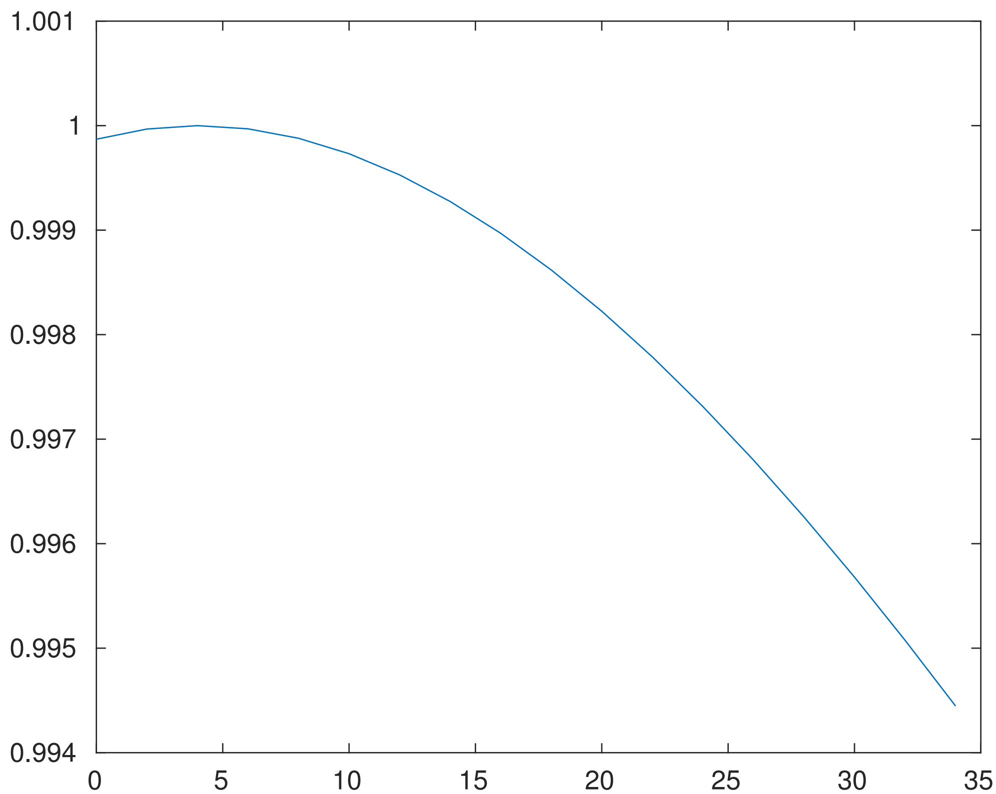
MATH 320: HOMEWORK 1 SOLUTIONS

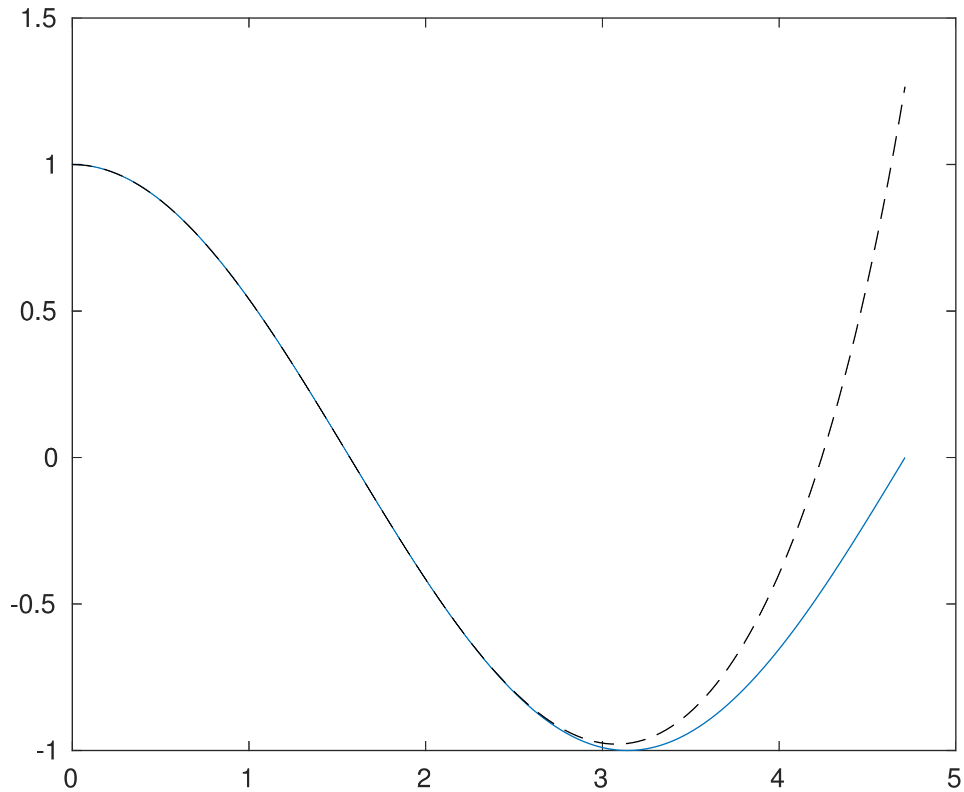
(1) The following code defines a function `rho` which inputs the temperature and outputs the density. We use `arrayfun` to apply it to a range of temperatures. Then we plot the result.

```
rho = (@(TC) 5.5289e-8 * TC^3 - 8.5016e-6 * TC^2 + 6.5622e-5*TC + .99987)
tempsF = 32:3.6:93.2;                  %define the temperature vector
tempsC = (5/9)*(tempsF - 32);          %convert to Celsius.
densities = arrayfun(@(t) rho(t), tempsC);
plot(tempsC,densities)
```



(2) First we define the x-range X. Then we compute both the cosine function, and the approximation as functions of the array X. Then we plot both together. The option '–' means the second plot will be a dashed line, while 'k' means it will be black.

```
X = 0:pi/100:3*pi/2;
Y1 = cos(X);
Y2 = 1 - X.^2/factorial(2) + X.^4/factorial(4) ...
    - X.^6/factorial(6) + X.^8/factorial(8);
plot(X,Y1,X,Y2,'--k')
```



(3) First we set up the MATLAB function PolarForm with all `if ... elseif` cases. r can be evaluated first, since the formula is always the same. Note that some `if` statements are nested.

```
function polar = polarForm(x,y)
%input: x,y coming from z =  x + i*y
%ouptut: r, theta for which re^(i*theta) = z
%r has the same formula everywhere, theta has
%different values for different cases.
r = sqrt(x^2 + y^2);
if x > 0
    theta = atan(y/x);
elseif x < 0
```

```
    if y > 0
        theta = atan(y/x) + pi;
    elseif y == 0
        theta = pi;
    elseif y < 0
        theta = atan(y/x) - pi;
    end
elseif x == 0
    if y > 0
        theta = pi/2;
    elseif y == 0
        theta = 0;
    elseif y < 0
        theta = -pi/2;
    end
end
polar = [r, theta];
end
```

Then, in order to execute the code, we use the following code which reads in each point, and then adds the (r, θ) value as a new column in a matrix L. The final command simply prints the values to the screen.

```
X = [2,2,0,-3,-2,-1,0,0,2];
Y = [0,1,3,1,0,-2,0,-2,2];

L = zeros(9,4);
for i=1:9
    L(i,1:2) = [X(i),Y(i)];
    v  = polarForm(X(i),Y(i));
    L(i,3:4) = [v(1),v(2)];
end
L
```

The output is:

```
L =
   2.000000000000000                   0   2.000000000000000                   0
   2.000000000000000   1.000000000000000   2.236067977499790   0.463647609000806
                   0   3.000000000000000   3.000000000000000   1.570796326794897
  -3.000000000000000   1.000000000000000   3.162277660168380   2.819842099193151
  -2.000000000000000                   0   2.000000000000000   3.141592653589793
  -1.000000000000000  -2.000000000000000   2.236067977499790  -2.034443935795703
                   0                   0                   0                   0
                   0  -2.000000000000000   2.000000000000000  -1.570796326794897
   2.000000000000000   2.000000000000000   2.828427124746190   0.785398163397448
```

(4) The first thing is to define a function that outputs the desired quantities. We call this function `dotCross`. Note that I use a subfunction `mgd` to return the magnitude of a vector. I also used a subfunction `crossP` which uses the standard determinant definition, but the built-in function `cross` is perfectly acceptable.

In order to plot the three vectors, we use plot3 on a set of two points: the origin, and the vector coordinates. We add in the option '–' so that the first two lines are dashed.

```
function [theta,c,mag] = dotCross(a,b)
%Input: two vectors in R^3
%Output: the angle between them, the cross
%product, and the magnitude of the cross product.
theta = acos(sum(a.*b)/(mgd(a)*mgd(b)));
c = crossP(a,b);
mag = mgd(c);
A = [zeros(1,3);a];
B = [zeros(1,3);b];
C = [zeros(1,3);c];
plot3(A(:,1),A(:,2),A(:,3),'--',...
    B(:,1),B(:,2),B(:,3),'--',...
    C(:,1),C(:,2),C(:,3));
end

function x = mgd(v)
%Input: vector
%Output: magnitude of the vector
x = sqrt(sum(v.^2));
end

function w = crossP(a,b)
%Input: pair of vectors
%Output: cross product of vectors.
m = [a;b];
w = [det([m(:,2) m(:,3)]),...
    -det([m(:,1) m(:,3)]),...
    det([m(:,1) m(:,2)])];
end
```

In order to evaluate each of the examples, we input the following code. Note that `figure` is called before each call to dotCross so that the plot is saved in a new figure.

```
A = [6 4 2; 3 2 -6; 2 -2 1; -1 0 0];
B = [2 6 4; 4 -3 1; 4 2 -4; 0 -1 0];

thetaList = zeros(1,4);
cList = zeros(4,3);
magList = zeros(1,4);
for i = 1:4
```

```
    figure;
    [theta, c, mag] = dotCross(A(i,:),B(i,:));
    thetaList(i)= theta;
    cList(i,:) = c;
    magList(i) = mag;
end
thetaList
cList
magList
```

The output revealed is:

```
thetaList =

    0.6669    1.5708    1.5708    1.5708

cList =

    4.0000  -20.0000   28.0000
  -16.0000  -27.0000  -17.0000
    6.0000   12.0000   12.0000
         0         0    1.0000

magList =

   34.6410   35.6931   18.0000    1.0000
```

The cross-product vectors are read from left to right.

The graphical displays that are plotted by the `dotCross` function are included below: