UniDyn--Demo-05.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 under free evolution, a heteronuclear COSY pulse sequence, and a homonuclear pulse sequence.

Instructions

Once you edit the directory path below, you should be able to run the notebook. This notebook will calculate, for two protons that are "J" coupled, the free induction decay and the two-dimensional COrrelation SpectroscopY (COSY) signal. Both the heteronuclear (e.g., a coupled ^{13}C and ^{1}H) and homonuclear (e.g., two coupled ^{1}H's) experiments are written out below. I have highlighted in yellow and grey the places in the file where you may want to change parameters.

Answer these questions after lab.

1) **Free evolution signal**

I give some suggested values for the two proton chemical shifts, the J coupling, and the transverse relaxation time that will give nice looking spectra.

a) Edit the J-coupling to zero and re-run the free-induction decay calculation. Does the spectrum look like you expect? (Reset the J-coupling back to something reasonble before continuing.)

b) Below I plot the real part of the Fourier transform. Now plot the imaginary part and magnitude. Is the resolution better for the real part of the signal or the magnitude part? Can you explain the result qualitatively?

2) **COSY**. Each COSY signal takes a few seconds to calculate.

a) Draw out the pulse sequence for the Heteronuclear COSY and the Homonuclear COSY experiments. How are the experiments different?

b) In each spectum, there is a small signal blip at the center of the plot. This is not due to the spins -- I added that fake signal. How did I add it? Why did I add it?

For both the Heteronuclear COSY and the Homonuclear COSY experiments,

c) Edit the J-coupling to zero and recalculate the spectrum. Does the COSY spectrum look any different? (Reset the J-coupling back to something reasonable before continuing.)

d) Below I plot the two-dimensional Fourier transform of the COSY signal. The signal is not symmetric in $\omega 1$ and $\omega 2$. Can you explain this? Look at the terms in the ***ρ4A***, ***ρ4B***, and ***ρ$4*** signals. How was each of these signals created?s

e) Like you did with the FID, plot the imaginary, real,and magnitude of the two-dimensional Fourier transform of the COSY(t1,t2) signal -- these are the plots **p1**, **p2**, and **p3** returned by the 2D Fourier transform function. Note the peak shapes. For example, are all the peaks positive? To see whether the peaks in the spectrum are positive or negative, you can rotate the plot around using the mouse.

Preliminaries

Set the path to the package

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

EDIT THE FOLLOWING PATH STRING:

```
In[1]:= $NCPath = "/Users/jam99/Dropbox";
        $UniDynPath =
          "/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.

```
In[3]:= $Path = AppendTo[$Path, $NCPath];
        $Path = AppendTo[$Path, $UniDynPath];
        FindFile["UniDyn`"]
        FindFile["NC`"]

Out[5]= /Users/jam99/Dropbox/MarohnGroup__Software_Library/UniDyn/unidyn/UniDyn.m

Out[6]= /Users/jam99/Dropbox/NC/init.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.

```
In[7]:= $VerboseLoad = True;
        Needs["UniDyn`"]

        NC: You are using the version of NCAlgebra which is found in: "  /Users /jam99 /Dropbox /NC/".
```

```
------------------------------------------------------------
NCAlgebra - Version 5.0.6
Compatible with Mathematica Version 10 and above

Authors:

  J. William Helton*
  Mauricio de Oliveira&

* Math, UCSD, La Jolla, CA
& MAE, UCSD, La Jolla, CA

with major earlier contributions by:

  Mark Stankus$
  Robert L. Miller#

$ Math, Cal Poly San Luis Obispo
# General Atomics Corp

Copyright:
  Helton and de Oliveira 2017
  Helton 2002
  Helton and Miller June 1991
  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.

Current primary support is from the
  NSF Division of Mathematical Sciences.

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.

For NCAlgebra updates see:

  www.github.com/NCAlgebra/NC
  www.math.ucsd.edu/~ncalg

------------------------------------------------------------
```

NCAlgebra : All lower cap single letter symbols (e.g. a,b,c,...) were set as noncommutative.

CreateOperator : 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 : 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 }].
- NCSort : NCSort [list] sorts the operators in list into canonical order.
- SortedMult : SortedMult [list] returns Mult [list$ordered], where list$ordered are the elements of list sorted into canonical order.
- MultSort : MultSort [NonCommutativeMultiplyt [list]] returns returns NonCommutativeMultiply [list$ordered], where list$ordered are the elements of list sorted into canonical order.
- Comm : Comm [a,b] calculates the commutator of two operators.
- SpinSingle$CreateOperators : SpinSingle$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.
- OscSingle$CreateOperators : OscSingle$CreateOperators [aL,aR] creates a raising operator aR and a lowering operator aL for single harmonic oscillator and defines the operator commutation relations.
- Evolve : 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 : Evolver [H, t, $\rho(0)$] calculates $\rho(t) = \text{Exp}[-I\,H\,t]\,\rho(0)\,\text{Exp}[+I\,H\,t]$, assuming that H is time independent, according to the commutation rules followed by $\rho(0)$ and H.

Evolver function with simplification

Create a free-evolution operator.

```
In[9]:= SimplifyingEvolver[H_, t_, ρ$0_] :=

  (*   H  = Hamiltonian operator *)
  (*   t  = time varaible *)
  (* ρ$0  = initial density operator *)

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

Implement pulses with simplification

I spins

Evolve the I spins under a short burst of applied on-resonance irradiation, in the delta-function pulse approximation. Specify the total pulse angle and the pulse phase and the PulseI operator evolves the density operator accordingly.

```
In[10]:= Clear[PulseI];
         PulseI[ρ$0_, ϕ_, θ_] :=

          (* ρ$0 = initial density operator *)
          (*   ϕ = pulse phase in radians *)
          (*   θ = pulse angle in radians *)

          ρ$0 //
           SimplifyingEvolver[ ω$I (Cos[ϕ] Ix + Sin[ϕ] Iy), t, #] & /.
            {t → θ / ω$I}
```

The pulse operator distrubutes over Plus, Times, and NoncommutativeMultiply. Scalars in front of operators can be pulled out front.

```
In[12]:= PulseI[ρ$0_Plus, ϕ_, θ_] :=
          Plus @@ (PulseI[#, ϕ, θ] &) /@ List @@ ρ$0
         PulseI[ρ$0_Times, ϕ_, θ_] :=
          Times @@ (PulseI[#, ϕ, θ] &) /@ List @@ ρ$0
         PulseI[ρ$0_NonCommutativeMultiply, ϕ_, θ_] :=
          NonCommutativeMultiply @@ (PulseI[#, ϕ, θ] &) /@ List @@ ρ$0
         PulseI[Times[a_ ? CommutativeQ, ρ$0_], ϕ_, θ_] :=
          a PulseI[ρ$0, ϕ, θ]
```

S spins

Define an analogous pulse operator for the S spins ...

```
In[16]:= Clear[PulseS];
         PulseS[ρ$0_, ϕ_, θ_] :=

          (* ρ$0 = initial density operator *)
          (*   ϕ = pulse phase in radians *)
          (*   θ = pulse angle in radians*)

          ρ$0 //
           SimplifyingEvolver[ω$S (Cos[ϕ] Sx + Sin[ϕ] Sy), t, #] & /.
            {t → θ / ω$S}
```

... with nice distributive and scalar-factoring properties.

```
In[18]:= PulseS[ρ$0_Plus, ϕ_, θ_] :=
          Plus @@ (PulseS[#, ϕ, θ] &) /@ List @@ ρ$0
         PulseS[ρ$0_Times, ϕ_, θ_] :=
          Times @@ (PulseS[#, ϕ, θ] &) /@ List @@ ρ$0
         PulseS[ρ$0_NonCommutativeMultiply, ϕ_, θ_] :=
          NonCommutativeMultiply @@ (PulseS[#, ϕ, θ] &) /@ List @@ ρ$0
         PulseS[Times[a_?CommutativeQ, ρ$0_], ϕ_, θ_] :=
          a PulseS[ρ$0, ϕ, θ]
```

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.   For *Mathematica* and the Evolve[] function to recognize unitary rotations, it is crucially important to define all the non-operator variables in your expressions to be scalars.

```
In[22]:= 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 *)
          t1,          (* time *)
          t2,          (* time *)
          a,           (* Curie Law factor *)
          b,           (* Curie Law factor *)
          ρ$0,         (* initial spin density operator *)
          H            (* spin Hamiltonian *)

         ]

         CreateScalar[Δ$I, Δ$S, J, t, t1, t2, a, b];

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

         CreateOperator[{{Ix, Iy, Iz}, {Sx, Sy, Sz}}];
         SpinSingle$CreateOperators[Ix, Iy, Iz, L = 1 / 2];
         SpinSingle$CreateOperators[Sx, Sy, Sz, L = 1 / 2];

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

Free evolution operator for two coupled spins

Evolve the density operator under the Hamiltonian

$$H = \Delta_I I_z + \Delta_S S_z + J I_z S_z$$

where Δ_I is the I-spin chemical shift, Δ_S is the S-spin chemical shift, and J is the through-bond coupling between the I spin and the S spin. The Evolve[] operator cannnot evolve the density operator all in one step, so we use the fact that all three terms in the above Hamiltonian commute with each other. This commutation relationship allows us to evolve the density stepwise, by one of the operators at a time.

```
In[28]:= Clear[FreeEvolution];
         FreeEvolution[ρ_, t_] :=

           (* ρ = density operator *)
           (* t = time [s] *)

           ((ρ // SimplifyingEvolver[Δ$I Iz, t, #] & //
                  SimplifyingEvolver[ Δ$S Sz, t, #] & //
                SimplifyingEvolver[ J Iz ** Sz, t, #] &) //
             NCExpand // MultSort) // NCExpand
```

The FreeEvolution operator distributes over Plus, Times, and NoncommutativeMultiply. Scalars in front of operators can be pulled out front.

```
In[30]:= FreeEvolution[ρ_Plus, t_] :=
          Plus @@ (FreeEvolution[#, t] &) /@ List @@ ρ
         FreeEvolution[ρ_Times, t_] :=
          Times @@ (FreeEvolution[#, t] &) /@ List @@ ρ
         FreeEvolution[ρ_NonCommutativeMultiply, t_] :=
          NonCommutativeMultiply @@ (FreeEvolution[#, t] &) /@ List @@ ρ
         FreeEvolution[Times[a_?CommutativeQ, ρ_], t_] :=
          a FreeEvolution[ρ, t]
```

Load Spin 1/2 matrices

```
In[34]:= << Matrices--two-spin-half.m

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

This set of rules will substitute appropriate 4 x 4 matrices for the symbolic spin operators above.

```
In[35]:= SubstituteSpinMatrices =
           {Ix → mIx, Iy → mIy, Iz → mIz, Sx → mSx, Sy → mSy, Sz → mSz};
```

Utility functions for one-dimensional data

Make an array of time data

```
In[36]:= TimeArray[NN_, T_] :=
          (* NN = number of points      *)
          (* T = total acquisition time *)
         Module[{dt, t},
          dt = T / (NN - 1);
          t = Table[ii * dt, {ii, 0, NN - 1}];
          Return[t]]
```

Print out the time step and the Nyquist frequency

```
In[37]:= TimeArrayReport[t_] :=
          (* t = List of time points *)
         Module[{dt},
          dt = t[[2]] - t[[1]];
          Print["The time step is ", dt, " s"];
          Print["The Nyquist frequency is ", 1 / (2 * dt), " Hz"]]
```

Calculate the digital Fourier transform of a one-dimensional time-domain signal and plot it.

```
In[38]:= DFFT[signal_, t_, query_ : True] :=


          (* signal = a function that computes the signal *)
          (*      t = a list of time points *)
          (*  query = True or False *)
          (*          True to plot the real part,
         imaginary part, and magnitude of the FT *)
```

```
 (*          False to
 plot only the eal part of the FT *)

Module[{NN, (* number of data points *)
        T,  (* total acquisition time;
  assumes data starts at zero *)
        f   (* array of frequency points *)
 },

 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]}],
     Transpose[{f, Abs[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}}]]
  ]
```

Calculate the digital Fourier transform of a one-dimensional time-domain signal and plot it.

```
In[39]:= DFFT$all[signal_, t_] :=

   (* signal = A function that computes the signal *)
   (*      t = A list of time points *)
   (*  query = True or False *)
   (*          True to plot the real part,
  imaginary part, and magnitude of the FT *)
   (*          False to
   plot only the real part of the FT *)

  Module[{NN, (* number of data points *)
          T,  (* total acquisition time;
    assumes data starts at zero *)
          f   (* array of frequency points *)
   },

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

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

  p3 = ListLinePlot[
    Transpose[{f, Abs[FFTS]}],
    PlotRange → All,
    AxesLabel → {"freq [Hz]", "Abs[DFT{signal}]"}];

  Show[GraphicsGrid[Transpose[{{p1, p2, p3}}]]]
 ]
```

Utility function to reduce density operator to a matrix

```
In[40]:= MakeNumerical[expression_] :=
          ReplaceAll[
           ReplaceAll[expression, NonCommutativeMultiply → Dot],
           SubstituteSpinMatrices]
```

Free-evolution signal

Edit the chemical shifts and the J-couplings here

```
In[41]:= HamiltonianParameters$coupled =

           {Δ$I → 2 π * 100, (* I-spin chemical shift [radians] *)
            Δ$S → 2 π * 300,
            (* S-spin chemical shift [radians] *)
            J → 2 π * 50,     (* scalar coupling [radians] *)
            T2 → 0.3          (* spin dephasing time [s] *)
            };
```

Make a copy of the parameters with the scalar coupling set to zero, e.g., with the two spins uncoupled. Useful for testing.

```
In[42]:= HamiltonianParameters$uncoupled =
           HamiltonianParameters$coupled;
         HamiltonianParameters$uncoupled⟦3⟧ = J → 0;
```

Evolve the initial density operator $I_x + S_x$ under the coupled-spin Hamiltonian. Add an exponential decay so we get a reasonable looking signal. Calculate the signal corresponding to observation of the complex signal $(I_x + S_x) + \mathrm{I}\,(I_y + S_{xy})$ using the matrix form of the operators.

```
In[44]:= ρ = FreeEvolution[Ix + Sx, t] Exp[-t / T2];
         Signal[t_] =
           MakeNumerical[Tr[ρ ** ((Ix + Sx) + I (Iy + Sy))]] /.
            HamiltonianParameters$coupled;
```

Check that the signal is a scalar.

In[46]:=
```
Signal[t]
```

Out[46]=
$$e^{-3.33333\,t}\left(\frac{1}{2}\,\mathrm{Cos}[50\,\pi\,t]\,\mathrm{Cos}[200\,\pi\,t]+\frac{1}{2}\,i\,\mathrm{Cos}[200\,\pi\,t]\,\mathrm{Sin}[50\,\pi\,t]+\frac{1}{2}\,i\,\mathrm{Cos}[50\,\pi\,t]\,\mathrm{Sin}[200\,\pi\,t]-\frac{1}{2}\,\mathrm{Sin}[50\,\pi\,t]\,\mathrm{Sin}[200\,\pi\,t]\right)+$$
$$e^{-3.33333\,t}\left(\frac{1}{2}\,\mathrm{Cos}[50\,\pi\,t]\,\mathrm{Cos}[600\,\pi\,t]+\frac{1}{2}\,i\,\mathrm{Cos}[600\,\pi\,t]\,\mathrm{Sin}[50\,\pi\,t]+\frac{1}{2}\,i\,\mathrm{Cos}[50\,\pi\,t]\,\mathrm{Sin}[600\,\pi\,t]-\frac{1}{2}\,\mathrm{Sin}[50\,\pi\,t]\,\mathrm{Sin}[600\,\pi\,t]\right)+$$
$$e^{-3.33333\,t}\left(\frac{1}{2}\,\mathrm{Cos}[50\,\pi\,t]\,\mathrm{Cos}[200\,\pi\,t]+\frac{1}{2}\,\mathrm{Cos}[50\,\pi\,t]\,\mathrm{Cos}[600\,\pi\,t]-\frac{1}{2}\,i\,\mathrm{Cos}[200\,\pi\,t]\,\mathrm{Sin}[50\,\pi\,t]-\frac{1}{2}\,i\,\mathrm{Cos}[600\,\pi\,t]\,\mathrm{Sin}[50\,\pi\,t]+\frac{1}{2}\,i\,\mathrm{Cos}[50\,\pi\,t]\,\mathrm{Sin}[200\,\pi\,t]+\frac{1}{2}\,\mathrm{Sin}[50\,\pi\,t]\,\mathrm{Sin}[200\,\pi\,t]+\frac{1}{2}\,i\,\mathrm{Cos}[50\,\pi\,t]\,\mathrm{Sin}[600\,\pi\,t]+\frac{1}{2}\,\mathrm{Sin}[50\,\pi\,t]\,\mathrm{Sin}[600\,\pi\,t]\right)$$

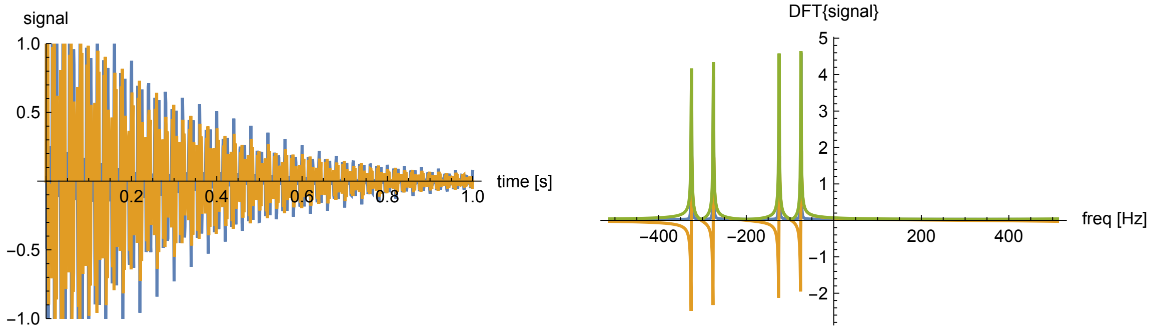
Create an array of 1024 time points.

In[47]:=
```
t$List = TimeArray[2^10, 1.0];
t$List // TimeArrayReport
```

```
The time step is 0.000977517 s

The Nyquist frequency is 511.5 Hz
```

Calculate the time-domain signal. Fourier transform it to reveal the spectrum of the two coupled spins.

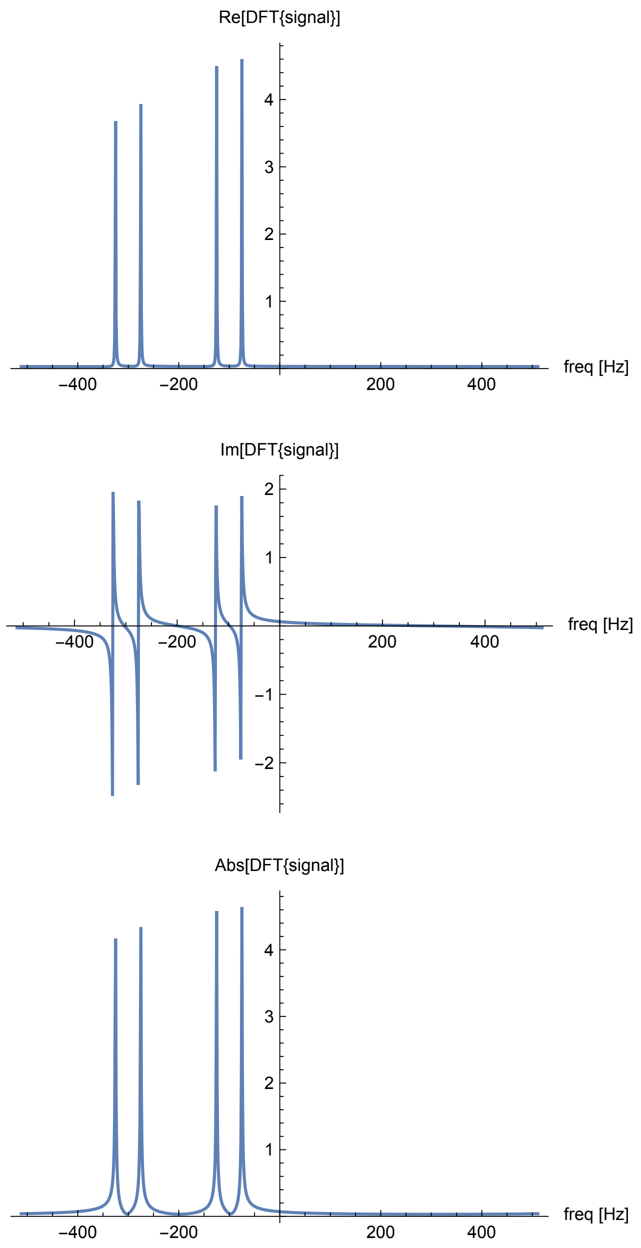
In[49]:=
```
DFFT[Signal, t$List, True]
```

Out[49]=


Let's plot the real, imaginary, and absolute value signals separately. Notice how broad the imaginary-channel and absolute-value channel signals are.

```
In[50]:= DFFT$all[Signal, t$List]
```

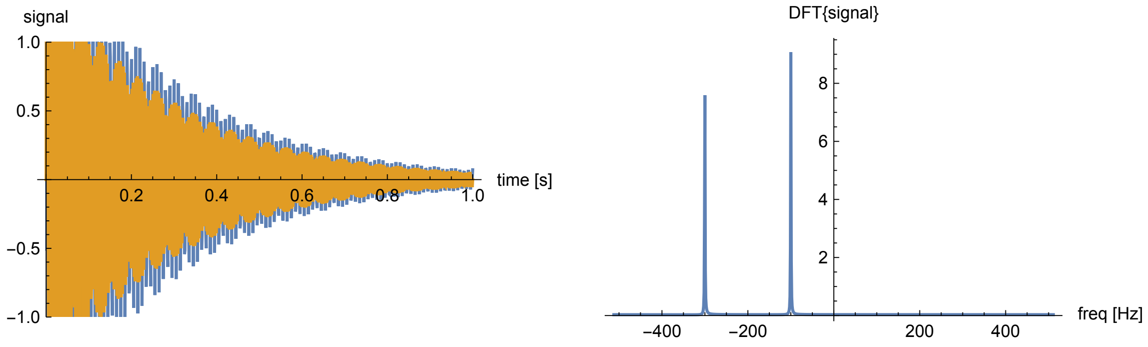
Out[50]=



Recalculate the signal with the spin-spin coupling dialed to zero. Note how the doublet pattern in the spectrum vanishes.

In[51]:=
```
ρ = FreeEvolution[(Ix + Sx) , t] Exp[-t / T2];
Signal[t_] =
  MakeNumerical[Tr[ρ ** ((Ix + Sx) + I (Iy + Sy))]] /.
   HamiltonianParameters$uncoupled;
DFFT[Signal, t$List, False]
```

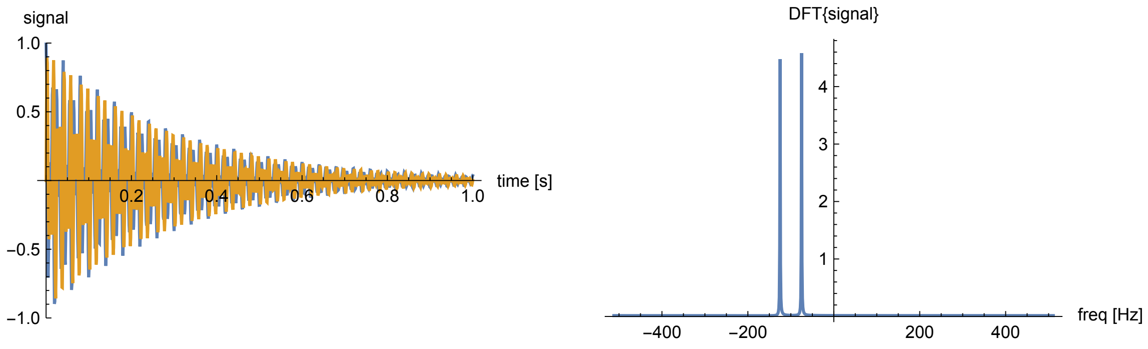
Out[53]=



Evolve just I_x. We see just one of the doublets.

In[54]:=
```
ρ = FreeEvolution[Ix , t] Exp[-t / T2];
Signal[t_] =
  MakeNumerical[Tr[ρ ** ((Ix + Sx) + I (Iy + Sy))]] /.
   HamiltonianParameters$coupled;
DFFT[Signal, t$List, False]
```

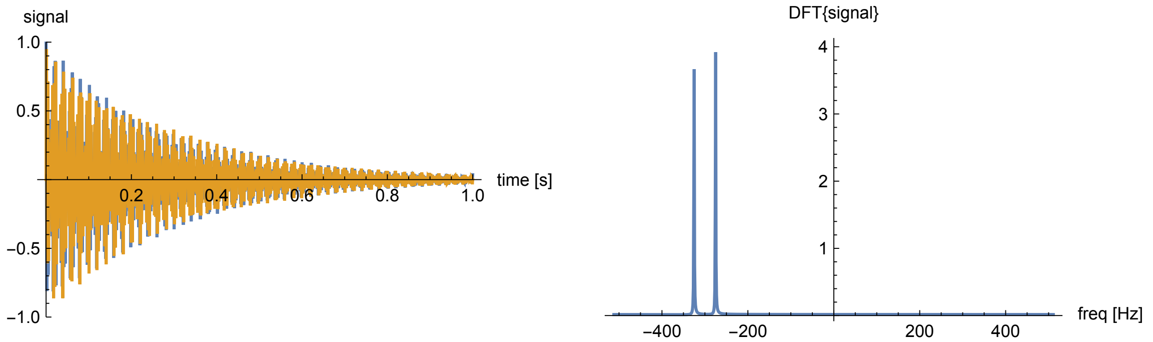
Out[56]=



Evolve just S_x. See see just the other doublet.

In[57]:=
```
ρ = FreeEvolution[Sx , t] Exp[-t / T2];
Signal[t_] =
  MakeNumerical[Tr[ρ ** ((Ix + Sx) + I (Iy + Sy))]] /.
   HamiltonianParameters$coupled;
DFFT[Signal, t$List, False]
```

Out[59]=



INEPT polarization transfer

Here INEPT is an acronym that stands for *insensitive nuclei enhanced by polarization transfer*. Thinking through the INEPT experiment helps you understand how two-dimensional spectroscopy works.

The INEPT experiment shows how a J coupling can be harnessed to transfer spin polarization from a nucleus with a large magnetic moment -- ^{1}H, the I-spin here -- to a nucleus with a small magnetic moment -- ^{13}C, the S-spin here. At thermal equilibrium the initial density operator is

$$\rho_{\text{equil}} = a\, I_z + b\, S_z$$

with *a* and *b* the Curie factors for each of the spins. We worked through these Curie-law factors in detail on a homework. To discuss the INEPT experiment it is enough to know that the factor b is about 1/16 the size of the a factor:

In[60]:=
```
SubstitutePrefactors = {a → 1, b → (1/4)^2};
```

In other words, the thermal-equilibrium ^{13}C signal is about 1/16 as large as the thermal-equilibrium ^{1}H signal in a magnetic resonance experiment.

The usual ^{13}C spectrum

Let’s calculate the usual ^{13}C signal. Deliver a $\pi/2$ pulse to the ^{13}C spins and then evolve the spins under the Hamiltonian

$$H = \Delta_I I_z + \Delta_S S_z + J I_z S_z$$

For simplicity we will work on resonance where $\Delta_I \to 0$ and $\Delta_S \to 0$.

```
In[61]:= ρ$0  = a Iz + b Sz //  PulseS[#, π / 2, π / 2] &;
         ρ$1 =
          (ρ$0 //  FreeEvolution[#, t] &  // NCExpand) /. {Δ$I → 0, Δ$S → 0}
```

Out[62]= $a\,Iz + b\,Sx\,\mathrm{Cos}\left[\frac{J\,t}{2}\right] + 2\,b\,Iz ** Sy\,\mathrm{Sin}\left[\frac{J\,t}{2}\right]$

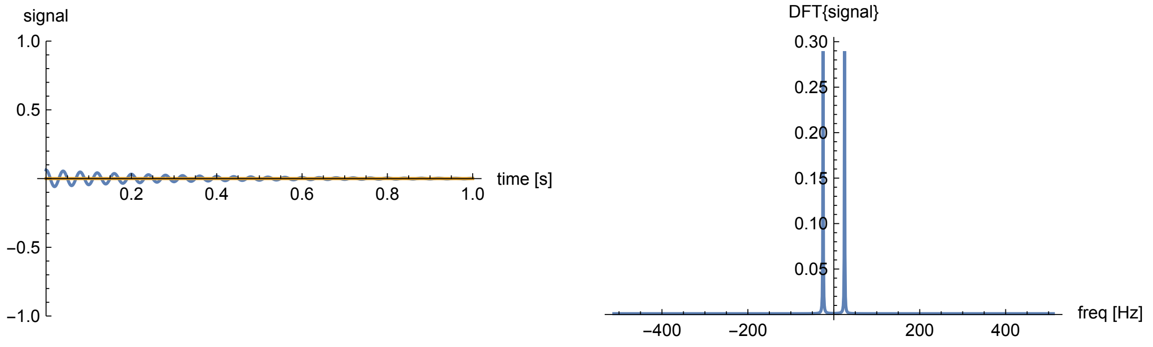
We detect the complex magnetization and Fourier transform it to obtain the ^{13}C spectrum.

```
In[63]:= Signal[t_] = (MakeNumerical[Tr[ρ$1 ** (Sx + I Sy)] Exp[-t / T2]] /.
             HamiltonianParameters$coupled ) /. SubstitutePrefactors
```

Out[63]= $\frac{1}{16}\,e^{-3.33333\,t}\,\mathrm{Cos}[50\,\pi\,t]$

```
In[64]:= DFFT[Signal, t$List, False]
```

Out[64]=



The spectrum shows a symmetric doublet, due the J-coupling, that is centered near zero frequency. The height of each peak in the spectrum is approximately ~0.28.

The ^{13}C spectrum following polarization transfer from ^{1}H

The INEPT experiment consists of a string of pulses applied to the I-spins and the S-spins.

I spins: $(\pi/2)_y - \tau - (\pi)_x - \tau - (\pi/2)_x$

S spins: $(\pi)_x - \tau - (\pi/2)_y$ – detect

with the time delay set to $\tau = 2\pi/(4\,J)$, with J the scalar coupling. Now let’s calculate the usual ^{13}C signal following the INEPT polarization-transfer sequence. For simplicity we will again work on resonance where $\Delta_I \to 0$ and $\Delta_S \to 0$. in the calculation below I print out the density operator at each step in the evolution.

```
In[65]:= ρ$0  = a Iz + b Sz // PulseI[#, π / 2, π / 2] &
ρ$1 = (ρ$0 // FreeEvolution[#, t] & // NCExpand) /. {Δ$I → 0}
ρ$2 = ρ$1 // PulseI[#, 0, π] & // PulseS[#, 0, π] &
ρ$3 = (ρ$2 // FreeEvolution[#, t] & // NCExpand) /. {Δ$I → 0}
ρ$4 = ρ$3 // PulseI[#, 0, π / 2] & // PulseS[#, π / 2, π / 2] &
```

Out[65]= a Ix + b Sz

Out[66]= $b\,\mathrm{Sz} + a\,\mathrm{Ix}\,\mathrm{Cos}\left[\frac{J\,t}{2}\right] + 2\,a\,\mathrm{Iy} ** \mathrm{Sz}\,\mathrm{Sin}\left[\frac{J\,t}{2}\right]$

Out[67]= $-b\,\mathrm{Sz} + a\,\mathrm{Ix}\,\mathrm{Cos}\left[\frac{J\,t}{2}\right] + 2\,a\,\mathrm{Iy} ** \mathrm{Sz}\,\mathrm{Sin}\left[\frac{J\,t}{2}\right]$

Out[68]= $-b\,\mathrm{Sz} + a\,\mathrm{Ix}\,\mathrm{Cos}\left[\frac{J\,t}{2}\right]^2 + 4\,a\,\mathrm{Cos}\left[\frac{J\,t}{2}\right]\,\mathrm{Iy} ** \mathrm{Sz}\,\mathrm{Sin}\left[\frac{J\,t}{2}\right] - a\,\mathrm{Ix}\,\mathrm{Sin}\left[\frac{J\,t}{2}\right]^2$

Out[69]= $-b\,\mathrm{Sx} + a\,\mathrm{Ix}\,\mathrm{Cos}\left[\frac{J\,t}{2}\right]^2 + 4\,a\,\mathrm{Cos}\left[\frac{J\,t}{2}\right]\,\mathrm{Iz} ** \mathrm{Sx}\,\mathrm{Sin}\left[\frac{J\,t}{2}\right] - a\,\mathrm{Ix}\,\mathrm{Sin}\left[\frac{J\,t}{2}\right]^2$

The signal in the “regular” experiment arose from the term

$$b\,\mathrm{Sx}\,\mathrm{Cos}\left[\frac{J\,t}{2}\right]$$

in the density operator. After the series of pulses in the INEPT experiment we now have a term in the density operator given by

$$4\,a\,\mathrm{Cos}\left[\frac{J\,t}{2}\right]\,\mathrm{Iz} ** \mathrm{Sx}\,\mathrm{Sin}\left[\frac{J\,t}{2}\right] \to 2\,a\,\mathrm{Iz} ** \mathrm{Sx}$$

where the term following the arrow is what we get by setting the delay time to $2\,\pi/(4\,J)$. This term arose from the *initial I-spin signal* evolving under the spin-spin coupling to give a *mixed I-spin & S-spin signal*. Further evolution under the J coupling Hamiltonian will give rise a a *pure S-spin signal*.

```
In[70]:= ρ$5 = ρ$4 /. {t → 2 π / (4 J)} // FreeEvolution[#, t] & //
           NCExpand (* Why is an extra NCExpand needed here? *)
```

Out[70]= $-b\,Sx\,\mathrm{Cos}\left[\frac{J\,t}{2}\right]\mathrm{Cos}[t\,\Delta\$S] + 2\,a\,\mathrm{Cos}\left[\frac{J\,t}{2}\right]\mathrm{Cos}[t\,\Delta\$S]\,Iz ** Sx + a\,Sy\,\mathrm{Cos}[t\,\Delta\$S]\,\mathrm{Sin}\left[\frac{J\,t}{2}\right] - 2\,b\,\mathrm{Cos}[t\,\Delta\$S]\,Iz ** Sy\,\mathrm{Sin}\left[\frac{J\,t}{2}\right] - b\,Sy\,\mathrm{Cos}\left[\frac{J\,t}{2}\right]\mathrm{Sin}[t\,\Delta\$S] + 2\,a\,\mathrm{Cos}\left[\frac{J\,t}{2}\right]Iz ** Sy\,\mathrm{Sin}[t\,\Delta\$S] - a\,Sx\,\mathrm{Sin}\left[\frac{J\,t}{2}\right]\mathrm{Sin}[t\,\Delta\$S] + 2\,b\,Iz ** Sx\,\mathrm{Sin}\left[\frac{J\,t}{2}\right]\mathrm{Sin}[t\,\Delta\$S]$

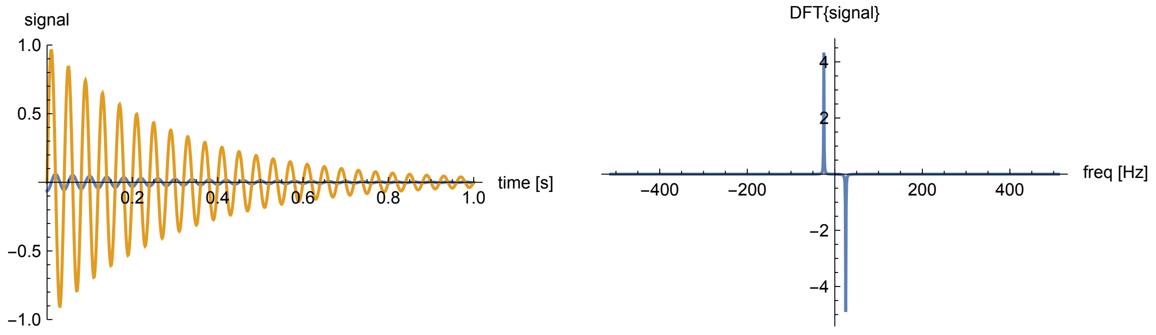
Remarkably, **the ^{13}C signal is now approximately 16x larger**. This signal increase corresponds to a $16^2 = 256$-fold savings in signal averaging time.

```
In[71]:= Signal[t_] = (MakeNumerical[Tr[ρ$5.(Sx + I Sy)] Exp[-t / T2]] /.
              {Δ$I → 0, Δ$S → 0}) /.
            HamiltonianParameters$coupled /. SubstitutePrefactors
```

Out[71]= $e^{-3.33333\,t}\left(-\frac{1}{16}\mathrm{Cos}[50\,\pi\,t] + i\,\mathrm{Sin}[50\,\pi\,t]\right)$

```
In[72]:= DFFT[Signal, t$List, False]
```

Out[72]=



Utility functions for two-dimensional data

Function to take the 2D Fourier Transform

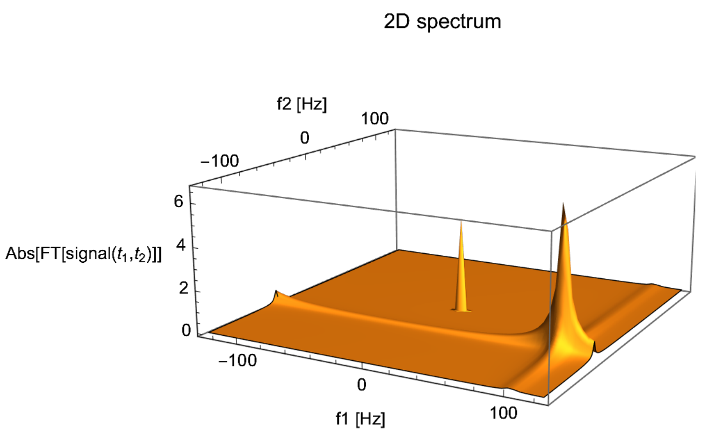
```
In[73]:= DFFT2[signal_, t1_, t2_, query_ : False] :=

           (* signal = A function that computes the signal *)
           (*     t1 = A list of time points *)
           (*     t2 = A list of time points *)
           (*  query = True or False,
whether to print out diagnostic messages *)

Module[{t1$NN,
  (* number of time points in the indirect, t1 dimension *)
     t2$NN,
  (* number of time points in the direct, t2 dimension *)
     f1,
  (* the frequency axis in the indirect dimension *)
     f2
  (* the frequency axis in the indirect dimension *)
 },

 (* Too fancy: signal$List = Outer[N[signal[#1,#2]]&,t1,t2]; *)

 t1$NN = Dimensions[t1]⟦1⟧;
 t2$NN = Dimensions[t2]⟦1⟧;

 signal$List = Table[signal[t1⟦i$1⟧, t2⟦i$2⟧] // N,
   {i$2, 1, t2$NN}, {i$1, 1, t1$NN}];

 If[query == True,
  Print["Number of t1 points = ", t1$NN];
  Print["Number of t2 points = ", t2$NN];
  Print["Dimensions of signal$List = ",
   Dimensions[signal$List ]];
 ];

 signal$List$FT =
  RotateLeft[Fourier[signal$List ], {t2$NN / 2, t1$NN / 2}];

 f1 = Table[jj / t1⟦-1⟧, {jj, -t1$NN / 2, t1$NN / 2 - 1}];
 f2 = Table[jj / t2⟦-1⟧, {jj, -t2$NN / 2, t2$NN / 2 - 1}];

 SetOptions[ListPlot3D,
  PlotRange → All,
   DataRange → {{f1⟦1⟧, f1⟦t1$NN⟧}, {f2⟦1⟧, f2⟦t2$NN⟧}},
   PlotLabel → "2D spectrum\n\n",
   Mesh → False];

  p1 = ListPlot3D[Re[signal$List$FT ],
    AxesLabel → {"f1 [Hz]", "f2 [Hz]", "Re[FT[signal(t1,t2)]]"},
    ViewPoint → {1.561`, -2.881`, 0.845`}];

  p2 = ListPlot3D[Im[signal$List$FT ],
    AxesLabel → {"f1 [Hz]", "f2 [Hz]", "Im[FT[signal(t1,t2)]]"},
    ViewPoint → {1.561`, -2.881`, 0.845`}];

  p3 = ListPlot3D[Abs[signal$List$FT ],
    AxesLabel → {"f1 [Hz]", "f2 [Hz]", "Abs[FT[signal(t1,t2)]]"},
    ViewPoint → {1.561`, -2.881`, 0.845`}];

  Return[{p1, p2, p3}]
 ]
```

Test case #1

```
In[74]:= t1$List = TimeArray[2^6, 0.25];
         t2$List = TimeArray[2^7, 0.50];

In[76]:= t1$List // TimeArrayReport
         t2$List // TimeArrayReport

         The time step is 0.00396825 s

         The Nyquist frequency is 126. Hz

         The time step is 0.00393701 s

         The Nyquist frequency is 127. Hz

In[78]:= Signal2D[t1_, t2_] = Exp[-I 2 π 100 t1]
            Exp[I 2 π 50 t2] Exp[-t1 / 0.08 ] Exp[-t2 / 0.15] + 0.05;
         {p1, p2, p3} = DFFT2[Signal2D, t1$List, t2$List, True];

         p3
```

```
Number of t1 points = 64
Number of t2 points = 128
Dimensions of signal$List = {128, 64}
```

Out[80]=

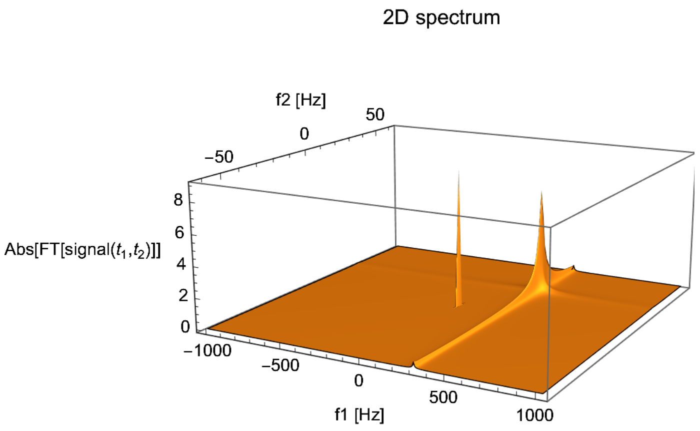


Test Case #2

In[81]:=
```
t1$List = TimeArray[2^9, 0.25];
t2$List = TimeArray[2^6, 0.50];

t1$List // TimeArrayReport
t2$List // TimeArrayReport

Signal2D[t1_, t2_] = Exp[-I 2 π 300 t1]
    Exp[-I 2 π 30 t2] Exp[-t1 / 0.05 ] Exp[-t2 / 0.10] + 0.05;
{p1, p2, p3} = DFFT2[Signal2D, t1$List, t2$List, True];

p3
```

```
The time step is 0.000489237 s

The Nyquist frequency is 1022. Hz

The time step is 0.00793651 s

The Nyquist frequency is 63. Hz

Number of t1 points = 512

Number of t2 points = 64

Dimensions of signal$List = {64, 512}
```

Out[87]=



Heteronuclear COSY signal

This is the signal that we worked out in class

Set the couplings and number of points

Edit the chemical shifts and the J-couplings here

In[88]:=

```
HamiltonianParameters$coupled =

  {Δ$I → 2 π * 100,  (* I-spin chemical shift [radians] *)
   Δ$S → 2 π * 150,  (* S-spin chemical shift [radians] *)
   J → 2 π * 40,     (* scalar coupling [radians] *)
   T2 → 0.10        (* spin dephasing time [s] *)
    };
```

Edit the number of points at the total acquisition time here

```
In[89]:= t1$List = TimeArray[2^7, 0.25];
         t2$List = TimeArray[2^7, 0.25];
```

Make a copy of the parameters with the scalar coupling set to zero, e.g., with the two spins uncoupled. Useful for testing.

```
In[91]:= HamiltonianParameters$uncoupled =
           HamiltonianParameters$coupled;
         HamiltonianParameters$uncoupled[[3]] = J → 0;
```

Some checks

Check out the Nyquist frequencies

```
In[93]:= t1$List // TimeArrayReport
         t2$List // TimeArrayReport

The time step is 0.0019685 s

The Nyquist frequency is 254. Hz

The time step is 0.0019685 s

The Nyquist frequency is 254. Hz
```

Evolve the density operator

Compute evolution. This computation takes many seconds.

```
In[95]:= ρ$0 = Iz + Sz;

         (* Initial (π/2)y pulse*)

         ρ$1$A = ρ$0 // PulseS[#, π / 2, π / 2] &;
         ρ$2$A = ρ$1$A // FreeEvolution[#, t1] & ;
         ρ$3$A = ρ$2$A // PulseS[#, π / 2, π / 2] & // PulseI[#, π / 2, π / 2] &;
         ρ$4$A = ρ$3$A // FreeEvolution[#, t2] & // NCExpand // MultSort;

         (* Initial (π/2)x pulse*)

         ρ$1$B = ρ$0 // PulseS[#, 0, π / 2] &;
         ρ$2$B = ρ$1$B // FreeEvolution[#, t1] & ;
         ρ$3$B = ρ$2$B // PulseS[#, π / 2, π / 2] & // PulseI[#, π / 2, π / 2] &;
         ρ$4$B = ρ$3$B // FreeEvolution[#, t2] & // NCExpand // MultSort;
```

Examine the two signals and combine them

```
In[104]:= MakeNumerical[Tr[ρ$4$A ** (Ix + I Iy)]] // FullSimplify
```

Out[104]= $e^{i\, t2\, \Delta\$I} \left(\text{Cos}\left[\frac{J\, t2}{2}\right] + i\, \text{Sin}\left[\frac{J\, t1}{2}\right] \text{Sin}\left[\frac{J\, t2}{2}\right] \text{Sin}[t1\, \Delta\$S]\right)$

```
In[105]:= MakeNumerical[Tr[ρ$4$B ** (Ix + I Iy)]] // FullSimplify
```

Out[105]= $e^{i\, t2\, \Delta\$I} \left(\text{Cos}\left[\frac{J\, t2}{2}\right] - i\, \text{Cos}[t1\, \Delta\$S]\, \text{Sin}\left[\frac{J\, t1}{2}\right] \text{Sin}\left[\frac{J\, t2}{2}\right]\right)$

```
In[106]:= ρ$4 = ρ$4$A + ρ$4$B;
          MakeNumerical[Tr[ρ$4 ** (Ix + I Iy)]] // FullSimplify // Expand
```

Out[107]= $2\, e^{i\, t2\, \Delta\$I}\, \text{Cos}\left[\frac{J\, t2}{2}\right] - i\, e^{i\, t2\, \Delta\$I}\, \text{Cos}[t1\, \Delta\$S]\, \text{Sin}\left[\frac{J\, t1}{2}\right] \text{Sin}\left[\frac{J\, t2}{2}\right] + i\, e^{i\, t2\, \Delta\$I}\, \text{Sin}\left[\frac{J\, t1}{2}\right] \text{Sin}\left[\frac{J\, t2}{2}\right] \text{Sin}[t1\, \Delta\$S]$

Compute the signal and apply the 2D FT

This computation should just take a few seconds.

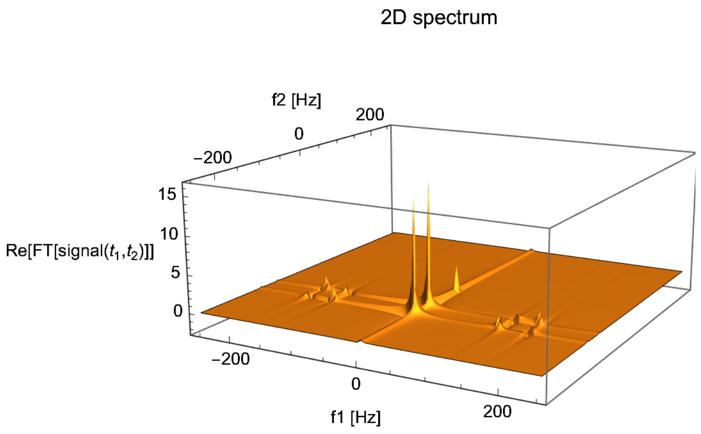
```
In[108]:= Signal2D[t1_, t2_] =
            (MakeNumerical[Tr[ρ$4 ** (Ix + I Iy)]] Exp[-t1 / T2]
                  Exp[-t2 / T2] + 0.025) /.
               HamiltonianParameters$coupled // Simplify // N;
```

Fourier transform the signal.   This takes a minute or so.

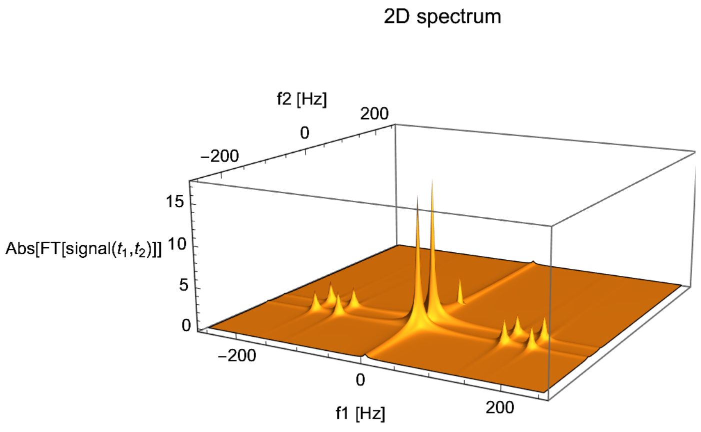
```
In[109]:= {p1, p2, p3} = DFFT2[Signal2D, t1$List, t2$List];
```

```
In[110]:= p1
          p3
```

Out[110]=



Out[111]=



Homonuclear COSY signal

This is the signal you get if you pulse and detect *both* nuclei at the same time.

Set the couplings and number of points

Edit the chemical shifts and the J-couplings here

In[112]:=
```
HamiltonianParameters$coupled =

  {Δ$I → 2 π * 50, (* I-spin chemical shift [radians] *)
   Δ$S → 2 π * 150,  (* S-spin chemical shift [radians] *)
   J → 2 π * 23,     (* scalar coupling [radians] *)
   T2 → 0.10        (* spin dephasing time [s] *)
   };
```

Edit the number of points at the total acquisition time here

In[113]:=
```
t1$List = TimeArray[2^7, 0.25];
t2$List = TimeArray[2^7, 0.25];
```

Make a copy of the parameters with the scalar coupling set to zero, e.g., with the two spins uncoupled. Useful for testing.

In[115]:=
```
HamiltonianParameters$uncoupled =
  HamiltonianParameters$coupled;
HamiltonianParameters$uncoupled[[3]] = J → 0;
```

Some checks

Check out the Nyquist frequencies

In[117]:=
```
t1$List // TimeArrayReport
t2$List // TimeArrayReport
```

```
The time step is 0.0019685 s

The Nyquist frequency is 254. Hz

The time step is 0.0019685 s

The Nyquist frequency is 254. Hz
```

Evolve the density operator

Compute evolution. This computation can take up to a minute.

```
In[119]:= ρ$0 = Iz + Sz;

 (* Initial (π/2)y pulses*)

 ρ$1$A = ρ$0 // PulseS[#, π / 2, π / 2] & // PulseI[#, π / 2, π / 2] & //
     NCExpand // MultSort;
 ρ$2$A = ρ$1$A // FreeEvolution[#, t1] & // NCExpand // MultSort;
 ρ$3$A = ρ$2$A // PulseS[#, π / 2, π / 2] & // PulseI[#, π / 2, π / 2] & //
     NCExpand // MultSort;
 ρ$4$A = ρ$3$A // FreeEvolution[#, t2] & // NCExpand // MultSort;

 (* Initial (π/2)x pulses*)

 ρ$1$B =
   ρ$0 // PulseS[#, 0, π / 2] & // PulseI[#, 0, π / 2] & // NCExpand //
    MultSort;
 ρ$2$B = ρ$1$B // FreeEvolution[#, t1] & // NCExpand // MultSort;
 ρ$3$B = ρ$2$B // PulseS[#, π / 2, π / 2] & // PulseI[#, π / 2, π / 2] & //
     NCExpand // MultSort;
 ρ$4$B = ρ$3$B // FreeEvolution[#, t2] & // NCExpand // MultSort;
```

Examine the two signals and combine them

```
In[128]:= MakeNumerical[Tr[ρ$4$A ** ((Ix + Sx) + I (Iy + Sy))]] // Simplify
```

Out[128]= $i\left(\mathrm{Cos}\left[\frac{J\,t1}{2}\right]\mathrm{Cos}\left[\frac{J\,t2}{2}\right](\mathrm{Cos}[t2\,\Delta\$I]\,\mathrm{Sin}[t1\,\Delta\$I] + i\,\mathrm{Sin}[t1\,\Delta\$I]\,\mathrm{Sin}[t2\,\Delta\$I] + \mathrm{Sin}[t1\,\Delta\$S]\,(\mathrm{Cos}[t2\,\Delta\$S] + i\,\mathrm{Sin}[t2\,\Delta\$S])) + \mathrm{Sin}\left[\frac{J\,t1}{2}\right]\mathrm{Sin}\left[\frac{J\,t2}{2}\right](\mathrm{Cos}[t2\,\Delta\$S]\,\mathrm{Sin}[t1\,\Delta\$I] + \mathrm{Cos}[t2\,\Delta\$I]\,\mathrm{Sin}[t1\,\Delta\$S] + i\,(\mathrm{Sin}[t2\,\Delta\$I]\,\mathrm{Sin}[t1\,\Delta\$S] + \mathrm{Sin}[t1\,\Delta\$I]\,\mathrm{Sin}[t2\,\Delta\$S]))\right)$

```
In[129]:= MakeNumerical[Tr[ρ$4$B ** ((Ix + Sx) + I (Iy + Sy))]] // Simplify
```

Out[129]= $\mathrm{Sin}\left[\frac{J\,t1}{2}\right]\mathrm{Sin}\left[\frac{J\,t2}{2}\right](-i\,\mathrm{Cos}[t2\,\Delta\$I]\,\mathrm{Cos}[t1\,\Delta\$S] + \mathrm{Cos}[t1\,\Delta\$S]\,\mathrm{Sin}[t2\,\Delta\$I] + \mathrm{Cos}[t1\,\Delta\$I]\,(-i\,\mathrm{Cos}[t2\,\Delta\$S] + \mathrm{Sin}[t2\,\Delta\$S])) + \mathrm{Cos}\left[\frac{J\,t1}{2}\right]\mathrm{Cos}\left[\frac{J\,t2}{2}\right](\mathrm{Cos}[t1\,\Delta\$I]\,(-i\,\mathrm{Cos}[t2\,\Delta\$I] + \mathrm{Sin}[t2\,\Delta\$I]) + \mathrm{Cos}[t1\,\Delta\$S]\,(-i\,\mathrm{Cos}[t2\,\Delta\$S] + \mathrm{Sin}[t2\,\Delta\$S]))$

```
In[130]:= ρ$4 = -ρ$4$A + I ρ$4$B;
          MakeNumerical[Tr[ρ$4 ** ((Ix + Sx) + I (Iy + Sy))]] // Simplify
```

Out[131]=
$$\text{Sin}\left[\frac{J\,t1}{2}\right]\text{Sin}\left[\frac{J\,t2}{2}\right]\big(-i\,\text{Cos}[t2\,\Delta\$S]\,\text{Sin}[t1\,\Delta\$I]+i\,\text{Cos}[t1\,\Delta\$S]\,\text{Sin}[t2\,\Delta\$I]+\text{Cos}[t2\,\Delta\$I]\,(\text{Cos}[t1\,\Delta\$S]-i\,\text{Sin}[t1\,\Delta\$S])+\text{Sin}[t2\,\Delta\$I]\,\text{Sin}[t1\,\Delta\$S]+\text{Cos}[t1\,\Delta\$I]\,(\text{Cos}[t2\,\Delta\$S]+i\,\text{Sin}[t2\,\Delta\$S])+\text{Sin}[t1\,\Delta\$I]\,\text{Sin}[t2\,\Delta\$S]\big)+2\,\text{Cos}\left[\frac{J\,t1}{2}\right]\text{Cos}\left[\frac{J\,t2}{2}\right]\text{Cos}\left[\frac{1}{2}(t1-t2)(\Delta\$I-\Delta\$S)\right]\left(\text{Cos}\left[\frac{1}{2}(t1-t2)(\Delta\$I+\Delta\$S)\right]-i\,\text{Sin}\left[\frac{1}{2}(t1-t2)(\Delta\$I+\Delta\$S)\right]\right)$$

Compute the signal and apply the 2D FT

This computation takes a minute .

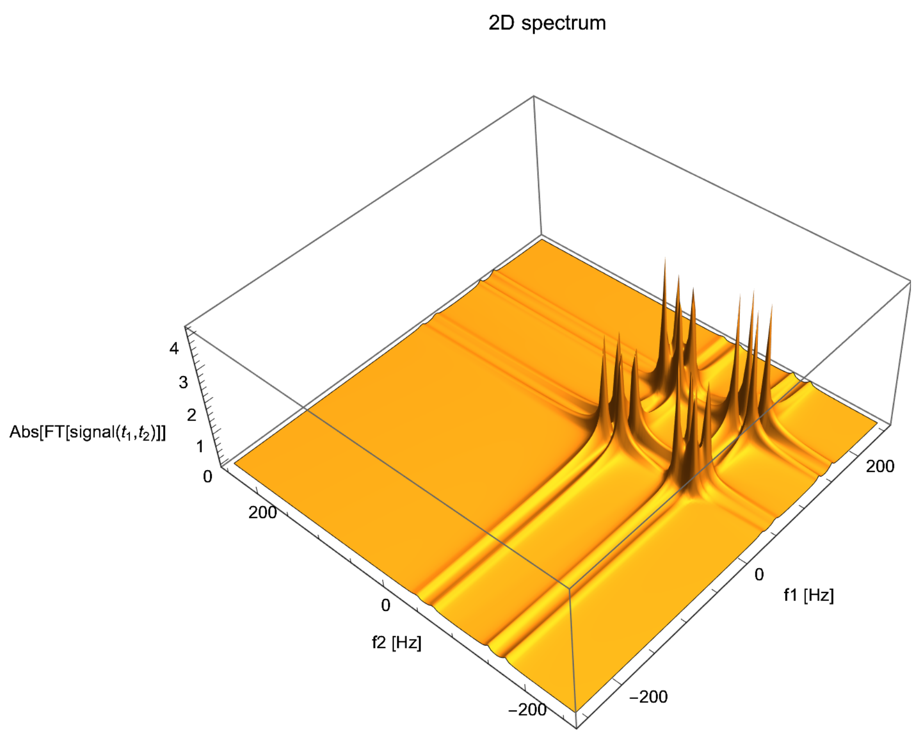
```
In[132]:= Signal2D[t1_, t2_] =
            ReplaceAll[
             Times[MakeNumerical[Tr[ρ$4 ** ((Ix + Sx) + I (Iy + Sy))]],
              Exp[-t1 / T2] Exp[-t2 / T2]], HamiltonianParameters$coupled];
```

Fourier transform the signal.   This takes a couple of seconds.

```
In[133]:= {p1, p2, p3} = DFFT2[Signal2D, t1$List, t2$List];
```

In[134]:=
```
p3
```

Out[134]=



Scratch

Spin operators

In this section we play with creating matrices for spin = 1/2 I and S-spin angular momentum operators in the 4 x 4 Zeeman basis. Name the matrices something different to avoid collisions.

In[135]:=
```
IzOp[1 / 2] = {{1 / 2, 0}, {0, -1 / 2}};
IzOp[1 / 2] // MatrixForm
```

Out[136]//MatrixForm=

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

```
In[137]:= IdOp[1 / 2] = IdentityMatrix[2];
          IdOp[1 / 2] // MatrixForm
```

Out[138]//MatrixForm=

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

The following procedure took a while to work out.

```
In[139]:= Iz$custom =
            Outer[Times, IzOp[1 / 2] , IdOp[1 / 2] ] // ArrayFlatten;
          Sz$custom =
            Outer[Times, IdOp[1 / 2] , IzOp[1 / 2]]  // ArrayFlatten;
```

```
In[141]:= Iz$custom  // MatrixForm
          Sz$custom  // MatrixForm
```

Out[141]//MatrixForm=

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

Out[142]//MatrixForm=

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

```
In[143]:= Iz$custom .Iz$custom  // MatrixForm
          Sz$custom .Sz$custom // MatrixForm
```

Out[143]//MatrixForm=

$$\begin{pmatrix} \frac{1}{4} & 0 & 0 & 0 \\ 0 & \frac{1}{4} & 0 & 0 \\ 0 & 0 & \frac{1}{4} & 0 \\ 0 & 0 & 0 & \frac{1}{4} \end{pmatrix}$$

Out[144]//MatrixForm=

$$\begin{pmatrix} \frac{1}{4} & 0 & 0 & 0 \\ 0 & \frac{1}{4} & 0 & 0 \\ 0 & 0 & \frac{1}{4} & 0 \\ 0 & 0 & 0 & \frac{1}{4} \end{pmatrix}$$

Bras and kets

This is how to implement column and row bras and kets in *Mathematica*

```
In[145]:= L$1 = {{a}, {b}};

In[146]:= L$2 = {c, d};

In[147]:= Outer[Q[#1, #2] &, L$2 , L$1] // FullForm

Out[147]//FullForm=
          List[List[List[Q[c, a]], List[Q[c, b]]], List[List[Q[d, a]], List[Q[d, b]]]]

In[148]:= Outer[Q[#1, #2] &, {a, b} , {c, d}] // FullForm

Out[148]//FullForm=
          List[List[Q[a, c], Q[a, d]], List[Q[b, c], Q[b, d]]]

In[149]:= L$1 . L$2
          L$2 . L$1

          ••• Dot : Tensors  {{a}, {b}} and {c, d } have incompatible shapes.

Out[149]= {{a}, {b}}.{c, d}

Out[150]= {a c + b d}
```