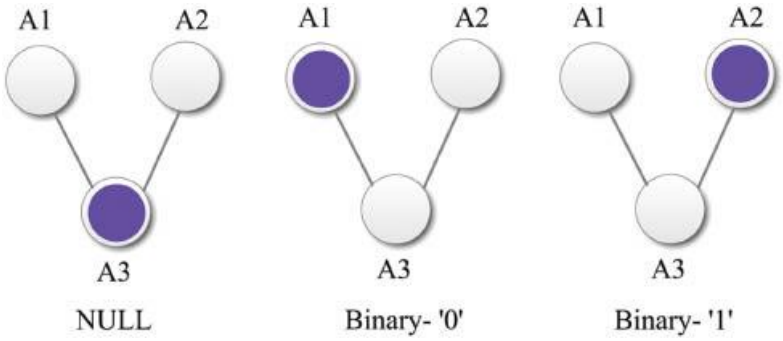
A Brief Tutorial on QCA Three Dot Cells

Joe Previti

22 May 22, 2018

Purpose

The purpose of this tutorial is to give the reader insight to how the basic structure and MATLAB code of a three dot driver cell's interaction with its target cell (also three dots). Instead of existing in just 0 or 1 states, there is a third NULL state, as shown below. An important note is that the 0 and 1 states can also coexist as a superposition, since this is at the quantum level. The picture below depicts the structure seen from the side view. The names A1-3 do not designate variables, this visual is just for clarification as to how the structure looks.



For the implementation of this structure, a few things need to be calculated. This tutorial will discuss how to go about all those calculations, and how they can be realized through a GUI in MATLAB. Once the user has made all the calculations and adapted those calculations to the GUI, he or she will be able to understand to a significantly higher degree just how this three dot model works.

Hamiltonian

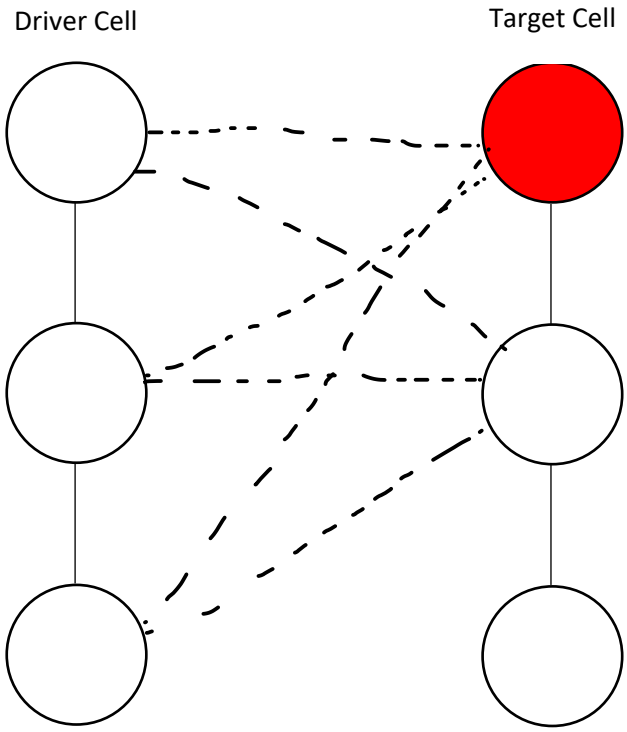
The Hamiltonian matrix is a 3x3 matrix found by calculating each energy of the three target cell dots, along with the clock voltage. For the purposes of this exercise, $E_N = 0J$. Another note is that the units for energy will be in eV, where $1eV = 1.602 * 10^{-19}J$ (the magnitude for the charge of an electron, and the value for which we will use q. The Hamiltonian is calculated as followed:

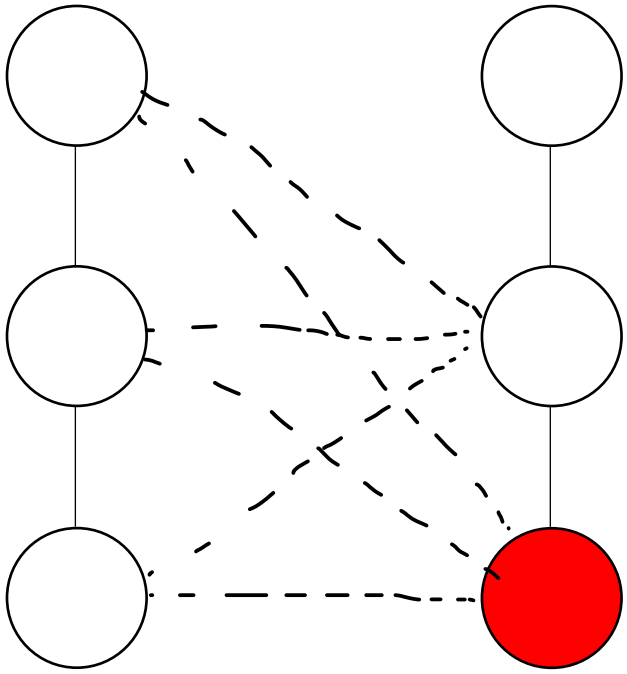
$$H = \begin{bmatrix} E_0 & -\gamma & 0 \\ -\gamma & E_N + V_{clk} & -\gamma \\ 0 & -\gamma & E_1 \end{bmatrix}$$

In order to find each energy, there are 6 calculations required for E_0 and E_1, all of which hinge on the equation $E = \frac{q_1 q_2}{4\pi\epsilon_0 r}$. The variable r is the distance between two charges. For the purposes

of these calculations, the user should begin with $r = a = 1nm$. This is because the electric field is in $\frac{V}{nm}$, making the final calculation a little easier.

The pair of three dot cells shown to the right are how we will look at the geometry for the calculations of each energy. For $q = 1.602 * 10^{-19}C$, the null dots will be perceived to have a +q charge, and the 1 (bottom) or 0 (top) dots will have a –q charge on the target cell. To calculate how much charge each dot on the *driver* has, we use the FindQ function (shown below) given driver polarization and activation. Once q_0 and q_1 are known, the calculation can begin. The dotted lines indicate which Coulombic relationships will be used for the energy calculation. Below the first diagram is the proper diagram for when the target cell is in the 1 state. In order to calculate the charge on q_0 and q_1, we can use the FindQ function on the next page.





```
function [Q1  Q0] = FindQ( P,A )
%Find charge on q1 and q0
q = 1.602e-19;
Q1=0;
Q0=0;
if A~=0
    if P < 0
        Q0=abs(q*((abs(P)+1)/2)*A) ;
        Q1=abs(q*(1-(((abs(P))+1)/2))*A);
    end
    if P > 0
        Q0=abs(q*(1-(((abs(P))+1)/2))*A) ;
        Q1=abs(q*((abs(P)+1)/2)*A);
    end
    if P==0
        Q0 = q*A/2;
        Q1=Q0;
    end
else
    Q0=0;
    Q1=Q0;
end
end
```

Given some polarization and activation of the driver cell, we can use the above function to determine the charges at the top and bottom of the driver cell. Now we can use the equation for energy $E = \frac{q_1 q_2}{4\pi\epsilon_0 r}$ provided that the distance between the two cells is $a = 1nm$ and the vertical length of each three dot cell is a as well. Using trigonometry, we can attain E_1 and E_0 and begin filling in our empty H matrix. The midpoint of each side of the matrix will also get $-\gamma$.

```
K=1/(4*pi*eps);
H=zeros(3);
H(1,1) = K*(q0/a - q0/a1 + q/a -q/a1 + q1/(a*sqrt(2)) - q1/a1);


H(2,2) = -a*Ez*10^9/2;


H(3,3) = K*(q1/a - q1/a1 + q/a -q/a1 + q0/(a*sqrt(2)) - q0/a1);
%continues on the next page…

 %filling in the 4 gamma spots
 H(2,1)=-g;
 H(3,2)=-g;
 H(1,2)=-g;
 H(2,3)=-g;
```

And now we have the Hamiltonian! With this we can calculate, P_{target} and A_{target}. Both of which are critical in the graphical understanding of this phenomemon.

Target Cell Calculations

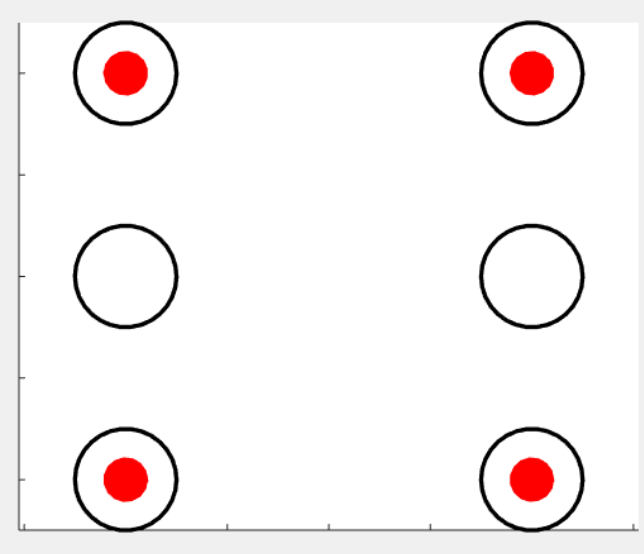
To find the polarization and activation of the target cell, that is simply plugged into the following code once we find the eigenvalues of the Hamiltonian. Something important to note is that the eig() function in MATLAB can be variable output. If the user sets the function up just like it is done below, then the A matrix will be the three eigenvectors, while the B matrix will be a diagonal matrix containing the three eigenvalues (rules of eigenspace, as you might recall from Linear Algebra). The first eigenvector is used in the calculation because it has the lowest energy.

```
[A,B] = eig(H);

psi = A(:,1);

Pt = psi' * Z * psi;

Pn = [ 0 0 0; 0 1 0; 0 0 0 ];

At = 1 - (psi' * Pn * psi);
```

Voila!

Visualization

In order to fully realize this system, the user needs a couple functions that are able to help draw the prevailing data. One of them is the function that draws the three quantum dots with appropriate levels of filling, and the other is for a pseudocolor plot. Furthermore, the ThreeDotGraph() function uses the circle() function to draw the circles. We will discuss the three dot graph first, then the pseudocolor, and finally the GUI itself and how the user can manipulate inputs to change the GUI's appearance.



The circle() function is as follows:

```
function h = circle(varargin)
%
% H = CIRCLE(x0, y0, R, C) creates a circular patch at position (x0,y0),
% with radius R, and color C.  Returns handle to circle.
%
%
% H = CIRCLE(x0, y0, R, C, prop1, val1, ...)
%
% PROPERTIES
%
%    'Points' is used to specify the number of points specifying a polygon
%             which approximates a circle. Specify this as an integer. The
%             default value is 15.
%
%    'EdgeColor' is used to specify the color of the line defining the
%                circle. Specify this as a RGB triple [R G B], with each
%                element in the color value on the interval [0, 1].
%
%    'FillColor' is used to specify the color of the interior of the
%                circle. Specify this as a RGB triple [R G B], with each
%                element in the color value on the interval [0, 1].
%
%    'LineWidth' is used to specify the width of the line defining the
%                circle. Specify this as a one-by-one number.
%
%    'LineStyle' is used to specify the style of the line defining the
%                circle. Specify this as a character string. Some options:
%
%                '-'     Solid line
%
%                '--'    Dashed line
%
%                ':'     Dotted line
%
nth=15;

x0 = varargin{1};
y0 = varargin{2};
R = varargin{3};
C = varargin{4};

EdgeColor = [0 0 0];
LineWidth = 2;
LineStyle = '-';


args = varargin(5:end);
while length(args) >= 2
    prop = args{1};
    val = args{2};
    args = args(3:end);
    switch prop
        case 'Points'
            if ischar(val)
                nth = str2num(val);
```

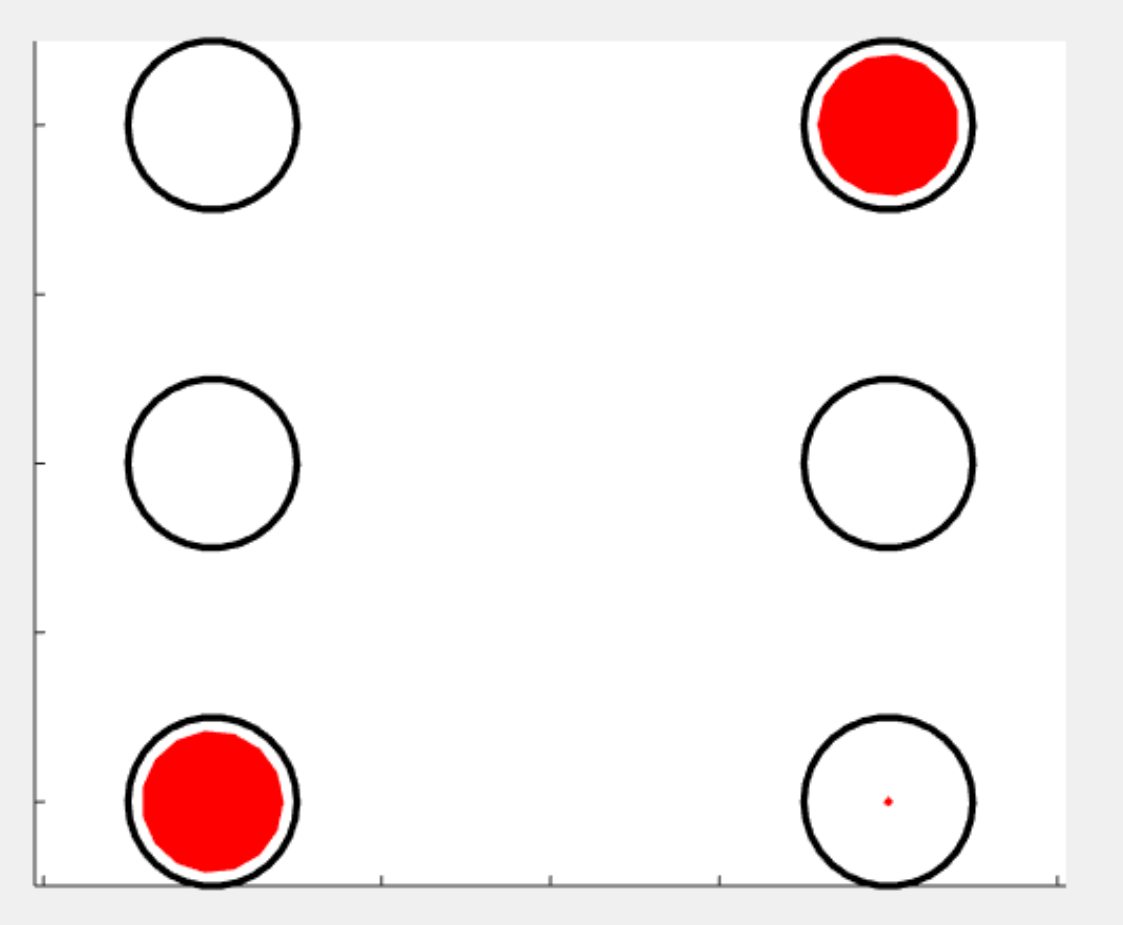
```
            else
                nth = val;
            end

        case 'FillColor'
            FillColor = val;
        case 'EdgeColor'
            EdgeColor = val;

        case 'LineWidth'
            LineWidth = val;

        case 'LineStyle'
            LineStyle = val;

        otherwise
            error(['CIRCLE.M: ', prop, ' is an invalid property specifier.']);
    end
end

thetas=[0:nth]*2*pi/nth;
dx=R*cos(thetas);
dy=R*sin(thetas);
x=x0+dx;
y=y0+dy;
h=patch(x,y,C, 'EdgeColor', EdgeColor, 'LineWidth', LineWidth, ...
    'LineStyle', LineStyle);

End
```

Thankfully, the user does not have to conjure up some code to create something as simple as a circle. The circle() function is the constituent part of the ThreeDotGraph() function. This function is the vessel through which the user can see how much of each dot is "filled" to determine the probability of an electron being on that specific dot. Using the arguments Pdrv,Adrv,Pt, At, and TgtAxes, the function will draw the pair of three dot cells next to each other, showing the probabilities discussed above.

```
function ThreeDotGraph( Pdrv,Adrv,Pt,At,TgtAxes )
%plot driven 3 dot cell
% model of how a 3 dot cell works under driver and E field circumstances
% Dots
axes(TgtAxes);
circle(-1, 1, 1/4, [1 1 1], 'EdgeColor', [0 0 0],'Points',30);
circle(-1, 0, 1/4, [1 1 1], 'EdgeColor', [0 0 0],'Points',30);
circle(-1,-1, 1/4, [1 1 1], 'EdgeColor', [0 0 0],'Points',30);

circle(1, 1, 1/4, [1 1 1], 'EdgeColor', [0 0 0],'Points',30);
circle(1, 0, 1/4, [1 1 1], 'EdgeColor', [0 0 0],'Points',30);
circle(1,-1, 1/4, [1 1 1], 'EdgeColor', [0 0 0],'Points',30);

% Electrons: consider how the activation of driver and target affects it
rDrv0 = 0.5*(Pdrv-1);
% rDrvN = .5*(Pdrv);
rDrv1 = 0.5*(Pdrv+1);
circle(-1, 1, Adrv*rDrv0/5, [1 0 0], 'EdgeColor', [1 0 0]);
% circle(-1, 0, rDrvN/5, [1 0 0], 'EdgeColor', [1 0 0]);
circle(-1,-1, Adrv*rDrv1/5, [1 0 0], 'EdgeColor', [1 0 0]);
rTgt0 = 0.5*(Pt-1);
rTgt1 = 0.5*(Pt+1);
circle(1, 1, At*rTgt0/5, [1 0 0], 'EdgeColor', [1 0 0]);
```

```
% circle(1, 0, rTgt1/5, [1 0 0], 'EdgeColor', [1 0 0]);
circle(1,-1, At*rTgt1/5, [1 0 0], 'EdgeColor', [1 0 0]);

axis equal
set(gca, 'XTickLabel', []);
set(gca, 'YTickLabel', []);end
```

Here is an example of what the three dot cells can look like after running this function. Take note that the target cell dots are each have at least some red in them. This demonstrates the notion of quantum superposition.

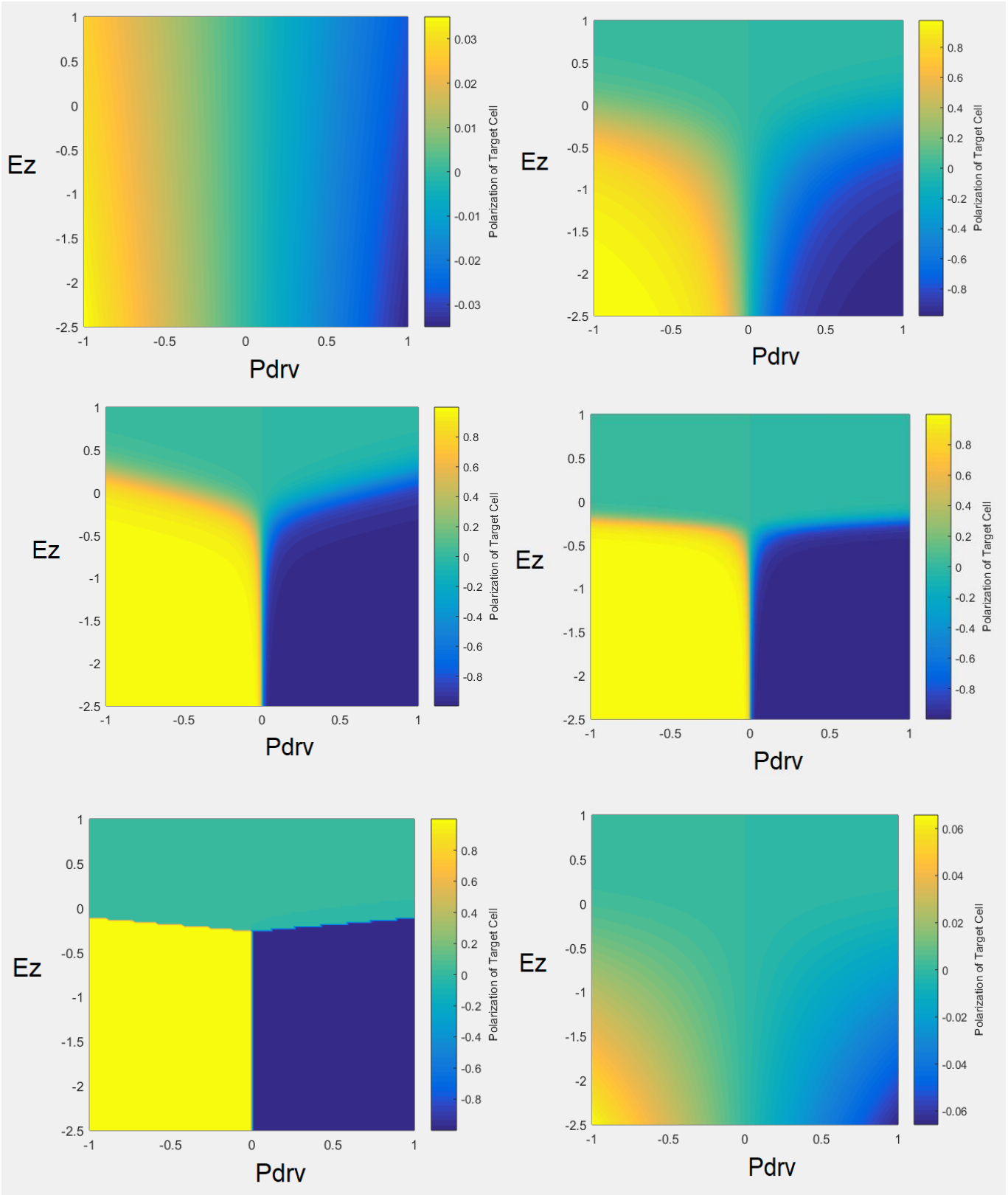


For the pseudocolor map, another function called ShadePlot() will be enlisted. This function will have its own axes separate from the axes which the ThreeDotGraph uses. The plot's output is as if one had a bird's eye view of a surface plot, so the surface looked flat with different levels of shading to the 3D effect. This effect is brought about by having the spectrum of output, P_{target}, being represented by a spectrum of yellow to green to blue. An important note for this function is that the "shading interp" will allow the graph to look more smooth, instead of few discrete chunks, the graph seems to be a continuous color spectrum. The code is on the next page. Take note that this is accomplished with the help of nested *for* loops in order to fill the dependent variable vectors of Electric Field and Driver Polarization. The dependent variable will be the Target Polarization.

```
function ShadePlot(Adrv, a, g, TgtAxes)

axes(TgtAxes);

Z= [-1 0 0;0 0 0;0 0 1];

q = 1.602e-19;

a1= sqrt(a^2 + (a^2)/4);

eps = 8.854e-12; %F/m

K=1/(4*pi*eps);

nPd=100;
nEz=150;

NewH=zeros(3);
PtargetArray=zeros(nPd,nEz);


Ezv = linspace(-2.5,1,nEz);
Pd=linspace(-1,1,nPd);

for Pdidx = 1:nPd
    for Ezidx = 1 : nEz
        [q1 , q0] = FindQ(Pd(Pdidx),Adrv);%C
        NewH(1,1) = K*(q0/a - q0/a1 + q/a -q/a1 + q1/(a*sqrt(2)) - q1/a1);


        NewH(2,2) = -a*Ezv(Ezidx)*10^9/2;


        NewH(3,3) = K*(q1/a - q1/a1 + q/a -q/a1 + q0/(a*sqrt(2)) - q0/a1);


        %filling in the 4 gamma spots
        NewH(2,1)=-g;
        NewH(3,2)=-g;
        NewH(1,2)=-g;
        NewH(2,3)=-g;

        [A,B] = eig(NewH);

        psi = A(:,1);

        Pt = psi' * Z * psi;

        Pn = [ 0 0 0; 0 1 0; 0 0 0 ];

        At = 1 - (psi' * Pn * psi);

        PtargetArray(Pdidx,Ezidx)=Pt;

    end
```

```
end
pcolor(Pd'*ones(1,nEz),ones(nPd,1)*Ezv,PtargetArray);
shading interp;
c=colorbar;
c.Label.String = 'Polarization of Target Cell';

end
```

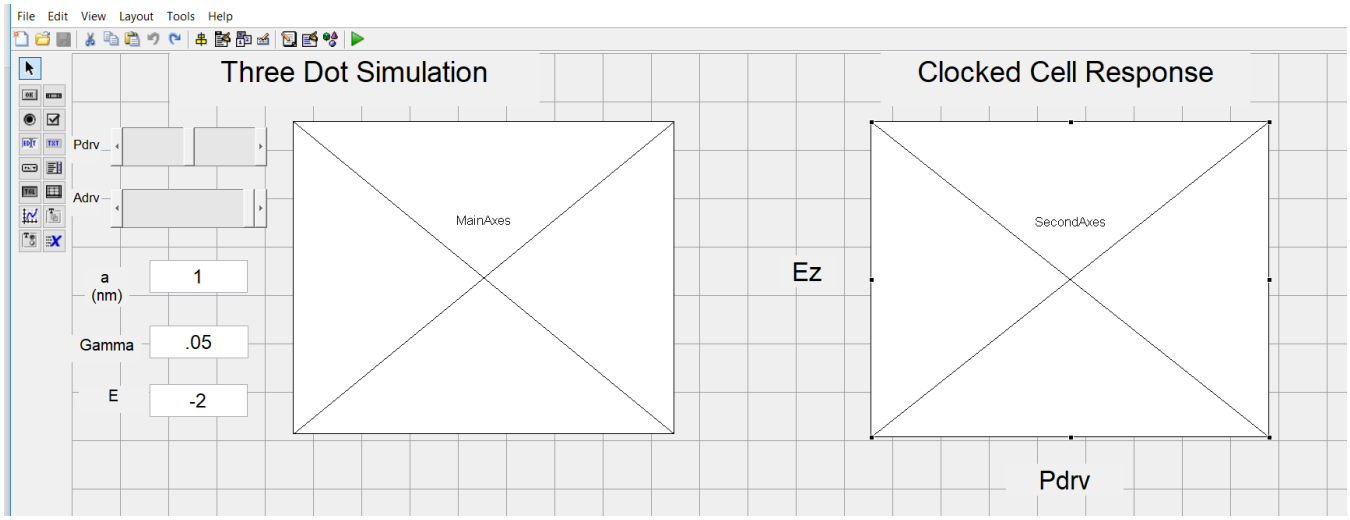
Let's look at some examples! Take note that they all have some different combination of distance, gamma, and Driver Cell Activation.



Making the GUI

This is quite possibly the easier part of the user's task, as long as all the other functions work appropriately. The user should have a main function that he or she used for the main calculations, or at least a master function that calls the constituent functions. Regardless, this main function should take the form (name does not matter)

```
function GroundState3DotPrac(handles)
```



If the user is going to implement a GUI using the *guide* command in MATLAB, this *must* be done. Once the .m file is made, the user can edit the GUI, and it will change the .m file upon saving the edits made. Whenever a uicontrol element is added, it would be in the user's best interests to change the *tag* on that control (i.e. a button or a slider) to something practical like "editGamma." Proper nomenclature will help ensure variables are not left out, and hair does not get pulled out in the process. Once the tag is changed and saved, the user must go into the .m file and a couple things. The first of which is the opening function.

```
function ThreeDotGui_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to ThreeDotGui (see VARARGIN)

% Choose default command line output for ThreeDotGui
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
GroundState3DotPrac(handles);
```

Take note that the highlighted portion is calling the main function with the proper argument *handles.* This allows the GUI to pass information to the main function.

The next aspect of the GUI .m file that must be changed is the *callback* functions within the said file. For instance:

```
% --- Executes on slider movement.
function editPdrv_Callback(hObject, eventdata, handles)
% hObject    handle to editPdrv (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
GroundState3DotPrac(handles);
```

Again, the main function is called with argument *handles* in order to edit the P_{driver} value. Another critical facet to pay attention to is to which axes and variables the graphing functions are tied. In the GUI on the prior page, the ThreeDotGraph is on *MainAxes* and the ShadePlot is with the *SecondAxes*. These will be referenced during the function call and used as the *TgtAxes* in each graphic function. The same logic holds for the affected variables within the main function, as shown directly below the two graph examples. Take note that for a slider uicontrol, the *value* is in the referenced aspect of handles, while the edit box has *String* as the point of reference. Furthermore, that *String* must be turned into a numerical value (see str2num).

```
ThreeDotGraph( Pdrv,Adrv,Pt,At,handles.MainAxes );
ShadePlot(Adrv, a, g, handles.SecondAxes);

Ez = str2num(get(handles.editE,'String'));
Pdrv = get(handles.editPdrv,'Value');
```

The rest should follow. As long as the use ensures that all the proper variables are being used in a appropriate manner, and all the variables affected by the GUI are properly defined within the main function (which is the only function for this GUI that may receive the argument *handles*), then the user can expect an output similar to this.