Vector Diagram Simulation Package:
A Matlab Toolbox

Ohio Advanced EPR Laboratory
Robert McCarrick - 1/23/2012

User Tutorial

It is recommended that the user read the brief Matlab Primer included with the package if not familiar with Matlab. In addition, there is a more concise description of the functions, their usage and the available parameters in the Read Me file within the unzipped directory.

This toolbox for Matlab consists of two functions. One accepts an array of spins with a given orientation and frequency offset from the rotating frame and allows the spins to evolve for a given time period according to the frequency offset, T_1 and T_2. The second accepts an array of spins and carries out a pulse of a specific frequency along one of the principle axes. The outputs of these functions are the final positions of the spins. This allows the two functions to be strung together to simulate various pulse sequences in order to visualize the classical vector diagram approach to understanding magnetic resonance.

With both functions, a figure is created with the vector diagram depiction of the spins on the left and the total magnetization along the X_R, Y_R and Z_R axes (M_X, M_Y and M_Z respectively) plotted as a function of angle for the pulse function and time for the evolution function.

We will start by constructing an array of spins with a defined offset frequency. The array needs to be put into a field in a structure with the label "degrees". Each row in the array corresponds to standard three dimensional spherical coordinates of [φ θ r] where φ is the angle between the X axis and the projection of the vector on the XY plane, θ is the angle between the Z axis and the vector and r is the length of the vector.

To start, lets just input one spin in the direction of each of the axes.

Remember to always use clear all at the beginning of scripts to ensure that old variables do not carry over to a new script!

```
clear all

Spins.degrees = [0 0 1;0 90 1;90 90 1]
```

First, we will use the pulse function to apply a 90 degree microwave pulse along the X axis. To do this, we need to give x as the direction of the pulse and 90 degrees as the tip angle.

```
Sim.direction = 'x'
Sim.angle = 90
```

Now we can call the pulse function in the following manner. We give the Spin.degrees variable as the output. The output of the pulse function is the final position of the spins in spherical coordinates. This

will allow us to use the output as the input into the next function.

```
Spins.degrees = pulsespin(Spins,Sim)
```

As you can see, the spins rotate about the X_R axis according to the right hand rule with the projections of the spins along each axis being reflected in the right hand side of the figure.

Now let's change the direction and angle of the pulses.

```
Sim.direction = 'y'
Sim.angle = 180
```

We can look at the final position of these spins by simply recalling the Spins.degrees field in the command window.

```
>> Spins.degrees

ans =

        0  180.0000    1.0000
  180.0000   90.0000    1.0000
   90.0000   90.0000    1.0000
```

As you can see, the length of the vectors has stayed the same and just the angles have changed.

Now let's use the evolution function. For this, we'll just use one spin for the time being and orient it along the X axis.

```
clear all

Spins.degrees = [0 90 1]
```

The evolution function allows the spins to evolve according to the Bloch equations for a rotating frame. The B_0 field causes a torque about the Z axis and the spins precess at their frequency offset from the rotating frame. It will also incorporate T_1 and T_2, but if they are left out, it will choose a default value of 100 seconds (which basically turns it off as those values are very long compared to a typical magnetic resonance experiment). To use the function, we will first put an offset frequency of 0 MHz.

```
Spins.offset = 0
```

For the simulation, we need to give a length of time for the spins to evolve . For this case, we'll use pretty standard EPR time frames.

```
Sim.length = 5e-7
```

We will call the evolution function in the same way as the pulse function.

```
Spins.degrees = evolution(Spins,Sim)
```

You should notice that nothing happened. Why is that? It's because we entered zero for the frequency offset. This means that the spin will stay aligned along the X_R axis. This is to say that the spin is precessing at the same frequency as the rotating frame.

Now let's add a frequency offset of 1 MHz. This means that the spin is rotating at 1 MHz faster than the rotating frame (for instance, if the microwave quantum was 9 GHz, this spin would be moving at 9.001 GHz or (9 GHz plus 1 MHz).

```
Spins.offset = 1e6
```

You will see that the spin rotates one half turn counter clockwise (remember the right hand rule). This is because the offset frequency is 1 MHz or 1,000,000 times per second and the duration of the simulation is 0.0000005 seconds.

Now lets change it to a negative offset. This means that the spin is moving slower than the rotating frame by 1 MHz (or to use the numbers above, 8.999 GHz).

```
Spins.offset = -1e6
```

You will see that the spin now moves in the opposite direction.

Now let's use evolution with multiple spins. We can create a degrees array with several spins oriented along the X axis and give them different offset frequencies.

```
Spins.degrees = [0 90 1;0 90 1;0 90 1;0 90 1;0 90 1]
Spins.offset = [-2e6;-1e6;0;1e6;2e6]
```

In this case, we have one spin that's stationary as it is moving as the same speed of the rotating frame along with two spins each going faster and slower than the rotating frame. In this simulation, you can see four of the five spins fanning out from the initial position.

Once a larger number of spins are being added, it can become cumbersome to manually enter the arrays. An alternative manner for constructing the above arrays is to use a for loop to construct the .degrees array and the Matlab function linspace to create the offset frequencies.

```
for i=1:5
Spins.degrees(i,:) = [0 90 1]
end
```

This loop will run five times, constructing the array one row at a time.

```
Spins.offset = linspace(-2e6,2e6,5)
```

The function linspace will construct an array of evenly spaced values between the first two values and with a total dimension of the third. This might not seem more convenient than manually entering the

values, but for larger arrays, it will save a lot of time.

```
for i=1:250
Spins.degrees(i,:) = [0 90 1]
end
Spins.offset = linspace(-2e6,2e6,250)
```

Now we can string these functions together to construct a simulation of a Hahn Echo sequence. In this script, we define an initial set of values for the spins, and then string together calls of pulse and evolution to construct the simulation. The output of each function call forms the input of the next function.

```
clear all

for i=1:250
    Spins.degrees(i,:) = [0,0,1];
end

Spins.offset = linspace(-1600000,1600000,250);

Sim.direction = 'y';
Sim.angle = 90;

Spins.degrees = pulsespin(Spins,Sim);

Sim.length = 0.0000005;

Spins.degrees = evolution(Spins,Sim);

Sim.direction = 'y';
Sim.angle = 180;

Spins.degrees = pulsespin(Spins,Sim);

Sim.length = 0.0000010;

Spins.degrees = evolution(Spins,Sim);
```

Not mentioned thus far are the extra parameters that we've left out. The parameters given above are simply the minimum parameters required to construct a simulation. There are many more parameters that can be used to change the number of frames, speed of the simulations and even to save the entire simulation as an Animated GIF or AVI file. See the Read Me file in the directory and the example files to become acquainted with the more advanced features of the simulation package.