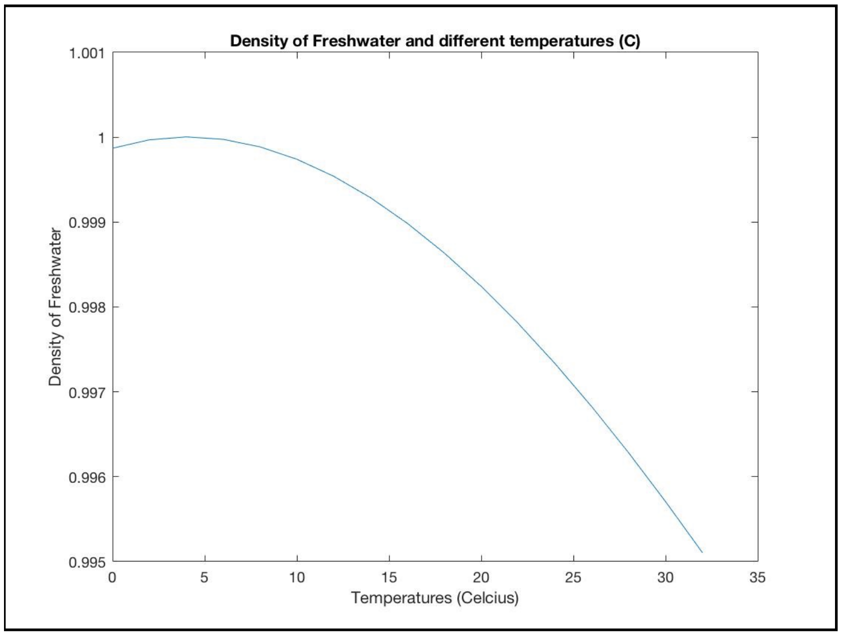
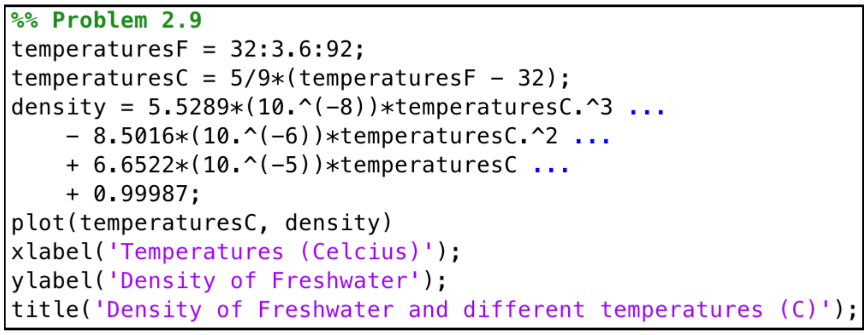
Mihir Trivedi
Math320
September 8, 2016

Problem 2.9

- Description of Code:
  - The code first creates a vector of Fahrenheit temperatures. Then, it applies the Celsius transformation formula to the vector, ensuring that each element of the Fahrenheit vector is converted into degrees Celsius. By similarly performing the density transformation on the Celsius vector, the code products a vector `density` which contains the density of water at each Celsius temperature.
- Results:



- Code:

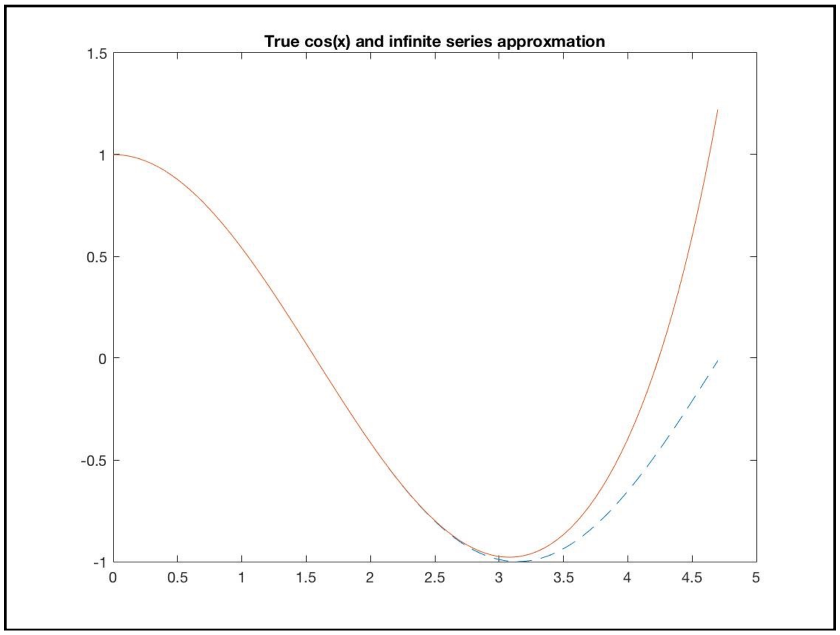


```
%% Problem 2.9
temperaturesF = 32:3.6:92;
temperaturesC = 5/9*(temperaturesF - 32);
density = 5.5289*(10.^(-8))*temperaturesC.^3 ...
    - 8.5016*(10.^(-6))*temperaturesC.^2 ...
    + 6.6522*(10.^(-5))*temperaturesC ...
    + 0.99987;
plot(temperaturesC, density)
xlabel('Temperatures (Celcius)');
ylabel('Density of Freshwater');
title('Density of Freshwater and different temperatures (C)');
```

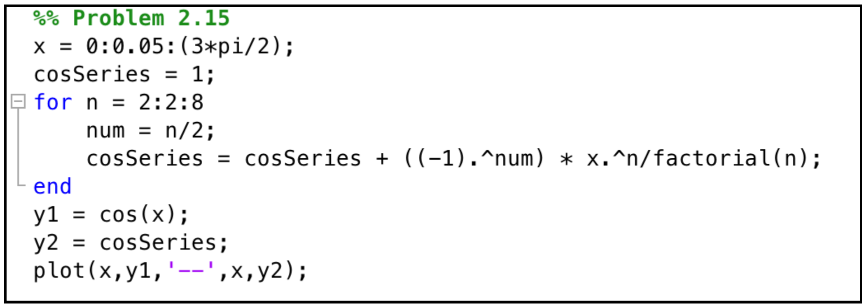
Mihir Trivedi
Math320
September 8, 2016

Problem 2.15

- Description of Code:
    - The code first creates a vector ranging from 0 to 3*pi/2 using an increment of 0.05. It then initializes the variable `cosSeries` to 1, and then iteratively adds onto this variable by applying the approximate cosine formula provided for n = 2, 4, 6, 8. The code concludes by plotting the two lines; the true cosine graph is dotted, whereas the approximation is solid.
- Results:



- Code:



```
%% Problem 2.15
x = 0:0.05:(3*pi/2);
cosSeries = 1;
for n = 2:2:8
    num = n/2;
    cosSeries = cosSeries + ((-1).^num) * x.^n/factorial(n);
end
y1 = cos(x);
y2 = cosSeries;
plot(x,y1,'--',x,y2);
```

Mihir Trivedi
Math320
September 8, 2016

Problem 3.16:

- Description of Code:
    - The code begins by initializing a 9x2 matrix. Each row represents a unique (x,y)-coordinate to test the function `polarCoords`. The code then iterates throw the rows of this matrix, extracts the (x,y)-coordinates, and applies the function `polarCoords` to each one. The output is then recorded in `results` (also 9x2), which is initialized with 0s. The function `polarCoords` takes as an input the (x,y)-coordinate and outputs the (r, theta)-polar coordinates. It immediately calculates `r` according to the formula provided. The code then iterates through several conditional statements based on the values of x and y, setting theta to a unique formula within each one. If none of the conditions pass, then the (x,y)-coordinate is located in the first or fourth quadrant, and `theta` is calculated according to the formula provided.
- Results:

x	**y**	**r**	**theta**
2	0	2	0
2	1	2.236067977	0.463647609
0	3	3	1.570796327
-3	1	3.16227766	2.819842099
-2	0	2	3.141592654
-1	-2	2.236067977	-2.034443936
0	0	0	0
0	-2	2	-1.570796327
2	2	2.828427125	0.785398163

- Code:

```
%% Problem 3.16
values = [2 0;2 1;0 3;-3 1;-2 0;-1 -2;0 0;0 -2;2 2];
[rows, columns] = size(values);
results = zeros(9,2);
for row = 1:rows
    x = values(row,1);
    y = values(row,2);
    [r,theta] = polarCoords(x,y);
    results(row,1) = r;
    results(row,2) = theta;
end
```

```
function [r,theta] = polarCoords(x,y)
r = sqrt(x.^2 + y.^2);
if(x<0 && y>0)
    theta = atan(y/x)+pi;
elseif(x<0 && y<0)
    theta = atan(y/x)-pi;
elseif(x<0 && y==0)
    theta=pi;
elseif(x==0 && y>0)
    theta=pi/2;
elseif(x==0 && y<0)
    theta=(-1)*pi/2;
elseif(x==0 && y==0)
    theta=0;
else
    theta=atan(y/x);
end
end
```

Mihir Trivedi
Math320
September 8, 2016

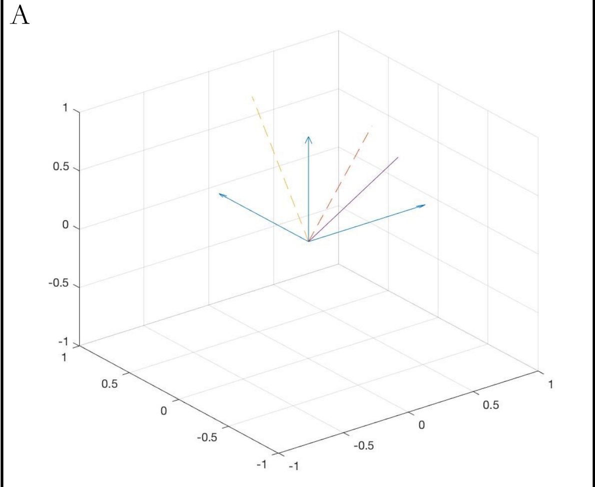
Problem 3.20

- Description of Code:
    - The code initializes a matrix of vectors. The vectors `a` and `b` are represented as a 1x6 matrix, with the first three entries denoting `a`, and the second three entries denoting `b`. These vectors will be the test inputs for `crossProduct`. The function `crossProduct` takes as inputs two unit vectors, and outputs the number of radians between the two vectors, the vector orthogonal to `a` and `b`, and the magnitude of the cross product. Then, by manipulating the dot-product formula provided, the code computes `theta`; this represents the angle in between the two vectors. The code then calculates the magnitude of vectors a and b using the in-built function `norm`. Then, using the cross-product formula, the code computes the magnitude of the cross-product of vectors `a` and `b`. In order to geometric, unit-vector representation of the cross product, the function calls the built-in function `cross`[1]. The orthogonal vector is saved as `ortho`. Then, in order to generate the plot, the code uses the function `quiver3`[2], which draws an arrow between the origin and the points specified by the `a`, `b`, and `ortho`. After plotting all three vectors, ensuring that the `a` and `b` are dotted and `ortho` remain solid, the code limits the x, y, and z axis to [-1, 1]. Before iterating through the matrix of vectors, the code creates two matrix, one to store the numeric results, and one to store the orthogonal vectors. The code then iterates through the rows, applying `crossProduct` to each row.
- Results (first row corresponds to input A, second row to input B, and so on):

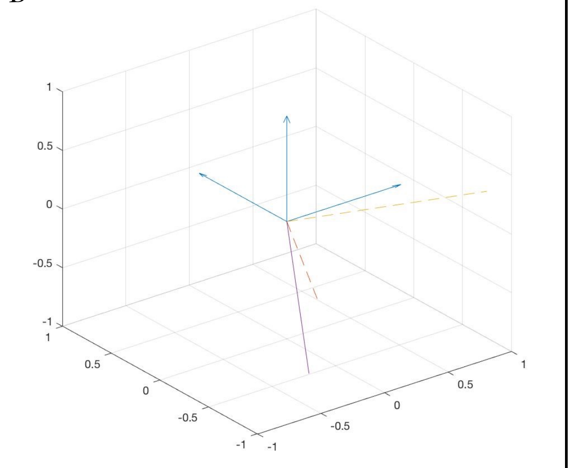
Theta	**Magnitude**
0 + 4.4772*i	0 + 2463.4*i
1.570796327	35.6931366
1.570796327	18
1.570796327	1

O(1)	**O(2)**	**O(3)**
4	-20	28
-16	-27	-17
6	12	12
0	0	1

A



B

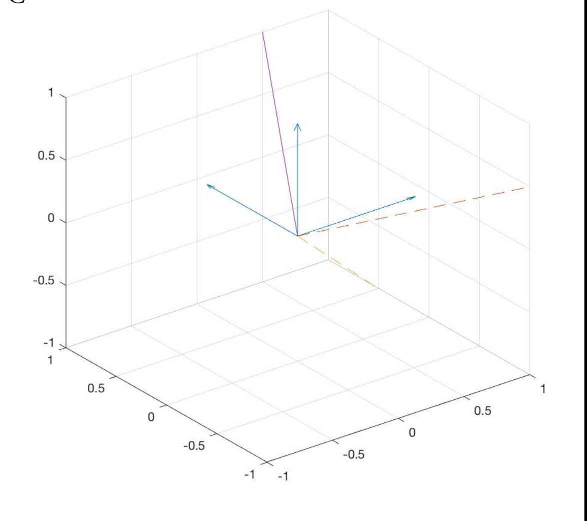


[1] See: http://www.mathworks.com/help/matlab/ref/cross.html

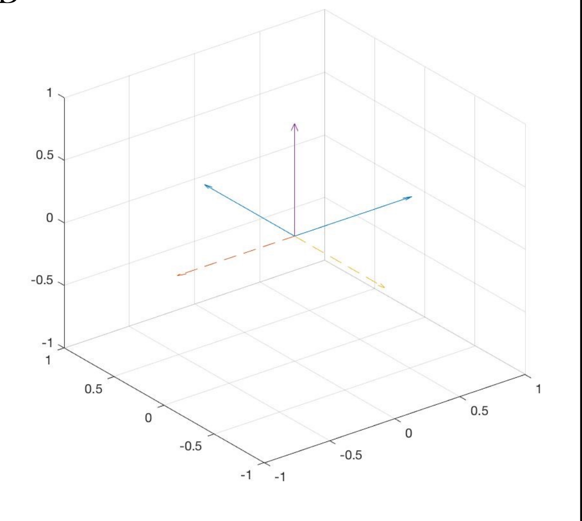
[2] See: http://www.mathworks.com/help/matlab/ref/quiver3.html

Mihir Trivedi
Math320
September 8, 2016

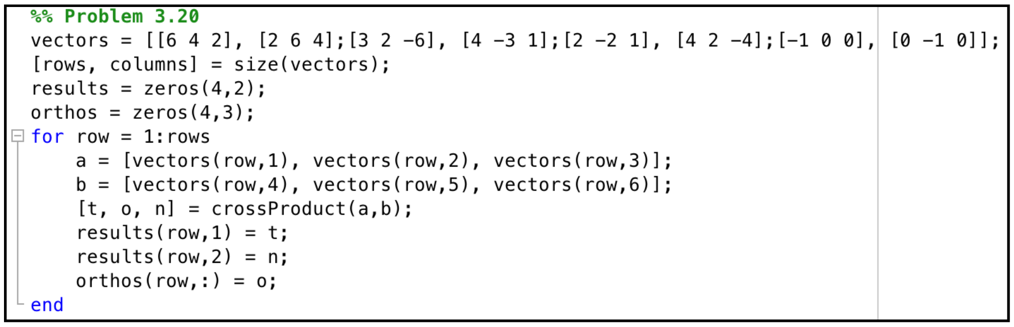
C



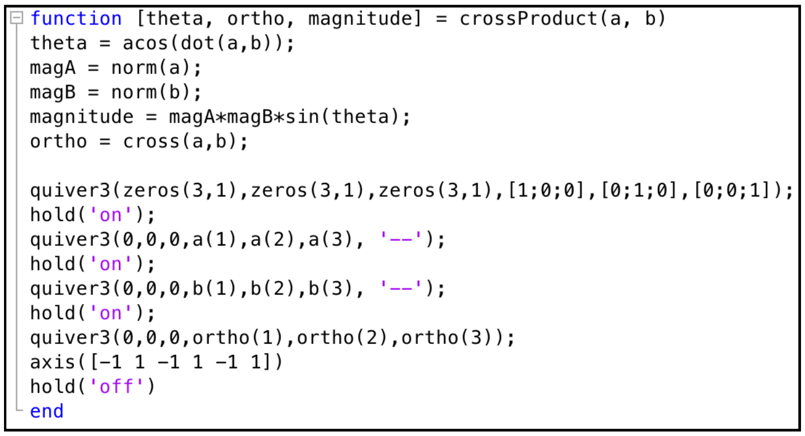
D



- Code:



```
%% Problem 3.20
vectors = [[6 4 2], [2 6 4];[3 2 -6], [4 -3 1];[2 -2 1], [4 2 -4];[-1 0 0], [0 -1 0]];
[rows, columns] = size(vectors);
results = zeros(4,2);
orthos = zeros(4,3);
for row = 1:rows
    a = [vectors(row,1), vectors(row,2), vectors(row,3)];
    b = [vectors(row,4), vectors(row,5), vectors(row,6)];
    [t, o, n] = crossProduct(a,b);
    results(row,1) = t;
    results(row,2) = n;
    orthos(row,:) = o;
end
```



```
function [theta, ortho, magnitude] = crossProduct(a, b)
theta = acos(dot(a,b));
magA = norm(a);
magB = norm(b);
magnitude = magA*magB*sin(theta);
ortho = cross(a,b);

quiver3(zeros(3,1),zeros(3,1),zeros(3,1),[1;0;0],[0;1;0],[0;0;1]);
hold('on');
quiver3(0,0,0,a(1),a(2),a(3), '--');
hold('on');
quiver3(0,0,0,b(1),b(2),b(3), '--');
hold('on');
quiver3(0,0,0,ortho(1),ortho(2),ortho(3));
axis([-1 1 -1 1 -1 1])
hold('off')
end
```