Supporting information:
Various facets of intermolecular transfer of phase coherence by nuclear dipolar fields

Philippe Pelupessy[1]

[1]Laboratoire des Biomolécules, LBM, Département de Chimie, École Normale Supérieure, PSL University, Sorbonne Université, CNRS, 75005 Paris, France

Correspondence: Philippe Pelupessy (philippe.pelupessy@ens.psl.eu)

Contents

Supporting information. Two additional figures and simulation program.

Figures

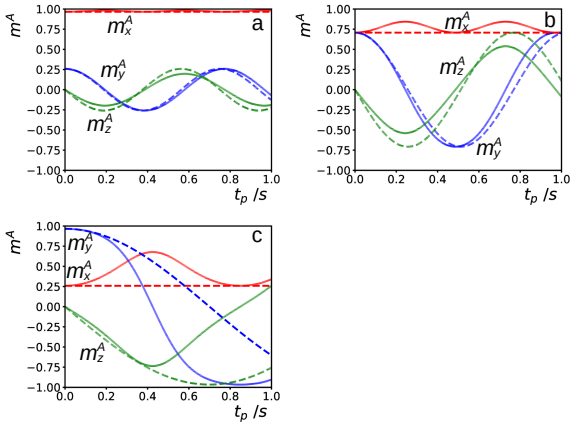

Figure SI1. Same as fig. 1a-c in the main text, except that the rf pulse-train was GARP instead of DIPSI-2. All other conditions for the simulations were identical.

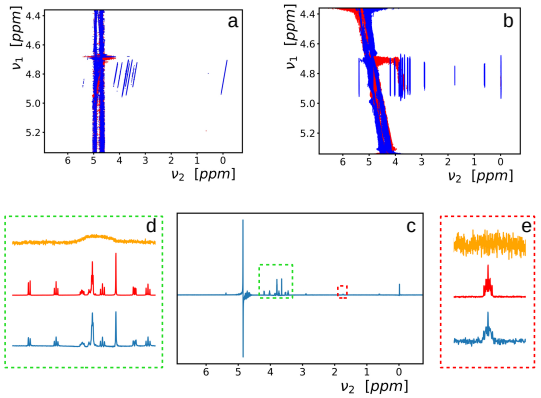

Figure SI2. Results of an experiment similar to the one of fig. 3 of the main text. The sole differences were a repetition time of about 3 s instead of 11 s, and 1 scan per increment instead of 4. This results in an experimental time of about 25 min. instead of 6 hours.

Program

```
'''
modified: 21-07-2023
Parameter file for simul_df_tocsy_01.py
    For fig. 1d use:
        iniDSS = "zed"
        vDF = 1.84
        order = [-1, 1]
        we look at the 'planeabs' results
    For fig. 5b use:
        iniDSS = "MH2O"
        vDF = 1.84 or -0.5*1.84
        order = [2]
        we look at the 'zabs' results
'''
B0 = -800.13          # main field (minus sign for positif gyrom.)
vDSS = -4.85          # chemical shift difference wr H2O (ppm)
vDF = 1.84            # amplitude dipolar field

mixing = "Dipsi2"     # "Dipsi2","Garp","cw" or "Waltz16"
tau90 = 30.e-6        # duration 90 pulse, defines rf power mixing
maxcyc = 120          # maximum number of mixing cycles

ngrid = 36            # number of equidistant (in angles) gridpoints for
                      # the dephased solvent magnetization (MH2O)
iniDSS = "MH2O"       # "iks", "why","zed" or "MH2O"
                      # MH2O is initially dephased in the plane, MDSS (the solute
                      # magnetization) can be along one of the axis or initially
                      # aligned with MH2O
order = [2]           # indicates the ratio beween the pfg before acquisition
                      # and the dephasing pfg
plot = True
plotopt = {'plane':False, 'planeabs':True,'z':False,'zabs':True}
                      # 'zed' plots the magnetization in the plane after:
                      #  pfg-mixing-spoiling-90-pfg*order
                      # 'plane' plots the magnetization in the plane after:
                      #  pfg-mixing-pfg*order
plotcol = ['blue','red','green','orange','cyan']
savedata = False
```

```
'''
modified: 21-07-2023
Simulation program transfer of phase coherence from abundant spins (H2O)
to solute (DSS) by dipolar field during TOCSY.
Only with mixing sequences along one axes with constant rf amplitude.
The solvent magnetization evolves as described in the main article.

Program uses numba for jit compilation. Uses para02.py file for parameters
so that numba can use cache.

The program has been run with versions:

Python      3.11.3
Numpy       1.25.1
Numba       0.57.1
Matplotlib 3.7.2

This code is provided for the purpose of checking and/or reproducing the
simulations of the main article and comes without any warranty. If you use
(part of) this code for your own work, please cite the original publication.

2023  Philippe Pelupessy
'''

# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from numba import njit,prange
import para02 as pa    #import parameter file
import importlib
importlib.reload(pa) #reload parameter file, needed if rerun from ipython

# Evolution of the solute magnetization
@njit(parallel=True,cache = True,fastmath=True)
def evolveDSS(MDSS,ngrid,maxcyc,MH2O,SupCyc,Angle,Phase,
              IA,sA,cA,rfA,wDSS,lA,lS,unit):
    '''Rotation around an axis during spinlock sequence along the x-axis. The
       axis of rotation for DSS is determined by the rf field and by the MH2O.
       RF power is constant but sign may alternate. On-resonance for abundant
       solvent spins, while magnetization of the off-resonant sparse spins is
       calculated. Optimized for numba just in time compilation. Gains in speed
       using fastmath=True depend strongly on computer.'''
    pi1 = np.pi/(180*unit)
```

```
    for i in prange(ngrid): #loop over all sample positions
        mxm,mym,mzm = MDSS[0,i,0],MDSS[1,i,0],MDSS[2,i,0] #initial MDSS
        c = 0
        for k in range(maxcyc):
            X,Y,Z = MH2O[0,i,k],MH2O[1,i,k],MH2O[2,i,k] #precalculated MH2O
            #at beginning of each super-cycle. Very little difference if one
            #takes average of k and k+1.
            for l in range(lS): #loop 'supercycle'
                for m in range(lA):#loop 'subcycle'
                    sig = Phase[m]*SupCyc[l] #sign rf
                    Xs,Ys,Zs = rfA*sig-0.5*(IA*X),Y*sig,Z*sig
                    for n in range(Angle[m]): #loop over points in one pulse
                        x = Xs #x-field felt by DSS
                        y = -0.5*(cA[n]*Y-sA[n]*Zs) #y field
                        z = wDSS+(cA[n]*Z+sA[n]*Ys) #z field
                        theta = np.sqrt(x**2+y**2+z**2) #the rest in loop serves
                        #to calculate the rotation of DSS
                        a,t = np.cos(theta*0.5),-np.sin(theta*0.5)/theta
                        at2,t2 = 2*a*t,t*t
                        a2 = a*a
                        b2,c2,d2 = t2*x*x,t2*y*y,t2*z*z
                        ab,ac,ad = at2*x,at2*y,at2*z
                        bc,bd = 2*t2*x*y,2*t2*x*z
                        cd = 2*t2*y*z
                        mxp = (a2+b2-c2-d2)*mxm+(bc+ad)*mym+(bd-ac)*mzm
                        myp = (bc-ad)*mxm+(a2-b2+c2-d2)*mym+(ab+cd)*mzm
                        mzp = (bd+ac)*mxm+(cd-ab)*mym+(a2-b2-c2+d2)*mzm
                        mxm,mym,mzm = mxp,myp,mzp
                    Ap = Angle[m]*pi1*sig
                    Y,Z = np.cos(Ap)*Y-np.sin(Ap)*Z,np.cos(Ap)*Z+np.sin(Ap)*Y #
                    #update MH2O ofter each rf pulse (purely driven by rf)
                    c += Angle[m]
            MDSS[0,i,k+1],MDSS[1,i,k+1],MDSS[2,i,k+1] = mxp,myp,mzp #store MDSS
            #after each cycle
    return MDSS



def main():
    B0 = pa.B0            #main field (minus sign for positive gyrom.)
    vDSS = pa.vDSS        #chemical shift difference with H2O (ppm)
    mixing = pa.mixing    #spin-lock seq.  "Dipsi2","Garp","cw" or "Waltz16"
    tau90 = pa.tau90      #duration 90 pulse spinlock
    maxcyc = pa.maxcyc    #maximum number of cycles spin-lock
```

```
88:     ngrid = pa.ngrid       #H20 magnetization is phasemodulated number of phases
89:     wDF = 2*np.pi*pa.vDF   #dipolar field amplitude
90:     iniDSS = pa.iniDSS     #inital DSS magn. "iks", "why","zed" or "MH2O"
91:
92:     order = pa.order       #lists of initial orders, SQ, DQ, TQ...(+ or -)
93:     plot = pa.plot         #True or False
94:     plotopt = pa.plotopt   #Dictionary of plotoptions (True or False):
95:                            #plane, planeabs,z, zabs
96:     plotcol = pa.plotcol   #colors plot (each order has its color)
97:     savedata = pa.savedata#if True save data to text file
98:
99:     #sequences consist of a sequence of pulses of rotation angle integer/unit
100:    #always along the x-axis, with positive or negative sign (0, 180),
101:    #repated in a supercycle (e.g. for Dipsi2 a full cycle is 4*9= 36 pulses)
102:    if mixing=="Dipsi2":
103:        unit = 1
104:        Phase =  np.array([0,180,0,180,0,180,0,180,0])*(-1./90)+1
105:        Angle  = np.array([320,410,290,285,30,245,375,265,370])
106:        SupCyc = np.array([0,180,180,0])*(-1./90)+1
107:    if mixing=="Waltz16":
108:        unit = 1
109:        Phase =  np.array([0,180,0])*(-1./90)+1
110:        Angle  = np.array([90,180,270])
111:        SupCyc = np.array([0,0,180,180,180,0,0,180,
112:                           180,180,0,0,0,180,180,0])*(-1./90)+1
113:    if mixing=="Dipsi20": #to test unit = 10,  gives same result a Dipsi2
114:        unit = 10
115:        Phase =  np.array([0,180,0,180,0,180,0,180,0])*(-1./90)+1
116:        Angle  = np.array([3200,4100,2900,2850,300,2450,3750,2650,3700])
117:        SupCyc = np.array([0,180,180,0])*(-1./90)+1
118:    if mixing=="Garp":
119:        unit = 10
120:        Phase =  np.array([0,180,0,180,0,180,0,180,0,180,0,180,0,180,
121:                           0,180,0,180,0,180,0,180,0,180,0])*(-1./90)+1
122:        Angle  = np.array([305,552,2578,2683,693,622,850,918,1345,2561,
123:                           664,459,255,727,1195,1382,2584,649,709,772,
124:                           982,1336,2559,656,534])
125:        SupCyc = np.array([0,180,180,0])*(-1./90)+1
126:    if mixing=="cw":
127:        unit = 1
128:        Phase =  np.array([0])*(-1./90)+1
129:        Angle  = np.array([360*1]) #multiples of 360
130:        SupCyc = np.array([0])*(-1./90)+1
131:
```

```
    tau1 = tau90/(90*unit) #duration of 1 step in calculation
    w1 = 2*np.pi/(4.*tau90) #rf amplitude

    H2Oini = np.array([[np.cos(i*2*np.pi/ngrid),np.sin(i*2*np.pi/ngrid),0]
                       for i in range(ngrid)]).T #initial condition solvent
    dt_sc = tau1*np.sum(Angle)*len(SupCyc) #lenght one supercycle
    time = dt_sc*range(maxcyc+1)#time from 0 to end mixing, in super cycle inc.
    print('Maximum mixing time:',time[-1])

    #next 3 lines. Evolution solvent after each supercycle. To a good
    #approximation a rotation arount the x-axis, with angular freq.=
    #-(3/4)*wDF*Mx(0)
    c = np.cos(-(3/4)*wDF*np.outer(H2Oini[0],time))
    s = np.sin(-(3/4)*wDF*np.outer(H2Oini[0],time))
    MH2O = np.array([np.outer(H2Oini[0],np.ones(maxcyc+1)),
                     np.einsum('i,ij->ij',H2Oini[1],c)-
                     np.einsum('i,ij->ij',H2Oini[2],s),
                     np.einsum('i,ij->ij',H2Oini[2],c)+
                     np.einsum('i,ij->ij',H2Oini[1],s)])

    MDSS = np.zeros_like(MH2O) #next lines initial MDSS
    if iniDSS == "MH2O":
        MDSS[:,:,0] = MH2O[:,:,0].copy()
    if iniDSS == "iks":
        MDSS[0,:,0]+=1.0
    if iniDSS == "why":
        MDSS[1,:,0]+=1.0
    if iniDSS == "zed":
        MDSS[2,:,0]+=1.0

    rfA  = w1*tau1 #rotation angle rf in 1 time-increment
    wDSS  = tau1*vDSS*B0*2*np.pi #rotation angle offset
    IA  = tau1*wDF #rotation angle wDF/MH2Ox around x-axis
    #next two lines, mixing factors for y and z contributions df (makes very
    #little difference to add 0.5 or not). This boils down to a precalculation
    #of sin and cos factors for MH2O evolution during pulses
    cA  = tau1*wDF*np.cos((np.arange(max(Angle))+0.5)*np.pi/(180*unit))
    sA  = tau1*wDF*np.sin((np.arange(max(Angle))+0.5)*np.pi/(180*unit))

    lA = len(Angle)
    lS = len(SupCyc)

    print('Compilation and parallel calculation starts')
    MDSS = evolveDSS(MDSS,ngrid,maxcyc,MH2O,SupCyc,Angle,Phase,IA,sA,
```

```
                    cA,rfA,wDSS,lA,lS,unit)
    #plot for different pfgs before acquistion. 'z' corresponds to, spoil-90
    #before the same pfg.
    c = 0
    for o in order:
        co = np.cos(o*2*np.pi*np.arange(ngrid)/(ngrid))/ngrid
        si = np.sin(o*2*np.pi*np.arange(ngrid)/(ngrid))/ngrid
        MDSSbuX = co.dot(MDSS[0])-si.dot(MDSS[1])
        MDSSbuY = co.dot(MDSS[1])+si.dot(MDSS[0])
        MDSSbuZx  = co.dot(MDSS[2])
        MDSSbuZy =  si.dot(MDSS[2])
        if plot:
            if plotopt['plane']:
                plt.figure('plane')
                plt.plot(time,MDSSbuX,color = plotcol[c])
                plt.plot(time,MDSSbuY,color = plotcol[c],linestyle='--')
            if plotopt['planeabs']:
                plt.figure('planeabs')
                plt.plot(time,np.sqrt(MDSSbuY**2+MDSSbuX**2),color = plotcol[c])
            if plotopt['z']:
                plt.figure('z')
                plt.plot(time,MDSSbuZx,color = plotcol[c])
                plt.plot(time,MDSSbuZy,color = plotcol[c],linestyle='--')
            if plotopt['zabs']:
                plt.figure('zabs')
                plt.plot(time,np.sqrt(MDSSbuZx**2+MDSSbuZy**2),color = plotcol[c])
        c +=1
        #can be saved to make own plots, especially to compare
        if savedata:
            np.savetxt('mag'+str(o)+'.txt',[time,MDSSbuX,MDSSbuY,MDSSbuZx,
                                            MDSSbuZy])
    plt.show()
if __name__ == '__main__':
    main()
```