UniDyn--Demo-04.nb

John A. Marohn
jam99@cornell.edu
Cornell University

Abstract: Use the **UniDyn** Evolver function to calculate the evolution of the magnetization of a two coupled spin = 1/2 particles.

Set the path to the package

Tell *Mathematica* the path to the directory containing the package.

EDIT THE FOLLOWING PATH STRING:

```
$PackagePath =
  "/Users/jam99/Dropbox/MarohnGroup__Software_Library/UniDyn/
    unidyn";
```

YOU SHOULD NOT NEED TO EDIT ANYTHING FROM HERE ONWARDS.

Load the package

Append the package path to the system path. Before trying to load the package, ask *Mathematica* to find it. This is a test that we directed *Mathematica* to the correct directory. The output of this command should be the full system path to the UniDyn.m file.

```
$Path = AppendTo[$Path, $PackagePath];
FindFile["UniDyn`"]
```

```
/Users/jam99/Dropbox/MarohnGroup__Software_Library/UniDyn/UniDyn/UniDyn.m
```

Now that we are confident that the path is set correctlly, load the package. Setting the global $VerboseLoad variable to True will print out the help strings for key commands in the package.

```
$VerboseLoad = True;
Needs["UniDyn`"]
```

```
You are using the version of NCAlgebra which is found in:

  /Users/jam99/Downloads/NC

You can now use "<< NCAlgebra`" to load NCAlgebra or "<< NCGB`" to load NCGB.
```

```
NCMultiplication.m loaded
NC1SetCommands.m loaded
NCInverses.m loaded
NCTransposes.m loaded
NCAdjoints.m loaded
NCCo.m loaded
NCRoots.m loaded
NC2SetCommands.m loaded
NCCollect.m loaded
NCSubstitute.m loaded
NCMonomial.m loaded
NCSolve.m loaded
NCTools.m loaded
NC2SimplifyRational.m loaded
NC1SimplifyRational.m loaded
NCSimplifyRational.m loaded
NCComplex.m loaded
NCMatMult.m loaded
NCDiff.m loaded
NCSchur.m loaded
NCAlias.m loaded
Grabs.m loaded
NCTaylorCoeff.m loaded
NCConvexity.m and NCGuts.m loaded
NCRealizationFunctions.m loaded
NCTeXForm.m loaded
NCTeX::Using 'open' as PDFViewer.
NCTeX.m loaded
NCMaster.m loaded
NCOutput.m loaded
```

```
------------------------------------------------------------
NCAlgebra - Version 4.0.6
Compatible with Mathematica Version 9

Authors:
  J. William Helton*
  Mauricio de Oliveira*
  Mark Stankus*
  Robert L. Miller#

* Math Dept, UCSD
# General Atomics Corp
  La  Jolla, CA 92093

Copyright:
  Helton and Miller June 1991
  Helton 2002
  All rights reserved.

The program was written by the authors and by:
  David Hurst, Daniel Lamm, Orlando Merino, Robert Obar,
  Henry Pfister, Mike Walker, John Wavrik, Lois Yu,
  J. Camino, J. Griffin, J. Ovall, T. Shaheen, John Shopple.
  The beginnings of the program come from eran@slac.
  Considerable recent help came from Igor Klep.


This program was written with support from
  AFOSR, NSF, ONR, Lab for Math and Statistics at UCSD,
  UCSD Faculty Mentor Program,
  and US Department of Education.
  Primary support in 2010 is from the
    NSF Division of Mathematical Sciences.

If you
  (1) are a user,
  (2) want to be a user,
  (3) refer to NCAlgebra in a publication, or
  (4) have had an interesting experience with NCAlgebra,
let us know by sending an e-mail message to

  ncalg@math.ucsd.edu.

We do not want to restrict access to NCAlgebra, but do
  want to keep track of how it is being used.

For NCAlgebra updates see the web page:

  www.math.ucsd.edu/~ncalg
------------------------------------------------------------
You are using the version of NCAlgebra which is found in:

  /Users/jam99/Downloads/NC

You can now use "<< NCAlgebra`" to load NCAlgebra or "<< NCGB`" to load NCGB.

You have already loaded NCAlgebra.m
```

CreateOperator ::usage :

CreateOperator [] is used to batch –define a bunch of operators . Example : CreateOperator [{{Ix, Iy, Iz},{Sx,Sy,Sz}}] will create six operators ; each of the operators in the first list is meant to commute with each of the operators in the second list .

CreateScalar ::usage :

CreateScalar [list] is used to batch –define a bunch of scalars. The parameter list can be a single scalar or a list of scalars. Example : CreateScalar [{w1 ,w2 }].

```
You are using the version of NCAlgebra which is found in:

  /Users/jam99/Downloads/NC

You can now use "<< NCAlgebra`" to load NCAlgebra or "<< NCGB`" to load NCGB.

You have already loaded NCAlgebra.m
```

NCSort::usage : NCSort[list] sorts the operators in list into canonical order.

SortedMult ::usage :
SortedMult [list] returns Mult [list$ordered], where list$ordered are the elements of list sorted into canonical order.

MultSort ::usage :
MultSort [NonCommutativeMultiplyt [list]] returns returns NonCommutativeMultiply [list$ordered], where list$ordered are the elements of list sorted into canonical order.

```
You are using the version of NCAlgebra which is found in:

  /Users/jam99/Downloads/NC

You can now use "<< NCAlgebra`" to load NCAlgebra or "<< NCGB`" to load NCGB.

You have already loaded NCAlgebra.m
```

Comm ::usage : Comm [a,b] calculates the commutator of two operators.

```
You are using the version of NCAlgebra which is found in:

  /Users/jam99/Downloads/NC

You can now use "<< NCAlgebra`" to load NCAlgebra or "<< NCGB`" to load NCGB.

You have already loaded NCAlgebra.m
```

SpinSingle$CreateOperators ::usage :
SpinSingle$CreateOperators [Ix,Iy,Iz,L] creates Ix, Iy, and Iz angular momentum operators and defines their commutation relations. When the total angular momentum L = 1/2, additional rules are defined to simplify products of the angular momentum operators. When the total angular momentum L is unspecified, no such simplification rules are defined.

```
You are using the version of NCAlgebra which is found in:

  /Users/jam99/Downloads/NC

You can now use "<< NCAlgebra`" to load NCAlgebra or "<< NCGB`" to load NCGB.

You have already loaded NCAlgebra.m
```

OscSingle$CreateOperators ::usage :
OscSingle$CreateOperators [aL,aR] creates a raising operator aR and a lowering operator aL for single harmonic oscillator and defines the operator commutation relations.

```
You are using the version of NCAlgebra which is found in:

  /Users/jam99/Downloads/NC

You can now use "<< NCAlgebra`" to load NCAlgebra or "<< NCGB`" to load NCGB.

You have already loaded NCAlgebra.m
```

Evolve::usage :
Evolve[H,t,ρ] represents unitary evolution of the density operator ρ for a time t under the Hamiltonian H. This function expands according to simplification rules but leaves the evolution unevaluated.

Evolver::usage : Evolver[H,t,ρ(0)] calculates ρ(t) = Exp[−I H t] ρ(0) Exp[+I H t], assuming that H is time independent, according to the commutation rules followed by ρ(0) and H.

Evolver with simplification

```
SimplifyingEvolver[H_, t_, ρ$0_] :=

 Evolver[H, t, ρ$0] // Simplify // ExpToTrig // FullSimplify
```

Unitary evolution of a single spin 1/2

Create a single spin

The assumptions define below are required for *Mathematica* to recognize $\sqrt{-\Delta^2-\omega^2} = I\sqrt{\Delta^2+\omega^2}$ inside an exponential. One of the variables has to be defined to be > 0 and not just ⩾ 0.

```
Clear[

 Δω,           (* Resonance offset frequency *)
 ω$1,          (* Rabi frequency of the applied irradiation *)
 Ix, Iy, Iz,   (* Spin angular momentum operators *)
 ρ$0,          (* Initial spin density operator *)
 H             (* Spin Hamiltonian *)]

CreateScalar[Δω, ω$1];
SpinSingle$CreateOperators[Ix, Iy, Iz, L = 1 / 2];

$Assumptions =
  {Element[Δω, Reals], Δω ⩾ 0, Element[ω$1, Reals], ω$1 > 0};
```

```
SpinSingle$CreateOperators ::create : Creating spin operators .

SpinSingle$CreateOperators ::comm : Adding spin commutations relations .

SpinSingle$CreateOperators ::simplify : Angular momentum L = 1/2. Adding operator simplification rules .
```

On-resonance nutation

On-resonance irradiation Hamilonian written in the interaction representation. The intial density operator is parallel to I_x.

```
H = ω$1 Ix;
ρ$0 = Iz;
```

Calculate the time-dependent denstity operator.

```
SimplifyingEvolver[H, t, ρ$0] /. {ω$1 -> Subscript[ω, 1]}
```

```
Iz Cos[t ω1] - Iy Sin[t ω1]
```

Free evolution

Zeeman-interaction Hamilonian written in the interaction representation. The initial density operator is parallel to I_x.

```
H = Δω Iz;
ρ$0 = Ix;
```

Calculate the time-dependent denstity operator.

```
SimplifyingEvolver[H, t, ρ$0]
```

```
Ix Cos[t Δω] + Iy Sin[t Δω]
```

Unitary evolution of two coupled spins

Create a two spins

The assumptions define below are required for *Mathematica* to recognize $\sqrt{-\Delta^2-\omega^2} = I\sqrt{\Delta^2+\omega^2}$ inside an exponential. One of the variables has to be defined to be > 0 and not just ⩾ 0.

```
Clear[

 Δ$I,          (* Resonance offset frequency *)
 Δ$S,          (* Resonance offset frequency *)
 J,            (* Spin-spin coupling *)
 Ix, Iy, Iz,   (* Spin angular momentum operators *)
 Sx, Sy, Sz,   (* Spin angular momentum operators *)
 ρ,            (* Spin density operator *)
 t,             (* Time *)
 ρ$0,          (* Initial spin density operator *)
 H             (* Spin Hamiltonian *)]

CreateScalar[Δ$I, Δ$S, J, t ];
CreateOperator[{{Ix, Iy, Iz}, {Sx, Sy, Sz}}];
SpinSingle$CreateOperators[Ix, Iy, Iz, L = 1 / 2];
SpinSingle$CreateOperators[Sx, Sy, Sz, L = 1 / 2];

$Assumptions = {Element[Δ$I, Reals], Δ$I ⩾ 0,
   Element[Δ$S, Reals], Δ$S ⩾ 0, Element[J, Reals], J > 0};
```

```
SpinSingle$CreateOperators ::nocreate : Spin operators already exist.
SpinSingle$CreateOperators ::comm : Adding spin commutations relations.
SpinSingle$CreateOperators ::simplify : Angular momentum L = 1/2. Adding operator simplification rules.
SpinSingle$CreateOperators ::nocreate : Spin operators already exist.
SpinSingle$CreateOperators ::comm : Adding spin commutations relations.
SpinSingle$CreateOperators ::simplify : Angular momentum L = 1/2. Adding operator simplification rules.
```

Evolution under J coupling

On-resonance irradiation Hamilonian written in the interaction representation. The intial density operator is parallel to I_x.

```
H$0 = Δ$I Iz + Δ$S Sz;
H$J =  J Iz ** Sz;
ρ$0 = Ix;
```

Try to calculate the density operator in a single step.

```
(* ρ$0// SimplifyingEvolver[H$0+H$J,t, #]& *)
```

This single-step evolution fails. Instead calculate the denstity operator in two steps. We can do this because the H$J and H$0 Hamiltonians commute. Evolve under the J-coupling first and under the chemical shift second.

```
ρ = ρ$0 // SimplifyingEvolver[H$J, t, #] & //
  SimplifyingEvolver[H$0, t, #] &
```

$$\text{Cos}\left[\frac{J\,t}{2}\right]\,(\text{Ix}\,\text{Cos}[t\,\triangle\$I] + \text{Iy}\,\text{Sin}[t\,\triangle\$I]) +$$

$$2\,\text{Sin}\left[\frac{J\,t}{2}\right]\,(\text{Cos}[t\,\triangle\$I]\,\text{Iy} ** \text{Sz} - \text{Ix} ** \text{Sz}\,\text{Sin}[t\,\triangle\$I])$$

Try another way -- evolve under the chemical shift first and under the J-coupling second.

```
ρ = ρ$0 // SimplifyingEvolver[H$0, t, #] & //
  SimplifyingEvolver[H$J, t, #] &
```

$$\text{Cos}\left[\frac{J\,t}{2}\right]\,(\text{Ix}\,\text{Cos}[t\,\triangle\$I] + \text{Iy}\,\text{Sin}[t\,\triangle\$I]) +$$

$$2\,\text{Sin}\left[\frac{J\,t}{2}\right]\,(\text{Cos}[t\,\triangle\$I]\,\text{Iy} ** \text{Sz} - \text{Ix} ** \text{Sz}\,\text{Sin}[t\,\triangle\$I])$$

We see by inspection that we get the same answer either way.

Load spin 1/2 operator matrices

Load matrices representing two J=1/2 spins and give them a test drive.

```
<< Matrices--two-spin-half.m
```

```
Loaded spin one-half matrices for mIx, mIy, mIz, mSx, mSy, mSz
```

Let's look at one of the matrices, the matrix for Iz.

```
mIz // MatrixForm
mSz // MatrixForm
```

$$\begin{pmatrix} -\frac{1}{2} & 0 & 0 & 0 \\ 0 & -\frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} \end{pmatrix}$$

$$\begin{pmatrix} -\frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & -\frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} \end{pmatrix}$$

For a single spin 1/2 particle, Iz is a 2 x 2 matrix. In the product space of two spin 1/2 particles, however, the Iz is now a 4 x 4 matrix. All the operators are 4 x 4 matrices.

Check that the [Ix, Iy]/I = Iz and [Sx, Sy]/I = Sz commutation relations hold with the matrices.

```
m$1 = (mIx . mIy - mIy.mIx) / I ;
m$2 = (mSx . mSy - mSy.mSx) / I ;
m$1 // MatrixForm (* should be Iz *)
m$2 // MatrixForm (* should be Sz *)
```

$$\begin{pmatrix} -\frac{1}{2} & 0 & 0 & 0 \\ 0 & -\frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} \end{pmatrix}$$

$$\begin{pmatrix} -\frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & -\frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} \end{pmatrix}$$

Make the (matrix) operator I$total = Iz + Sz. We can see by inspection that this matrix looks correct. The I$total operator should have eigenvalues {-1,0,0,+1}. The matrix I$total is diagonal and we can read the eigenvalues off diagonal elements.

```
m$1 + m$2 // MatrixForm
```

$$\begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Calculate the analytical signal using the matrices

Calculate the density operator matrix.

```
ρ$matrix =
  ρ /. {Ix → mIx, Iy → mIy, Iz → mIz, Sx → mSx, Sy → mSy, Sz → mSz};
ρ$matrix // MatrixForm
```

$$\begin{pmatrix} 0 & 0 & \text{Cos}\left[\frac{J t}{2}\right] \left(\\ 0 & 0 & \\ \text{Cos}\left[\frac{J t}{2}\right] \left(\frac{1}{2} \text{Cos}[t\, \triangle\$I] - \frac{1}{2} i\, \text{Sin}[t\, \triangle\$I]\right) & 0 & \\ 0 & \text{Cos}\left[\frac{J t}{2}\right] \left(\frac{1}{2} \text{Cos}[t\, \triangle\$I] - \frac{1}{2} i\, \text{Sin}[t\, \triangle\$I]\right) & \end{pmatrix}$$

From this matrix we can calculate the I-spin signals as Trace[ρ Ix] and Trace[ρ Iy]

```
Tr[ρ$matrix . mIx] // Simplify
Tr[ρ$matrix . mIy] // Simplify
```

$$\text{Cos}\left[\frac{J\,t}{2}\right]\,\text{Cos}[t\,\Delta\$I]$$

$$\text{Cos}\left[\frac{J\,t}{2}\right]\,\text{Sin}[t\,\Delta\$I]$$

We can mimic the complex signal collected by the NMR spectrometer by calculating the expectation value of the Ix + I Iy operator: Trace[ρ (Ix+ I Iy)].

```
S[t_] := Tr[ρ$matrix . (mIx - I mIy)] // TrigToExp
```

Plot the signal as a function of time

Create a more realistic experimental signal by multiplying the above-calculated signal by a decaying exponential.

```
S$expt[t_] = S[t] Exp[-t / T2]
```

$$e^{-\frac{t}{T2}}\left(\frac{1}{2}\,e^{-\frac{1}{2}\,i\,J\,t-i\,t\,\Delta\$I}+\frac{1}{2}\,e^{\frac{i\,J\,t}{2}-i\,t\,\Delta\$I}\right)$$

To plot, set the total number of points (NN) and the total time (T). We set NN equal to a power of 2 in anticipation of taking a digital Fourier transform.

```
NN = 2^10;
T = 10.0;
```

From NN and T we derive the time step (dt) and the frequency step (df). Now generate a list of data points based on the above function. At the same time, generate a list of time points (t) and frequency points (f) for plotting.

```
dt = T / (NN - 1);
df = 1 / dt;

Y = Table[y[t], {t, 0, T, dt}];

f = Table[jj / T, {jj, -NN / 2, NN / 2 - 1}];
t = Table[ii * dt, {ii, 0, NN - 1}];

Print["The time step is ", dt, " s"]
Print["The Nyquist frequency is ", 1 / (2 * dt), " Hz"]
```

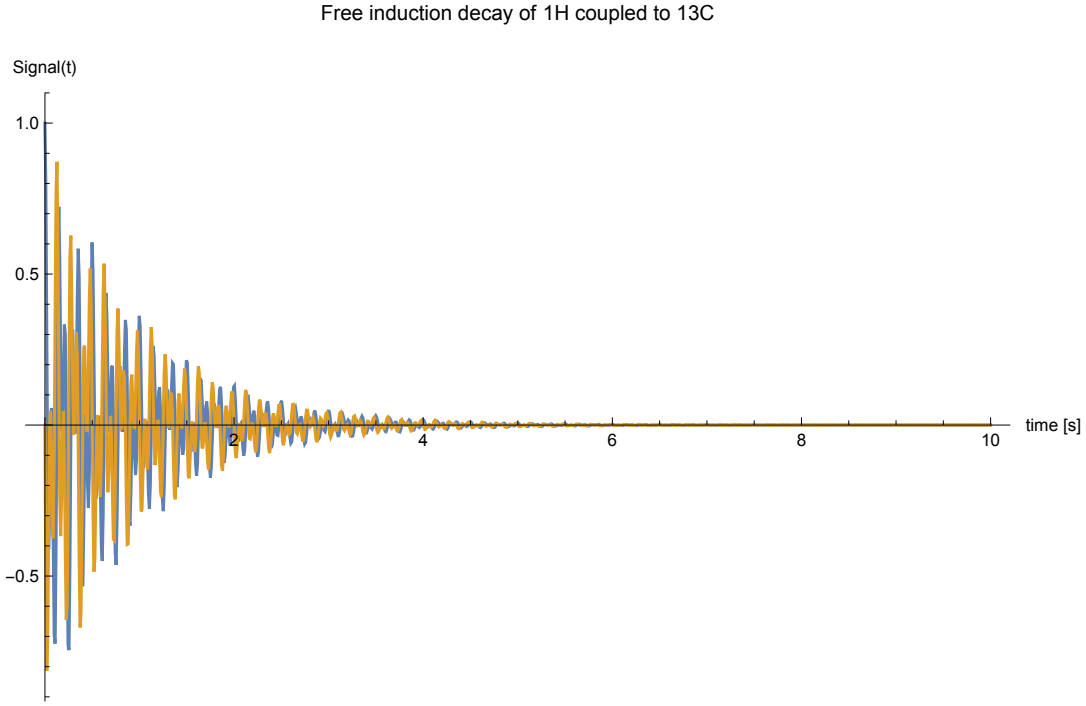
```
The time step is 0.00977517 s

The Nyquist frequency is 51.15 Hz
```

Give numbers for the chemical shift and the J-coupling.

```
Δ$I = 2 π × 10.0;  (* chemical shift [rad/s] *)
J = 2 π × 8.0;
(* heteronuclear scalar coupling constant [rad/s] *)
T2 = 1.0;          (* spin dephasing time [s] *)
```

Plot the real and imaginary part of the signal

```
ListLinePlot[
 {
  {t, S$expt /@ t // Re} // Transpose,
  {t, S$expt /@ t // Im} // Transpose},
 Joined → True,
 PlotRange → All,
 PlotLabel →
  "Free induction decay of 1H coupled to 13C\n",
 AxesLabel → {"time [s]", "Signal(t)"}]
```

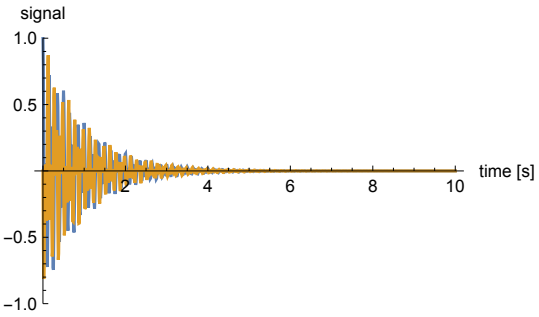


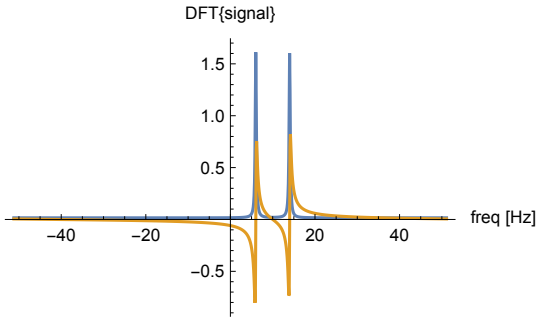
A function to calculate the digital Fourier transform of signal.

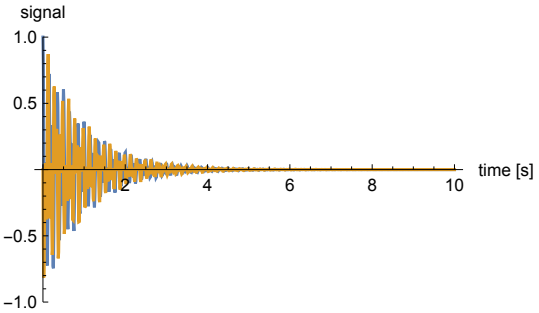
```
DFFT[signal_, t_, query_: True] :=
 (* True to plot both real and imaginary FT parts *)
 Module[{NN},

  NN = Dimensions[signal /@ t][[1]];
  FFTS = RotateRight[Fourier[signal /@ t], NN / 2];

  T = t[[-1]];
  f = Table[jj / T, {jj, -NN / 2, NN / 2 - 1}];

  p1 = ListLinePlot[
    {Transpose[{t, Re[signal /@ t]}],
     Transpose[{t, Im[signal /@ t]}]},
    PlotRange → {-1, 1},
    AxesLabel → {"time [s]", "signal"}];

  If[query == True,

   p2 = ListLinePlot[
     {Transpose[{f, Re[FFTS]}],
      Transpose[{f, Im[FFTS]}] },
     PlotRange → All,
     AxesLabel → {"freq [Hz]", "DFT{signal}"}],

   p2 = ListLinePlot[
     Transpose[{f, Re[FFTS]}],
     PlotRange → All,
     AxesLabel → {"freq [Hz]", "DFT{signal}"}]
  ];

  Show[GraphicsGrid[{{p1, p2}}]]
 ]
```

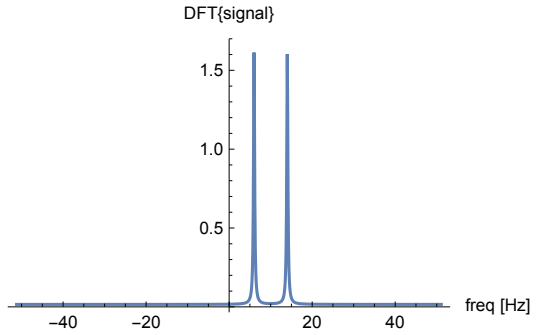
Fourier transform the signal to obtain the spectrum.

```
DFFT[S$expt, t]
DFFT[S$expt, t, False]
```









Clean up