MATLAB for the Materials Scientist/Engineer

Kenneth R. Shull
Department of Materials Science and Engineering
Northwestern University

June 8, 2016

Contents

1 MATLAB tutorial

A variety of useful MATLAB tutorials can be found on the web. We have adapted one from MIT's open courseware site [1]. This tutorial can be further adapted for your own purposes, according to the license agreement set up by MIT [2]. Our implementation of the tutorial is available at the following link:

http://msecore.northwestern.edu/matlabtutorial.pdf

This tutorial is identical to the original one from MIT, but we've removed some of the pages that we did not think were as useful for our own students at this time.

Another very useful resources is the online tuturial available to Northwestern students through Lynda.com. The following link will get you there:

http://www.northwestern.edu/hr/workplace-learning/lynda/#lynda

From this page you should be able to login to Lynda.com through one of the quick links. Once your are there, search for MATLAB and you should find the tutorial. The title of the tutorial is 'Up and Running with MATLAB'.

2 MATLAB Script Examples: 316-1

The remainder of this file includes a list of example MATLAB scripts that have been used throughout Northwestern's core course curriculum (http://msecore.northwestern.edu). They are consolidated here in a single location for reference purposes. Each of the plots includes a hyperlink to the actual m file so that you can download it.

2.1 Setting Plotting Defaults (figformat.m)

Scripts that you write can be called from other scripts. This is a convenient way to include a list of commands that you find yourself using very often. Here's a script that sets the defaults for the current figure, so that the plot looks pretty once you create it (or at least it looks the way I want it to look). It is called in a lot of the other scripts that I use. In order for it to work, you need to make sure that this script is somewhere where MATLAB can find it. The easiest thing to do is just to put figfortmsqmat.m in the same directory as the MATLAB script that calls it.

```
set(0,'defaultaxesbox', 'on') % draw the axes box (including the top and right axes)
set(0,'defaultlinelinewidth',2)
set(0,'defaultaxesfontsize',16)
set(0,'defaultfigurepaperposition',[0,0,7,5])
set(0,'defaultfigurepapersize',[7,5]')
```

2.2 Plotting a User-Defined Function (erfsolution.m)

This program was used to plots the error function solution to the diffusion equation.

```
figure
figformat % set some defaults so the figures look pretty
z=linspace(-1,1,200);  % These are the z points
w=[0.2,0.4,0.6]; % these are the three values of the normalized diffusion length that we
  will include in our calculations
c=@(z,w) erf(z/w);  % define a function of two variables, z and w
col={[1,0,0],[0,0.5,0],[0,0,1]};  % these are the three colors (rgb format)
linetype={'-','--','-.'};  % these are the three line types we well used (plain, dashed
  and dash-dot)
axes
hold on
for i=1:3
    plot(z,c(z,w(i)),'color',col{i},'linestyle',linetype{i})
    legendtext{i}=['$w/L$=' num2str(w(i))];
end
legend(legendtext,'location','best','interpreter','latex')
ylabel('C_{a}')
xlabel('z/L')
ylim([-1.2 1.2])
```

```
set(gca,'ytick',[-1,1])
set(gca,'yticklabel',[])  % turn off the y axis tick labls by making 'yticklable' an
  empty vector
text(-1.15, -1, 'C_{1}','fontsize',16)
text(-1.15, 1, 'C_{2}','fontsize',16)
print(gcf,'../figures/erfsolution.eps','-depsc2') % save as an eps file
```

The plot generated by erfsolution.m is shown in Figure 2.1.

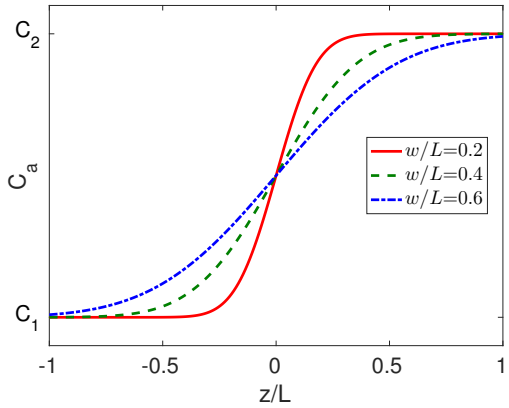

Figure 2.1: Plot generated by the 'erfsolution.m' script.

2.3 Simulation of Interdiffusion by Vacancy Hopping (vacancydiffusion.m)

The following program was used to run a vacancy diffusion simulation.

```
tic % start a time so that we can see how long the program takes to run
n=30;  % set the number of boxes across the square grid
vfrac=0.01;  % vacancy fraction
matrix=ones(n);
map=[1,1,1;1,0,0;0,0,1];  % define 3 colors:  white, red, blue
figure
colormap(map)  % set the mapping of values in 'matrix' to a specific color
caxis([0 2]) % range of values in matrix goes from 0 (vacancy) to 2
% the previous three commands set things up so a 0 will be white, a 1 will
% be red and a 2 sill be blue
matrix(:,n/2+1:n)=2;  % set the right half of the matrix to 'blue'
i=round(n/2);   % put one vacancy in the middle
j=round(n/2);
matrix(i,j)=0;
imagesc(matrix); % this is the command that takes the matrix and turns it into a plot
t=0;
times=[1e4,2e4,5e4,1e5];
showallimages=1; % set to zero if you want to speed things up by not showing images,  set
   to 1 if you want to show all the images during the simulation

%% now we start to move things around
vacancydiff.matrices={};  % makea blank cell array
while t<max(times);
    t=t+1;
    dir=round(4*rand+0.5);
    if dir==1
        in=i+1;
        jn=j;
        if in==n+1; in=1; end
    elseif dir==2
        in=i-1;
        jn=j;
        if in==0; in=n; end
```

```
    elseif dir==3
        in=i;
        jn=j+1;
        if jn>n; jn=n; end
    elseif dir==4
        in=i;
        jn=j-1;
        if jn==0; jn=1; end
    end
    % now we need to make switch
    neighborix=sub2ind([n n],in,jn);
    vacix=sub2ind([n n],i,j);
    matrix([vacix neighborix])=matrix([neighborix vacix]);
    if showallimages
        imagesc(matrix);
        drawnow
    end
    if ismember(t,times)
        vacancydiff.matrices=[vacancydiff.matrices {matrix}]; % append matrix to output
          file
        imagesc(matrix);
        set(gcf,'paperposition',[0 0 5 5])
        set(gcf,'papersize',[5 5])
        print(gcf,['vacdiff' num2str(t) '.eps'],'-depsc2')
    end
    i=in;
    j=jn;
end
vacancydiff.times=times;
vacancydiff.n=n;
save('vacancydiff.mat','vacancydiff') % writes the vacancydiff structure to a .mat file
  that we can read in later
toc
```

2.4 Plotting Data from the Vacancy Diffusion Simulation (vacancyplot.m)

The following script takes the data that was saved to the .mat file created by the vacancy diffusion simulation, and plots it in a useful way.

```
load vacancydiff % load the previously saved output.mat file
figure
figformat % not necessary, this is the standard initialization script I use to
  standardize what my plots look like
n=vacancydiff.n;
matrix=vacancydiff.matrices{4};
matrixsum=sum(matrix,1); % sum of each column in the matrix
plot(1:n,matrixsum/n,'+b')
xlabel ('z')
ylabel ('C')
print(gcf,'../figures/vacancyplot.eps','-depsc2') % this creates an .eps file, which I
  use for the coursenotes but which may not be as useful for many of you as the jpg file
% saveas(gcf,'vacancyplot.jpg') % this is what to do if you just want to save a .jpg file
```

3 MATLAB Script Examples: 316-2

3.1 A Simple MATLAB Plot (nvplot.m)

Here's a script that plots some user-inputed data set, with x and y axis labeled appropriately. It generates the plot shown in Figure 3.1.

```
clear all
close all
set(0, 'DefaultAxesFontSize', 16);
set(0, 'DefaultLineLineWidth', 2);
set(0, 'DefaultFigurePaperposition',[0 0 5 5])
set(0, 'DefaultFigurePaperSize',[5 5])
xdata=linspace(70,80,100);
ydata=1e33*exp(-xdata);
semilogy(xdata,ydata);
xlabel('W_{R}^{*}/k_{B}T')
ylabel('N_v (cm^{-3}s^{-1})')
print(gcf,'-depsc2','nvplot.eps')
```

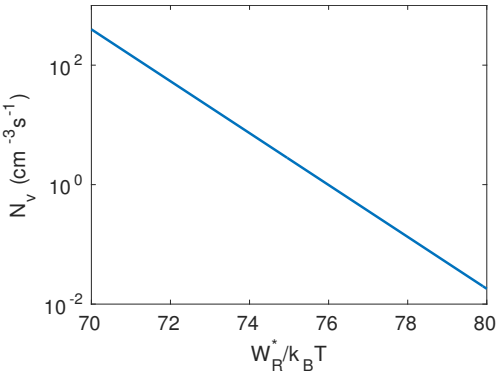

Figure 3.1: Plot generated by the 'nvplot.m' script.

4 MATLAB Script Examples: 331

4.1 Fitting a User-Defined Function to a Data Set (atplot.m)

Here's a function that takes set of data, in this case a_T as a function of T, and fits it to the following functional form:

$$\log a_T = A + \frac{B}{T - T_\infty}$$

```
% input the data that we ned
t=[75 85 100 120 140];
at=[1 0.16 2e-2 2.2e-3 4.5e-4];
logat=log10(at);

% make a new figure window and set the defaults for the plot
figure
pause(1)
figformat % if you use this line, make sure that you include figformat.m in the same
  directory as atplot.m

% plot the experimental data, label it and set the plot limits
% appropriately
plot(t,logat,'b+', 'markersize',18)
xlabel('T (^{\circ}C)')
```

```
ylabel('log(a_{T})')
xlim([70 150])
ylim([-4 1])

% the fitting parameters are all stored as elements of a single variable,
% which in our case is called parm
parm(1)=0; % initial guess for T infinity
parm(2)=700;  % initial guess for B
parm(3)=logat(1)-parm(2)/(t(1)-parm(1));  % requires at=1 for first data point

% the function that we fit to is called volgel in this case.  It is a
% function of two variables:  parm (which includes all of the adjustable
% parameters) and the independent varialbe, which in this case is the
% temperature
vogel=@(parm,t) parm(3)+parm(2)./(t-parm(1));

% lsqcurvefit (least sqaures curve fit) is the MATLAB function that does
% all the work for us.  It retrns the values of parm that give the best fit
% to the data
yfit=lsqcurvefit(vogel,parm,t,logat);

% now we plot the fitted curve over the experimental data
xforplot=linspace(min(t),max(t),100);
yforplot=vogel(yfit,xforplot);
hold on % we set hold to 'on' so that the previously plotted data is not erased when we
  generate the new plot
plot(xforplot,yforplot,'-b')

% now we save the plot as an .eps file
print(gcf,'atplot.eps','-depsc2')
```

The plot generated by this script is shown in Figure 4.1.

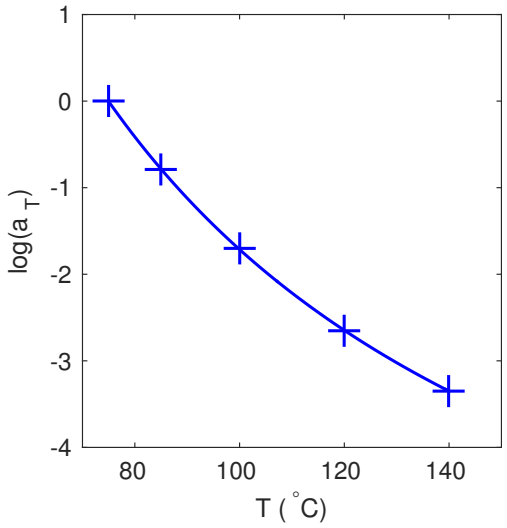

Figure 4.1: Plot generated by the 'atplot.m' script.

4.2 Common Tangent Construction (commontangent.m)

The following script shows the common tangent construction for the regular solution form of the free energy that is used to describe the phase behavior of polymer blends. It generates the plot shown in Figure 4.2

```
global na nb chi % these values also get used in function definitions
```

```
set(0,'defaultaxesfontsize',16)
set(0,'defaultlinelinewidth',2)
phi=linspace(0.001,0.999,1000);
na=150;
nb=100;
chi=0.02;
% fv is the expression for the free energy of mixing
fv=@(phi) phi.*log(phi)/nb+(1-phi).*log(1-phi)./na+chi.*phi.*(1-phi);

%mua and  mub are the chemical potentials of A and B
mua=@(phi) log(1-phi)+phi*(1-na/nb)+chi*na*phi.^2;
mub=@(phi) log(phi)+(1-phi)*(1-nb/na)+chi*nb*(1-phi)^2;

% now write the function that is equal to zero when the A and B chemical
% potentials are equal to one another for phi=phi(1) and phi=phi(2)

ftosolve=@(phi) [mua(phi(1))-mua(phi(2));
    mub(phi(1))-mub(phi(2))];

plot(phi,fv(phi));
xlabel('\phi_{b}')
ylabel('G_{v}V_{0}/RT')

% start with a guess for the equilibrium volume fractions
phiguess = [0.2; 0.8];     % Make a starting guess at the equilibrium compositions
[phicalc,fval] = fsolve(ftosolve,phiguess);  % Call solver

% now we add the tangent line
slope=(fv(phicalc(2))-fv(phicalc(1)))/(phicalc(2)-phicalc(1));
intercept=fv(phicalc(1))-slope*phicalc(1);
tangentline=intercept+slope.*phi;
hold on
plot(phi,tangentline,'r')
hold off
title(['\chi=' num2str(chi) ', N_a=' num2str(na), ', N_b=' num2str(nb)], 'fontsize', 11)

% now save the plot as a jpg file
saveas(gcf,'commontangent.jpg');

% this saves the file a .eps file, used to embed the figure into the
% solution set - students can comment out this next line if they don't wan
% the .svg file
print(gcf,'-dsvg','commontangent.svg')
```

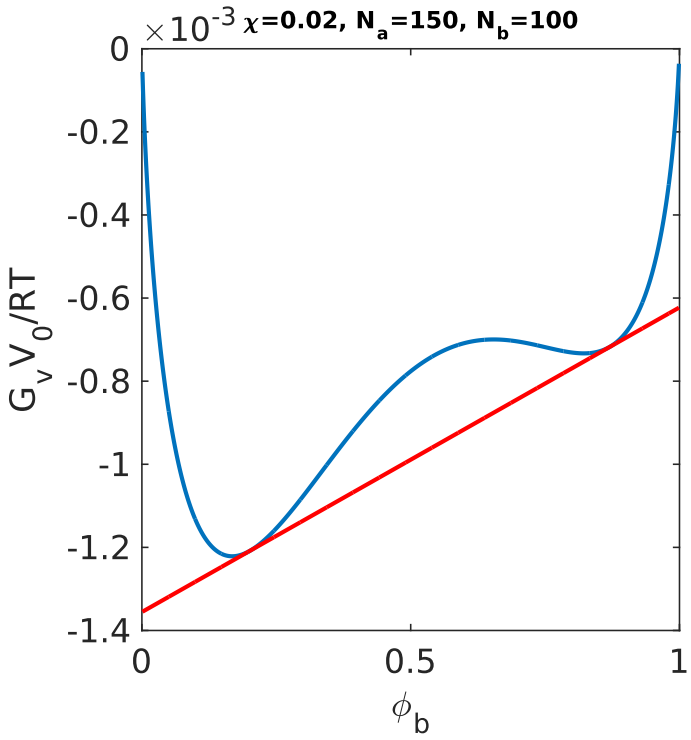

Figure 4.2: Illustration of the common tangent construction.

4.3 Regular Solution Phase Diagram (calcphasediagram.m)

This script generates a phase diagram based on the regular solution theory that is often used to describe mixing of polymer systems. I generates the plots shown in Figure 4.3.

```
close all % close all the existing plot windows to clean things up a bit
clear all % clear all defined variables

% the following are optional commands, used here to set the font
% size and width of the plotted lines so that the plots look good
set(0,'defaultaxesfontsize',16)
set(0,'defaultlinelinewidth',2)
set(0,'defaultfigurepapersize',[5 5])
set(0,'defaultfigurepaperposition',[0 0 5 5])

na=150;
nb=100;

% solution for equilibrium phase compositions
tcrit=350; % critical temperature
chimaxratio=1.5; % this is maximum chi for phase diagram, normalized by chicrit
npoints=1000; % number of different chi points to include in diagram
chicrit=(na+nb+2*(na*nb)^0.5)/(2*na*nb);
phicrit=na^0.5/(na^0.5+nb^0.5);
chimax=chicrit*chimaxratio;
chimin=chicrit*(1+chimaxratio/(npoints-1));

%{
Now we determine the rest of the phase diagram, starting from the largest
values of chi and working toward the critical point
```

```
%}
j=0;
phib=[0.01 0.99]; % initial guess for solution at highest chi
for chi=linspace(chimax, chimin, npoints);
    j=j+1
    % define all the necessary equations
    fv=@(phi) phi.*log(phi)/nb+(1-phi).*log(1-phi)./na+chi.*phi.*(1-phi);
    mua=@(phi) log(1-phi)+phi*(1-na/nb)+chi*na*phi.^2;
    mub=@(phi) log(phi)+(1-phi)*(1-nb/na)+chi*nb*(1-phi)^2;
    ftosolve=@(phi) [mua(phi(1))-mua(phi(2));
        mub(phi(1))-mub(phi(2))];
    chiplot(j)=chi;
    a=0.5-1/(4*chi*na)+1/(4*chi*nb);
    phis1(j)=a-(a^2-1/(2*nb*chi))^0.5;
    phis2(j)=a+(a^2-1/(2*nb*chi))^0.5; %#ok<*SAGROW>
    phib = fsolve(ftosolve,phib); % previous solution is used as guess for
    % this value of chi
    phib1(j)=phib(1);
    phib2(j)=phib(2);
end

% now we generate the chi-based phase diagram
figure
plot(phis1,chiplot, 'r');
xlabel('\phi_{b}')
ylabel('\chi')
hold on
plot(phis2,chiplot, 'r')
plot(phib1,chiplot)
plot(phib2,chiplot)
plot(phicrit,chicrit,'r+','markersize',18)
hold off
title(['N_{a}=' num2str(na) '; N_{b}=' num2str(nb)])
pause(1) % not sure why this is needed, but the svg file isn't always generated correctly
    without it
print(gcf,'-dsvg','flory_chi_diagram')

% now plot temeprature curve - we just need to convert chi values to Temp.
% values
figure % make a new figure
t=tcrit*chicrit./chiplot;
plot(phis1,t, 'r');
xlabel('\phi_{b}')
ylabel('T (K)')
hold on
plot(phis2,t, 'r')
plot(phib1,t)
plot(phib2,t)
plot(phicrit,tcrit,'r+','markersize',18)
hold off
title(['N_{a}=' num2str(na) '; N_{b}=' num2str(nb) '; T_{crit}=' num2str(tcrit) 'K'])

%  now write the file
print(gcf,'-dsvg','flory_t_diagram')
```

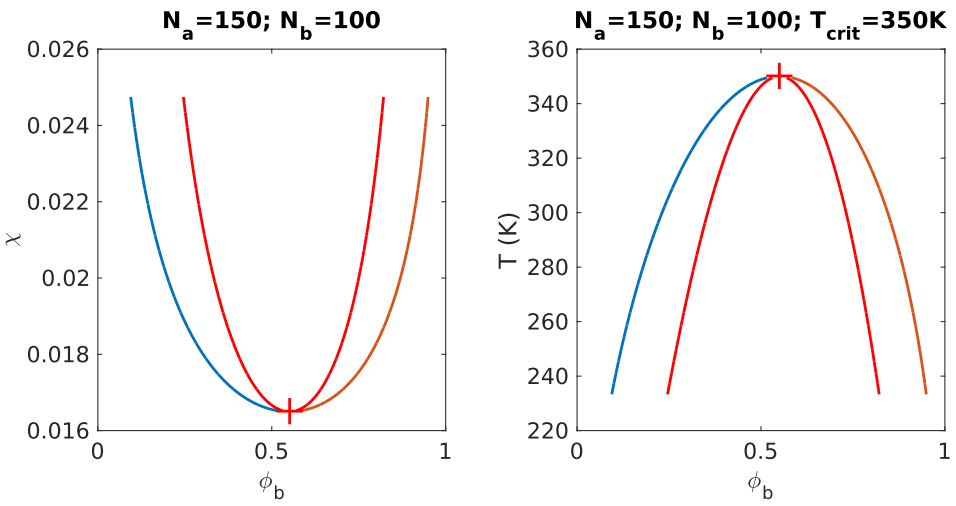

Figure 4.3: Phase diagrams obtained from calcphasediagram.m.

5 MATLAB Script Examples: 332

5.1 Rotation of a Stress Tensor I: Nested Loops (rotate45.m)

Suppose we want to obtain the stress tensor in the transformed coordinate system obtained from a 45° counterclockwise rotation around the z axis. The following MATLAB code solves for the full transformed tensor. This following code produces the output shown in Figure 5.1, which is a representation of the tensor stress state in the transformed coordinate system. We show it here as an example of a calculation with lots of nested 'for' loops:

```
%rotate45.m
sig=zeros(3); % create stress tensor and set to zero
sig(1,1)=5e6; % this is the only nonzero component

sigp=zeros(3); % initalize rotated streses to zero
phi=45;
theta=[phi,90-phi,90;90+phi,phi,90;90,90,0];
for i=1:3
    for j=1:3
        for k=1:3
            for l=1:3
                sigp(i,j)=sigp(i,j)+cosd(theta(k,i))*cosd(theta(l,j))*sig(k,l);
            end  % cosd takes degrees instead of radians as the unit
        end
    end
end
sigp %display the transformed tensor components
```

```
sigp =

   1.0e+06 *

    2.5000    2.5000         0
    2.5000    2.5000         0
         0         0         0
```

Figure 5.1: MATLAB output generated by rotate45.m and by rotate45matrix.m.

5.2 Rotation of a Stress Tensor II: Matrix Multiplication (rotate45matrix.m)

In MATLAB matrices can be multiplied as regular numbers, which in some cases is really handy. The MATLAB code to take a uniaxial stress state and rotate the coordinate system by 45° (the same thing we accomplished in the previous example), can be written this way:

```
%rotate45_matrix.m
sig=zeros(3); % create stress tensor and set to zero
sig(1,1)=5e6; % this is the only nonzero component
phi=45;
theta=[phi,90+phi,90;90-phi,phi,90;90,90,0];
Q=cosd(theta);
QT=transpose(Q);
sigp=Q*sig*QT
```

Running this script gives the output shown in Figure 5.1.

5.3 Rotation of a Stress Tensor III: Symbolic Math (rotatesym_matrix.m)

The symbolic math capability of MATLAB can do some pretty complex algebra for us. As an example, we can consider the rotation of the stress tensor considered in the previous example, but we'll specify the rotation as a variable (phi) instead of as a single numerical value. In this example we use a slightly more complicated stress tensor as the input, with normal stresses in all three untransformed directions but without shear stresses. The following MATLAB script results in the output shown in Figure 5.2.

```
%rotatesym.m
clear all
syms phi; % declare as symbolic variable
sig=sym(zeros(3)); % create sybolic stress tensor and set to zero
syms sig1p sig2p sig3p % these are the  normal stresses
sig(1,1)=sig1p; % put normal stresses into the tensor
sig(2,2)=sig2p;
sig(3,3)=sig3p;
theta=[phi,pi/2+phi,pi/2;pi/2-phi,phi,pi/2;-pi/2,-pi/2,0];
Q=cos(theta);
QT=transpose(Q);
sigp=Q*sig*QT;
sigp=simplify(sigp);
pretty(sigp);
```

```
+-                                                                                  -+
|                          2                 2          sin(2 phi) (sig1p - sig2p)   |
|  sig1p - sig1p sin(phi)  + sig2p sin(phi) ,          -------------------------,  0 |
|                                                                   2                |
|                                                                                    |
|          sin(2 phi) (sig1p - sig2p)                                 2              2 |
|          -------------------------,       sig2p + sig1p sin(phi)  - sig2p sin(phi) ,  0 |
|                     2                                                              |
|                                                                                    |
|                     0,                                  0,                  sig3p  |
+-                                                                                  -+
```

Figure 5.2: MATLAB output generated by rotatesym_matrix.m.

5.4 Principal Stress Calculation (principal_stress_calc.m)

The principle stresses are the Eigen values of the stress tensor, so we use the 'eigs' command to get them from MATLAB. Here's an example script that does this:

```
clear all % clear the workspace of previously defined variables
sig=1e6*[2.5,2.5,0;2.5,2.5,0;0,0,0];
[directions,principalstresses]=eigs(sig);
% the columns in 'directions' correspond to the dot product of the
% principal axes with the orignal coordinate system
% The rotation angles are obtained by calculating the inverse cosines
theta=acosd(directions)
principalstresses
```

Here's the output generated by this script:

```
theta =

   45.0000   90.0000  135.0000
   45.0000   90.0000   45.0000
   90.0000         0   90.0000


principalstresses =

    5000000          0          0
          0          0          0
          0          0          0
```

Figure 5.3: MATLAB output generated by principal_stress_calc.m.

References

[1] Introduction to MATLAB | Electrical Engineering and Computer Science | MIT OpenCourseWare [accessed 2015-03-18].
URL http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-094-introduction-to-matlab-january-iap-2010/ 1

[2] Creative Commons — Attribution-NonCommercial-ShareAlike 3.0 United States — CC BY-NC-SA 3.0 US [accessed 2015-03-18].
URL http://creativecommons.org/licenses/by-nc-sa/3.0/us/deed.en_US 1

Index