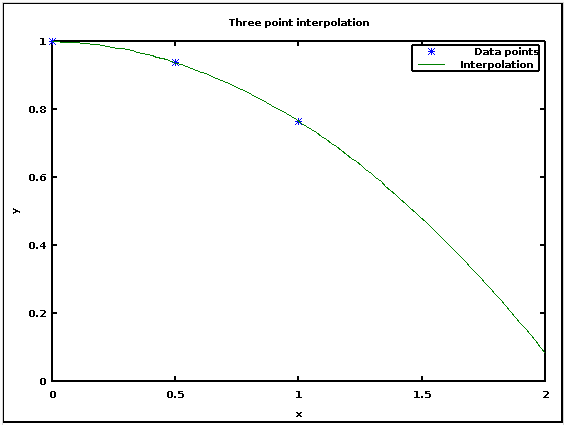


Figure 1: plot of the approximate values of the zeroth-order Bessel function, found using the Lagrange polynomial algorithm

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| x | 0.3 | 0.9 | 1.1 | 1.5 | 2.0 |
| J0(x) (program) | 0.97652 | 0.80880 | 0.71712 | 0.48010 | 0.083200 |
| J0(x) (table) | 0.9776 | 0.8075 | 0.7196 | 0.5118 | 0.2239 |

Table of values from Schaum’s Outlines: Mathematical Handbook of Formulas and Tables, 4th Ed.

Since the Bessel function is not a parabolic relationship, the calculated values get further and further away from the actual values as x increases.