MATLAB® Program for Product Operator Formalism of Spin-1/2

Kosuke Ohgo

URL: https://github.com/ohgo1977/ProductOperator

Purpose

This program is designed to handle the product operator formalism of spin-1/2 using MATLAB. The calculation is based on the cyclic commutation rules thus it is faster than the actual matrix calculation. The program can handle multiple-spin systems with a flexibility of labeling of spins. This program will be helpful for educational purpose, for example, showing how to calculate product operators and explaining how components of pulse sequences (Hahn-echo, INEPT etc.) work. Also, this program can be used to calculate an evolution of a density operator under your own pulse sequence including phase cycling. The code is written in the object-oriented programming (OOP) style.

Requirement

MATLAB and MATLAB Symbolic Math Toolbox are required to run this program. It is tested under the MATLAB R2020b environment. Basic knowledge of MATLAB programming and the toolbox is required.

Limitation

This program can only handle weakly coupled spin-1/2 systems as same as the product operator formalism.

Design of the Programming and How to Use It

With a manner of the OOP, parameters for constructing product operators and functions for NMR interactions are described as **properties** and **methods** of a **class** named **PO**, respectively.

PO Class Properties

Any product operator can be described by two properties, axis label (x, y or z) and a coefficient.
For example, in the case of -4IxSzKz*cos(q), the axis labels are x, z and z for the 1st (I), 2nd (S) and 3rd (K) spins, respectively and the coefficient is -cos(q). Note that "4" is related to the number of spins in the system and thus it is not considered as an independent coefficient.

In the **PO** class, information on the axis label and coefficient are stored as **axis** and **coef** properties, respectively.

axis: This property stores axis labels of the product operators. It is a M x N matrix where M is a number of product operators in a system and N is a number of spin types. Each component in the matrix has a value of 0, 1, 2, 3 corresponding to 1-, x-, y- and z-operator, respectively. For example, Ix + Sx + Sy ($M = 3$) in the IS system ($N = 2$) has [1 0; 0 1; 0 2] as **axis**.

coef: This property stores coefficients including signs for the product operators. It is a M x 1 vector. Note that the 2^{N-1} coefficient at the beginning of a product operator is stored in other property, **Ncoef**.

There are additional properties in the **PO** class.
spin_label: This property defines labels of the spins in the current system such as, I,S,K, … or I1, I2, I3, … etc.. The default labels are I, S, K, L, M.

Ncoef: This property stores the 2^{N-1} coefficients for the product operators in the system. It is automatically calculated from the axis property.

txt: This property stores a text output of the current system. It is automatically generated.

M: This property stores a matrix form of the current system. It is automatically generated.

sqn: This property stores a spin quantum number. As a default, it stores sym(1/2) for spin-1/2.

disp: This property stores a binary value to control the display of the applied method and result on the command window. The default value is 1 for display ON.

coherence: This property is a 2^N x 2^N matrix displaying populations of spin states on the diagonal and coherences between states off the diagonal. a and b means |α> and |β> states, respectively, and m and p means coherences of |α> => |β> and |β> => |α>, respectively.

PO Class Methods

1. Creating an Initial State of a System (Class Constructor)

An initial state of a system, i.e., a density operator at the beginning, is created from the class constructor **PO()**.

As an example, if you like to construct ρ = Ix*cos(q) + 2IySz*sin(q),

```
>> syms q;
>> rho = PO(2, {'Ix' 'IySz'},{cos(q) sin(q)},{'I' 'S'})
rho =
  PO with properties:
        axis: [2×2 double]
        coef: [2×1 sym]
  spin_label: {'I'  'S'}
        disp: 1
  coherence : [4×4 sym]
       Ncoef: [2×1 sym]
         txt: 'Ix*cos(q) + 2IySz*sin(q)'
           M: [4×4 sym]
         sqn: [1×1 sym]
```

The general syntax of the constructor is

obj = PO(spin_no, sp_cell, symcoef_cell, spin_label_cell)

where **obj** is a PO-class object, **spin_no** is a number of spin types in the system, **sp_cell** is a cell array for product operators (without 2^{N-1} coefficients), **symcoef_cell** is a cell array for coefficients (coefficients must be sym class) and **spin_label_cell** is a cell array for spin labels that will be used as the class property **spin_label**.

There are some examples below how to use the constructor.

ρ = Ix + Sy

```
>> rho = PO(2, {'Ix' 'Sy'});
```

If only the first two parameters are set, the third, **symccoef_cell**, is automatically set as {sym(1) sym(1)} and the fourth, **spin_label_cell**, is set as {'I' 'S' 'K' 'L' 'M'}.

ρ = I1x + I2y

```
>> rho = PO(2, {'I1x' 'I2y'},{sym(1) sym(1)},{'I1' 'I2'});
```

Since I1 and I2 will be used instead of I and S, it is necessary to input {'I1' 'I2'} as **spin_label_cell** in addition to {sym(1) sym(1)} as **symcoef_cell**.

ρ = I1x in a 3-spin system (I1, I2 and I3)

```
>> rho = PO(3, {'I1x'},{sym(1) },{'I1' 'I2' 'I3'});
```

It is necessary to set the number of spins in the system by the constructor. Currently, it is not possible to increase the number of spins in the system later. Also, **spin_label_cell** should be assigned for this particular 3-spin system.

ρ = Ix + 4IxSyKz

```
>> rho = PO(3, {'Ix' 'IxSyKz'});
```

The 2^{N-1} coefficient, 4 in 4IxSyKz, should not be included in both **sp_cell** and **symcoef_cell**.

As a special case,

```
>> rho = PO(1,{'1'});
```

creates a 1/2E operator with rho.axis = [0], rho.coef = [1], rho.Ncoef = [1/2] and rho.txt = '1/2E' and rho.M = [1/2 0; 0 1/2]. Actually, any types of characters not defined in spin_label are used in sp_cell, they are considered as 1/2E operator.

Values of the PO properties can be obtained by a syntax **obj.PropertyName**. For example,

```
>> rho_matrix = rho.M;
```

In the current code, it is possible to rewrite many of properties directly from the command line, for example,

```
>> rho.axis = [1 2 3];
```

that is not an expected operation. In the future version, these properties will be only rewritable from the PO methods.

2. Applying NMR Interactions to a System

There are PO-class methods for NMR interactions i.e., RF pulse, chemical shift and *J*-coupling to apply a spin system.

RF Pulses

Single Pulse

A method to apply a single pulse is

obj = pulse(obj, sp, ph, q) or

obj = obj.pulse(sp, ph, q) .

where **obj** is a PO-class object, **sp** is a type of spin for the pulse, **ph** is a quadrature phase, and **q** is a flip angle in radian.
sp can be characters ('I', 'S', 'I1' or 'I2' etc. defined in **spin_label**) or the order number of spin (1 for 'I', 2 for 'S' etc.).
ph can be characters such as 'x', 'X' or '-y' or numbers 0, 1, 2 or 3 for x, y, -x or -y, respectively.
q can be double or symbolic class, such as pi/2 (double) or syms q (symbolic). If q is double class, it is converted to symbolic class in the code.

Examples

```
>> rho = PO(1, {'Iz'});
>> rho = pulse(rho, 'I', 'x', pi/2);
```

or equivalently,

```
>> rho = rho.pulse(1, 0, pi/2);
```

Simultaneous Pulses

A method to apply simultaneous pulses is

obj = simpulse(obj, sp_cell, ph_cell, q_cell) or

obj = obj.simpulse(sp_cell, ph_cell, q_cell) .

sp_cell, **ph_cell** and **q_cell** are cell arrays corresponding to **sp**, **ph** and **q** for **pulse()**.

```
>> rho = PO(2, {'Iz' 'Sz'});
>> rho = simpulse(rho, {'I' 'S'}, {'x' 'x'}, {pi/2 pi/2});
```

or equivalently,

```
>> rho = simpulse(rho, {1 2}, {0 0}, {pi/2 pi/2});
```

Pulses with Phase Shift

A method to apply a single pulse with a phase shift is

obj = pulse_phshift(obj, sp, ph, q) or

obj = obj.pulse_phshift(sp, ph, q) .

The difference from **pulse()** is that **ph** is an arbitrary phase in radian. Accordingly, a method for simultaneous pulses with phase shifts is

obj = simpulse_phshift(obj, sp_cell, ph_cell, q_cell) or

obj = obj.simpulse_phshift(sp_cell, ph_cell, q_cell) .

Chemical Shift

A method to apply a chemical shift evolution is

obj = cs(obj, sp, q) or

obj = obj.cs(sp, q) ,

where **sp** is a type of spin and **q** is a rotation angle in radian. Formats of **sp** and **q** are same as the ones used for **pulse()**.

It is possible to obtain chemical shift evolutions of multiple spins with

obj = simcs(obj, sp_cell, q_cell) or

obj = obj.simcs(sp_cell, q_cell) .

```
>> syms oI oS t1
>> rho = PO(2, {'Ix' 'Sx'});
>> rho = cs(rho, 'I', oI*t1);
>> rho = cs(rho, 'S', oS*t1);
```

or equivalently,

```
>> rho = simcs(rho, {1 2}, {oI*t1 oS*t1});
```

***J*-coupling**

A method to apply a *J*-coupling evolution is

obj = jc(obj, sp, q) or

obj = obj.jc(sp, q) ,

where **sp** is labels of a spin pair and **q** is a rotation angle in radian.

sp can be characters 'IS', 'I1I3' etc. or a 1 x 2 vector showing the index of spins such as [1 2] or [1 3].

```
>> rho = PO(2, {'Ix'});
>> syms J12 t
>> rho = jc(rho, 'IS', pi*J12*t);
```

Since the program is written in the OOP style, you can write a command like this.

```
>> rho = rho.pulse('I','y',pi/2).cs('I',q).jc('IS',pi*J12*t);
```

that is equivalent to

```
>> rho = rho.pulse('I','y',pi/2);
>> rho = rho.cs('I',q);
>> rho = rho. jc('IS',pi*J12*t);
```

Note that the order is left to right (**pulse()** => **cs()** => **jc()**) but not right to left (**jc()** => **cs()** => **pulse()**).

3. Utilities

There are several methods as utilities. Some of them are static methods, i.e., they are called by **PO.MethodName()**. These utilities can be used when a pulse sequence is emulated.

phout = PO.phmod(phx,ii)

Read a phase value, **phout**, from a phase table (vector), **phx**.

If **ii** is smaller or equal to length(phx), phout = phx(ii) otherwise phout = phx(mod(ii,length(phx)).

This method can be used in cases where there are phase tables with different lengths and phase-cycle them together.

dispPOtxt(obj) or

obj.dispPOtxt()

displays the **txt** property of **obj** to the Command Window.

dispPO(obj) or

obj.dispPO()

displays terms in **obj** in the following manner.

ID Product-Operator Coefficient

For example, in the case of ρ = Ix*cosθ + Iy* sinθ

```
>> dispPO(rho)
    1     Ix         cos(q)
    2     Iy         sin(q)
```

id_vec = findcoef (obj, coef_in_cell) or

id_vec = obj. findcoef (coef_in_cell)

finds particular terms that includes coefficients defined in **coef_in_cell** and returns ID numbers for these terms as **id_vec**.

obj = delPO(obj,id_in) or

obj = obj.delPO(id_in)

Delete particular terms from obj using the definition given by **id_in**.

If **id_in** is a vector including numbers, these numbers are ID for terms to be deleted.

The ID number of each term can be found by **obj.dispPO()** or be obtained from **obj.findcoef()**. The example is

```
>> rho = delPO(rho,[1 2])
```

If **id_in** is a cell array with product operators, terms including these operators are deleted. A wildcard character '*' can be used.

There are some examples.

```
>> rho = delPO(rho,{'Ix'}); % delete Ix term
>> rho = delPO(rho,{'IzSz'}) % delete 2IzSz term
>> rho = delPO(rho,{'Ix' 'IzSz'})% delete Ix and 2IzSz term
>> rho = delPO(rho,{'IxS*'})% delete 2IxSx, 2IxSy and 2IxSz if they exist
>> rho = delPO(rho,{'I*S*' 'I*S*K*'})% delete any terms including 2I*S* and
4I*S*K*
```

[a0_V,rho_V] = SigAmp(obj,sp,phR) or

[a0_V,rho_V] = obj.SigAmp(sp,phR)

Calculation of the initial signal amplitudes s(0) corresponding to (-1) quantum coherences in the equation

s(t) = 2*i*ρ[-](t) *exp(-i*ϕrec)

in Spin Dynamics, p. 288 (eq. 11.48).

For example, in the case of homonuclear 2-spin system, the equation is

s(t) = 2*i*(ρ[-β](t) + ρ[-α](t) + ρ[β-](t) + ρ[α-](t))*exp(-i*ϕrec)

shown in p. 379.

Related topics: Spin Dynamics (2nd Ed.), p.262, p. 287, p. 371, p.379, pp.608-610.

sp describes spin types to be observed (e.g., 'I', 'IS', [1], [1 2]). **phR** is a quadrature receiver phase (e.g. 'x', 'y', 0, 1).

a0_V is a vector corresponding to 2i*[ρ[-β…](0) ρ[-α…](0) ρ[β-…](0) ρ[α-…](0) …] *exp(-i*ϕrec).
rho_V is a vector describing which component of a0_V comes from which coherence in the density operator (e.g., 'ma' for '-α', 'bm' for 'β-').

<u>**Technical Details**</u>

Description of a product operator in the code

In the code, a product operator is described as a 1 x N vector where N is a number of spins in a system. The axis label of a product operator is described as 1 for x, 2 for y, and 3 for z, and 0 for 1/2*E. For example, Ix and IzSz in the IS system are described as [1 0] and [3 3], respectively. The intrinsic coefficient $2^{(N-1)}$ (e.g., 2 for 2IzSz) is automatically calculated from the vector size and stored separately. In the case of multiple terms, 1 x N vectors are combined to a M x N matrix where M is a number of terms. A coefficient of each term is stored separately.

Cyclic commutation rules

If the operators A, B, and C shows $[A, B] = \mathrm{i}C$, $[B, C] = \mathrm{i}A$ and $[C, A] = \mathrm{i}B$ (cyclic commutation) then
$\exp(-\mathrm{i}\theta A)\, B \exp(\mathrm{i}\theta A) = B \cos\theta + C \sin\theta$,
$\exp(-\mathrm{i}\theta B)\, C \exp(\mathrm{i}\theta B) = C \cos\theta + A \sin\theta$, and
$\exp(-\mathrm{i}\theta C)\, A \exp(\mathrm{i}\theta C) = A \cos\theta + B \sin\theta$

It is known that Ix, Iy, Iz are in the cyclic commutation ([Iz, Ix] = iIy) and the formula above can be used to describe an evolution of a density operator under a Hamiltonian. For example, a density operator ρ(0) = Ix evolves under a chemical shift Hamiltonian H = ωIz during a time period of t as
ρ(t) = exp(-iHt) ρ(0) exp(iHt) = exp(-iωtIz) Ix exp(iωtIz) = Ix cos(ωt) + Iy sin(ωt)

Use of the master table to accelerate calculation

				B	
			x	y	z
			1	2	3
	x	1	**0**	**3**	**-2**
A	y	2	**-3**	**0**	**1**
	z	3	**2**	**1**	**0**
				C	

The cyclic commutation rules can be summarized as a table (master table) using the equation exp(-iθA) B exp(iθA) = B cosθ + C sinθ above and Ix, Iy and Iz. For example, in the case of exp(-iθIz) Ix exp(iθIz) = Ix cosθ + Iy sinθ, the axis numbers of the A and B positions are 3 (z) and 1 (x), respectively. Then the axis number for C is the (3, 1) component in the table, i.e., 2 meaning Iy.

If the value for C is 0 for given A and B, then A and B are not in the cyclic commutation (e.g., A = B = Ix). If the value for C is negative, then -C sinθ is used instead of +C sinθ.

This is the basic idea of the calculation in the code, and it is not necessary to handle a matrix calculation which usually has a high calculation cost. above.

When can the master table be used for the calculation?

If the two rules below are satisfied, an evolution of ρ under H can be calculated with using the master table. If the rules are not satisfied, ρ does not evolve under H.

Rule 1. There should be at least one spin type matching between H and ρ.

AND

Rule 2. Only one spin type in the matching spin types has different axis labels between H and ρ.

These rules are true for spin-1/2.

The rule 1 is obvious but how about the rule 2? Here is the analysis.

Consider H = 16ABIaSbKc and ρ = 16CDIbSbKc where A, B, C, D are product operators which are different types each other in addition to I, S and K. Suppose [Ia, Ib] = iIc (cyclic commutation). There are three spin types matching between H and ρ (satisfying Rule 1) and only one of them (I spin) has different labels between H and ρ (satisfying Rule 2).

Then [H, ρ] = 16ABIaSbKc 16CDIbSbKc – 16CDIbSbKc 16ABIaSbKc = 16^2ABCDIaIbSb^2Kc^2 - 16^2ABCDIbIaSb^2Kc^2
= 16^2ABCD[IaIb – IbIa]Sb^2Kc^2 = 16ABCD[IaIb – IbIa] = i16ABCDIc. Note that Sb^2 = Kc^2 = 1/4E for spin-1/2.

[16ABCDIc, H] = 16ABCDIc 16ABIaSbKc - 16ABIaSbKc 16ABCDIc = $16^2A^2B^2$CDIcIaSbKc – $16^2A^2B^2$CDIaIcSbKc
= 16CDIcIaSbKc – 16CDIaIcSbKc = 16CD[Ic, Ia]SbKc = i16CDIbSbKc = iρ. Again A^2 = B^2 = 1/4E.

[ρ, 16ABCDIc] = 16CDIbSbKc 16ABCDIc - 16ABCDIc 16CDIbSbKc = $16^2ABC^2D^2$IbIcSbKc – $16^2ABC^2D^2$IcIbSbKc
= 16ABIbIcSbKc - 16ABIcIbSbKc = 16AB[IcIb – IbIc]SbKc = i16ABIaSbKc = iH. Again C^2 = D^2 = 1/4E.

This analysis can only work when the rule 2 is satisfied.

What if the rule 2 is not satisfied? This can be calculated using the matrix representation. For example, [2IaSb, 2IbSc] for spin-1/2 is 0 from the matrix calculation (Read Levitt p.403, Eq. 15.24 and p. 407, Note 3).

Examples for these rules

ρ = Iy and H = Iz

Both ρ and H include I-spin (Rule 1: yes) and they have different axis labels (Iy vs. Iz) (Rule 2: yes). ➔ Master Table: Yes

ρ = Sy and H = Iz

ρ and H don't have a same type of spin (Rule 1: No) ➔ Master Table: No

ρ = Iy and H = Iy

Both ρ and H include I-spin (Rule 1: yes) but they have same axis labels (Iy) (Rule 2: no). ➔ Master Table: no

ρ = 2IzSy and H = Iz

Both ρ and H include I-spin (Rule 1: yes) but I-spins have same axis labels (Iz) (Rule 2: no). ➔ Master Table: no

ρ = 2IzSy and H = 2IzSz

Both ρ and H include I- and S-type product operators (Rule 1: yes) and only S-spins have different axis labels (Sy vs. Sz) (Rule 2: yes). ➔ Master Table: yes ρ = 2IxSx and *H* = Iz

Both ρ and *H* include I-spin (Rule 1: yes) and they have different axis labels (Ix vs. Iz) (Rule 2: yes). ➔ Master Table: Yes

ρ = 2IxSx and *H* = 2IzSz

Both ρ and *H* include I- and S-type product operators (Rule 1: yes) but both spin types have different axis labels (Ix vs. Iz and Sx vs. Sz) (Rule 2: no). ➔ Master Table: No

Special cases for *J*-coupling evolutions

According to Spin Dynamics, p. 483, there are four cases where a product operator does not evolve under *J*-coupling Hamiltonian $I_j z I_k z$.

1. If both spin I_j and I_k are missing in the product operator.
2. If only one spin I_j or I_k is present, and that spin carries a z label.
3. If both spins I_j and I_k are present, but both spins carry a z label.
4. If both spins I_j and I_k are present, but neither spin carries a z label.

These four cases are excluded by the two rules above.

Case 1. Rule 1 is not satisfied.

Case 2. Rule 1 is satisfied but Rule 2 is not satisfied.

Case 3. Rule 1 is satisfied but Rule 2 is not satisfied.

Case 4. Rule 1 is satisfied but Rule 2 is not satisfied.

Implementation of the two rules in the programing code

The two rules above can be evaluated in the programming code as shown below.

```
type_mask_vec = (rho_axis.*H_axis)~=0;
% Check how many spin types get matched, matched: 1, unmatched: 0
axis_diff_vec = rho_axis ~= H_axis;
% Check the difference of the direction of each spin type, unmatched: 1, matched: 0
axis_mask_vec = type_mask_vec.*axis_diff_vec;
% Comparing the spin-type matching and spin-label unmatching.
axis_mask = sum(axis_mask_vec);
```

axis_mask becomes 1 ONLY when the two rules are satisfied.

If axis_mask is 1, then

H_axis = [h_1 h_2 h_3 …] corresponding to *A*

rho_axis = [r_1 r_2 r_3 …] corresponding to *B*

are used to calculate a product operator

axis_tmp = [a_1 a_2 a_3 …] corresponding to *C*.

if both r_n and h_n are not 0, a_n takes an absolute value of the (h_nth row, r_nth column) component of the master table. If the value from the master table is negative, the sign of the coefficient is inverted.

if r_n is not 0 but h_n is 0, a_n is same as r_n. This is for cases such as ρ = 2IzSz and *H* = Ix ➔ *C*: 2IySz
if r_n is 0 but h_n is not 0, a_n is same as h_n. This is for cases such as ρ = 2Ix and *H* = 2IzSz ➔ *C*: 2IySz

Note that the sign of *H* (i.e., Ix vs. -Ix) is not reflected in the calculation above. Instead, the sign of *H* is considered in the coefficient calculation of *C*.

References

Levitt, M. H. Spin Dynamics, 2nd Edition, Wiley, 2008.