Supplement of Magn. Reson., 4, 271–283, 2023
https://doi.org/10.5194/mr-4-271-2023-supplement
© Author(s) 2023. CC BY 4.0 License.

Open Access
MAGNETIC
RESONANCE

Supplement of

Various facets of intermolecular transfer of phase coherence by nuclear dipolar fields

Philippe Pelupessy

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

The copyright of individual parts of the supplement might differ from the article licence.

Contents

Four additional figures and a simulation program.

- Figure S1) Simulations GARP 2
- Figure S2) Short experiment in inhomogeneous field 3
- Figure S3) Line shapes in 2D experiments in inhomogeneous fields 4
- Figure S4) 1D spectrum with assignments 5
- Code simulation program (parameter file) 6
- Code simulation program (main) 7-11

Figures

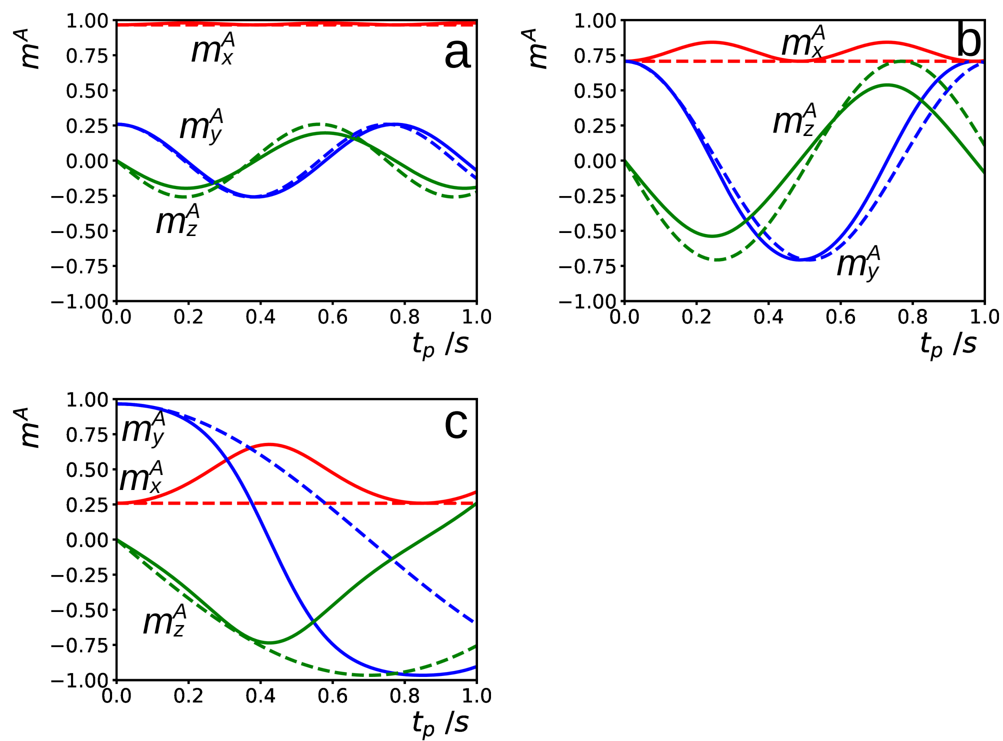

Figure S1. Same as fig. 1a-c in the main text, except that the rf pulse-train was GARP instead of DIPSI-2. All other conditions for the simulations were identical.

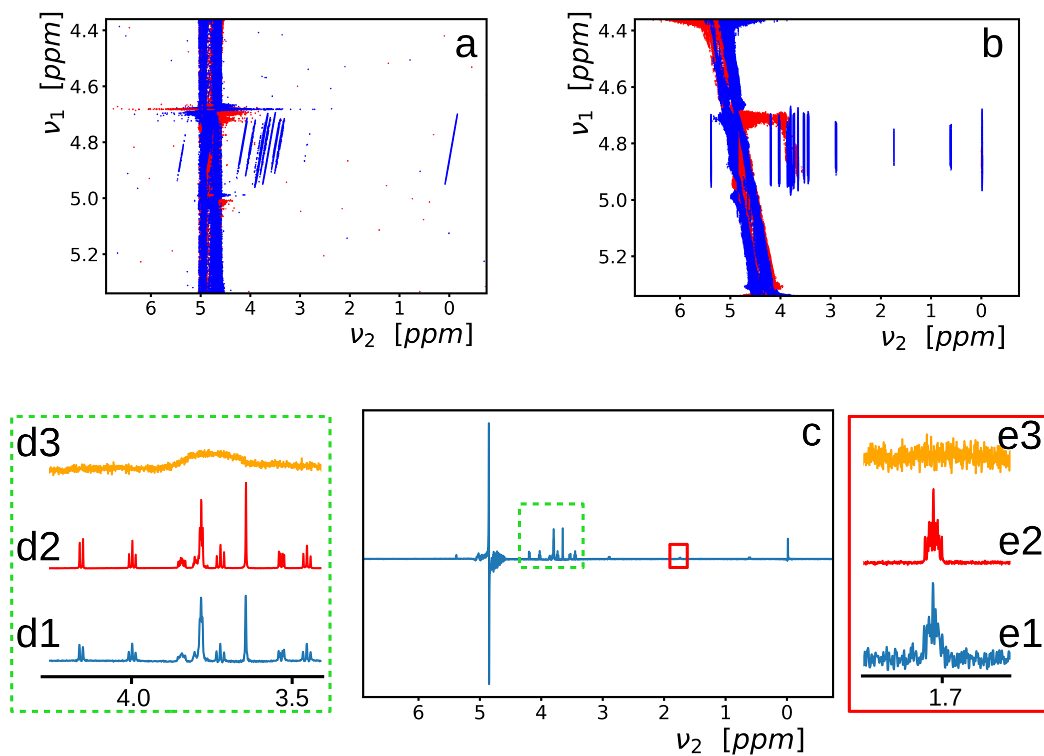

Figure S2. Results of an experiment similar to the one of fig. 3 of the main text. The sole differences were a repetition time of about 3 s instead of 11 s, a DIPSI-2 irradiation time of 100 ms instead of 200 ms, and 1 scan per increment instead of 4. This results in an experimental time of about 12 minutes instead of 3 hours.

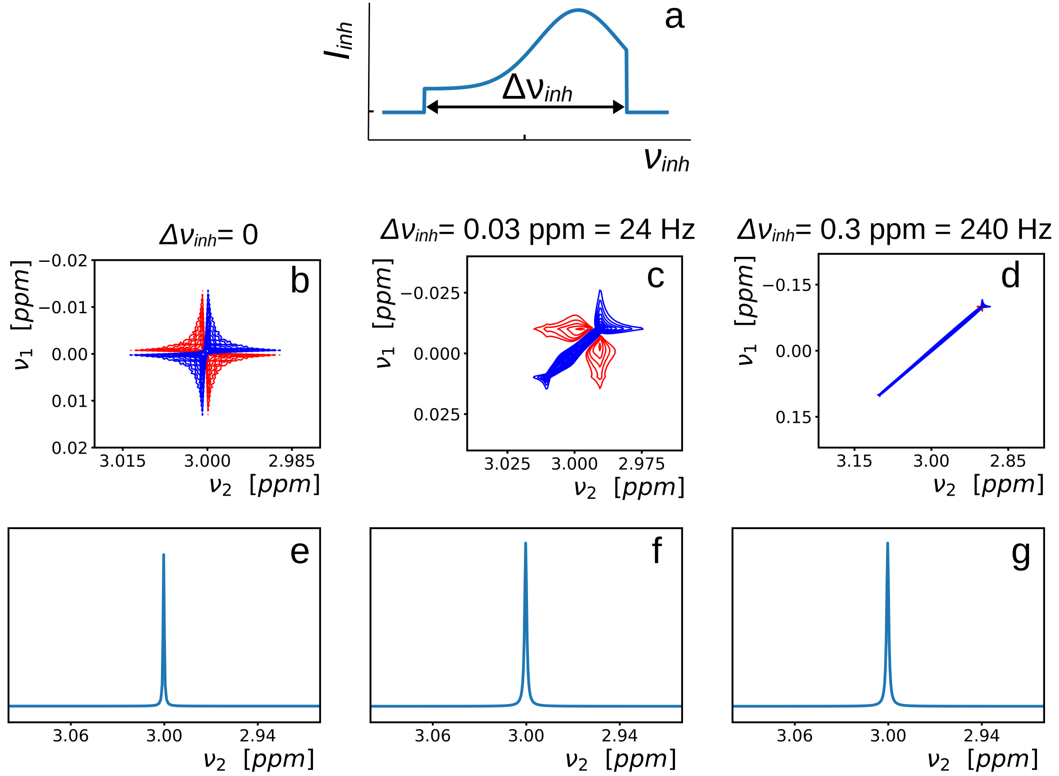

Figure S3. Calculated line shapes resulting from the experiment of fig. 2 of the main text. **(a)** The profile that characterizes the B_0 inhomogeneities, which is considered in the calculations. For a spin species S, the experiment results in:

$$sig(S) = \int cI_{inh} e^{(-i\omega_{inh} - R_2^A)t_1} e^{(i\omega_{inh} + i\omega_0^S - R_2^S)t_2} d\omega_{inh}, \tag{S1}$$

where c is a constant that depends on several factors such as the spin-density and transfer and detection efficiencies. R_2^A and R_2^S are the transverse relaxation times of the A and S spins. The signal $sig(S)$ has been calculated for proton spin S with $\omega_0^S = 3$ ppm, R_2^A = 2 s^{-1} and R_2^S= 2 s^{-1} at a field of B_0 = 18.8 T (800 MHz proton frequency). 12×1024 complex points for a bandwidth of 12 ppm were used in the direct t_2 dimension and 1024 indirect t_1 increments for a bandwidth of 1 ppm (more points have been used than in the experiments of fig. 3 in the main manuscript and fig. S2 to avoid truncation effects). In the center row the real part of the 2D Fourier transform of eq. S1 for three different scenarios is plotted: **(b)** a homogeneous B_0 field ($\Delta\nu_{inh} = 0$), **(c)** a moderately inhomogeneous field ($\Delta\nu_{inh} = 0.03$ ppm), and **(d)** a substantially inhomogeneous field ($\Delta\nu_{inh} = 0.3$ ppm). The spectra zoom in on the region of interest and the lowest contour levels are at 2% of the maximum intensity of each spectrum. Since only 1 coherence pathway is recorded, the spectrum cannot be phased to pure absorption mode, as evidenced by the severe phase twist in (b). However, due to the symmetry of these distortions, the phase twists do not appear in the sum of the rows as shown in **(e)**. The phase-twists are attenuated with increasing B_0 inhomogeneities, and are not visible in the sum, after shearing of the 2D spectra, of the rows as shown below the spectra (c) and (d) in **(f)** and **(g)**. The sum spectra in (e-g) have the same normalization factor. Thus increased inhomogeneities do not reduce the signal intensity, but, since the signal is spread out over more rows, lead to a decrease of the signal-to-noise ratio. The line-width in (e) depend on R_2^S, the line-width in (f) and (g) on the sum $R_2^A + R_2^S$.

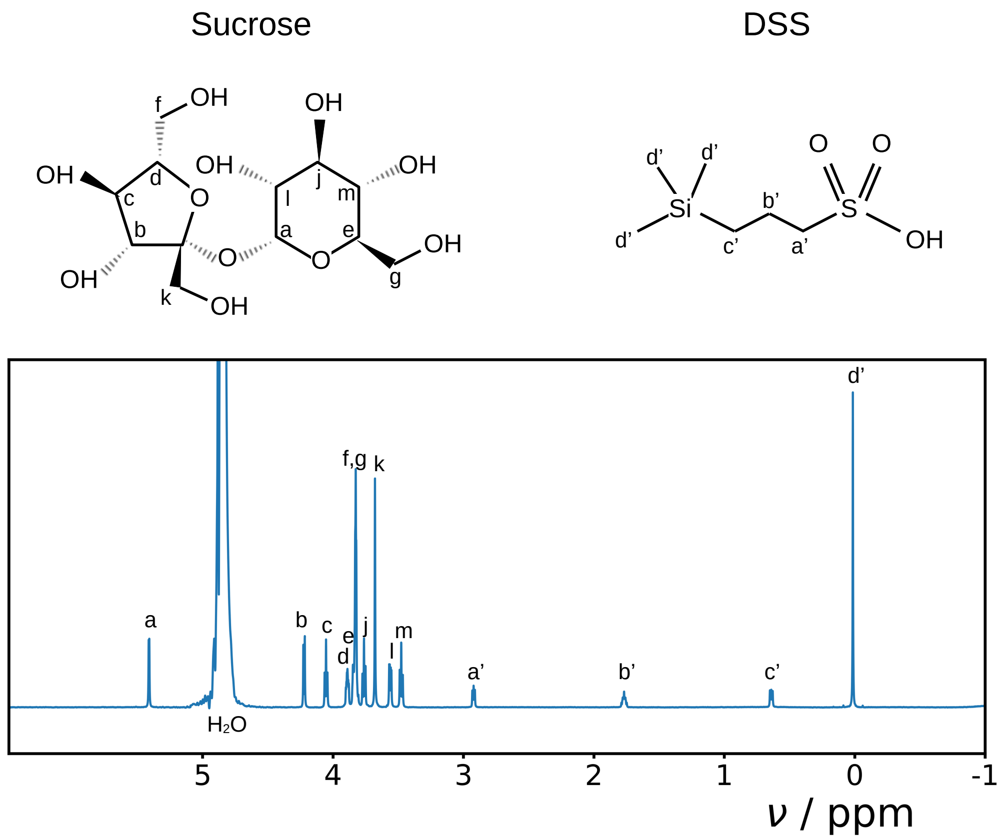

Figure S4. Spectrum obtained with a pulse-acquire experiment preceded by saturation of the solvent signal (H_2O). The sample, 0.5 mM DSS and 2.0 mM sucrose in a 90%/10% mixture of H_2O/D_2O, has been used for all experiments in this work. The assignments come from the Spectral Database for Organic Compounds, SDBSWeb, https://sdbs.db.aist.go.jp (National Institute of Advanced Industrial Science and Technology, date of access: 05-09-2023).

Program

```
1: '''
2: modified: 21-07-2023
3: Parameter file for simul_df_tocsy_01.py
4:     For fig. 1d use:
5:         iniDSS = "zed"
6:         vDF = 1.84
7:         order = [-1, 1]
8:         we look at the 'planeabs' results
9:     For fig. 5b use:
10:        iniDSS = "MH2O"
11:        vDF = 1.84 or -0.5*1.84
12:        order = [2]
13:        we look at the 'zabs' results
14: '''
15: B0 = -800.13          # main field (minus sign for positif gyrom.)
16: vDSS = -4.85          # chemical shift difference wr H2O (ppm)
17: vDF = 1.84            # amplitude dipolar field
18:
19: mixing = "Dipsi2"     # "Dipsi2","Garp","cw" or "Waltz16"
20: tau90 = 30.e-6        # duration 90 pulse, defines rf power mixing
21: maxcyc = 120          # maximum number of mixing cycles
22:
23: ngrid = 36            # number of equidistant (in angles) gridpoints for
24:                       # the dephased solvent magnetization (MH2O)
25: iniDSS = "MH2O"       # "iks", "why","zed" or "MH2O"
26:                       # MH2O is initially dephased in the plane, MDSS (the solute
27:                       # magnetization) can be along one of the axis or initially
28:                       # aligned with MH2O
29: order = [2]           # indicates the ratio beween the pfg before acquisition
30:                       # and the dephasing pfg
31: plot = True
32: plotopt = {'plane':False, 'planeabs':True,'z':False,'zabs':True}
33:                       # 'zed' plots the magnetization in the plane after:
34:                       #  pfg-mixing-spoiling-90-pfg*order
35:                       # 'plane' plots the magnetization in the plane after:
36:                       #  pfg-mixing-pfg*order
37: plotcol = ['blue','red','green','orange','cyan']
38: savedata = False
```

```
1: '''
2: modified: 21-07-2023
3: Simulation program transfer of phase coherence from abundant spins (H2O)
4: to solute (DSS) by dipolar field during TOCSY.
5: Only with mixing sequences along one axes with constant rf amplitude.
6: The solvent magnetization evolves as described in the main article.
7:
8: Program uses numba for jit compilation. Uses para02.py file for parameters
9: so that numba can use cache.
10:
11: The program has been run with versions:
12:
13: Python     3.11.3
14: Numpy      1.25.1
15: Numba      0.57.1
16: Matplotlib 3.7.2
17:
18: This code is provided for the purpose of checking and/or reproducing the
19: simulations of the main article and comes without any warranty. If you use
20: (part of) this code for your own work, please cite the original publication.
21:
22: 2023  Philippe Pelupessy
23: '''
24:
25: # Import necessary libraries
26: import numpy as np
27: import matplotlib.pyplot as plt
28: from numba import njit,prange
29: import para02 as pa   #import parameter file
30: import importlib
31: importlib.reload(pa) #reload parameter file, needed if rerun from ipython
32:
33: # Evolution of the solute magnetization
34: @njit(parallel=True,cache = True,fastmath=True)
35: def evolveDSS(MDSS,ngrid,maxcyc,MH2O,SupCyc,Angle,Phase,
36:               IA,sA,cA,rfA,wDSS,lA,lS,unit):
37:     '''Rotation around an axis during spinlock sequence along the x-axis. The
38:        axis of rotation for DSS is determined by the rf field and by the MH2O.
39:        RF power is constant but sign may alternate. On-resonance for abundant
40:        solvent spins, while magnetization of the off-resonant sparse spins is
41:        calculated. Optimized for numba just in time compilation. Gains in speed
42:        using fastmath=True depend strongly on computer.'''
43:     pi1 = np.pi/(180*unit)
```

```
44:     for i in prange(ngrid): #loop over all sample positions
45:         mxm,mym,mzm = MDSS[0,i,0],MDSS[1,i,0],MDSS[2,i,0] #initial MDSS
46:         c = 0
47:         for k in range(maxcyc):
48:             X,Y,Z = MH2O[0,i,k],MH2O[1,i,k],MH2O[2,i,k] #precalculated MH2O
49:             #at beginning of each super-cycle. Very little difference if one
50:             #takes average of k and k+1.
51:             for l in range(lS): #loop 'supercycle'
52:                 for m in range(lA):#loop 'subcycle'
53:                     sig = Phase[m]*SupCyc[l] #sign rf
54:                     Xs,Ys,Zs = rfA*sig-0.5*(IA*X),Y*sig,Z*sig
55:                     for n in range(Angle[m]): #loop over points in one pulse
56:                         x = Xs #x-field felt by DSS
57:                         y = -0.5*(cA[n]*Y-sA[n]*Zs) #y field
58:                         z = wDSS+(cA[n]*Z+sA[n]*Ys) #z field
59:                         theta = np.sqrt(x**2+y**2+z**2) #the rest in loop serves
60:                         #to calculate the rotation of DSS
61:                         a,t = np.cos(theta*0.5),-np.sin(theta*0.5)/theta
62:                         at2,t2 = 2*a*t,t*t
63:                         a2 = a*a
64:                         b2,c2,d2 = t2*x*x,t2*y*y,t2*z*z
65:                         ab,ac,ad = at2*x,at2*y,at2*z
66:                         bc,bd = 2*t2*x*y,2*t2*x*z
67:                         cd = 2*t2*y*z
68:                         mxp = (a2+b2-c2-d2)*mxm+(bc+ad)*mym+(bd-ac)*mzm
69:                         myp = (bc-ad)*mxm+(a2-b2+c2-d2)*mym+(ab+cd)*mzm
70:                         mzp = (bd+ac)*mxm+(cd-ab)*mym+(a2-b2-c2+d2)*mzm
71:                         mxm,mym,mzm = mxp,myp,mzp
72:                     Ap = Angle[m]*pi1*sig
73:                     Y,Z = np.cos(Ap)*Y-np.sin(Ap)*Z,np.cos(Ap)*Z+np.sin(Ap)*Y #
74:                     #update MH2O ofter each rf pulse (purely driven by rf)
75:                     c += Angle[m]
76:             MDSS[0,i,k+1],MDSS[1,i,k+1],MDSS[2,i,k+1] = mxp,myp,mzp #store MDSS
77:             #after each cycle
78:     return MDSS
79:
80:
81:
82: def main():
83:     B0 = pa.B0            #main field (minus sign for positive gyrom.)
84:     vDSS = pa.vDSS        #chemical shift difference with H2O (ppm)
85:     mixing = pa.mixing    #spin-lock seq.  "Dipsi2","Garp","cw" or "Waltz16"
86:     tau90 = pa.tau90      #duration 90 pulse spinlock
87:     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:
```

```
132:     tau1 = tau90/(90*unit) #duration of 1 step in calculation
133:     w1 = 2*np.pi/(4.*tau90) #rf amplitude
134:
135:     H2Oini = np.array([[np.cos(i*2*np.pi/ngrid),np.sin(i*2*np.pi/ngrid),0]
136:                         for i in range(ngrid)]).T #initial condition solvent
137:     dt_sc = tau1*np.sum(Angle)*len(SupCyc) #lenght one supercycle
138:     time = dt_sc*range(maxcyc+1)#time from 0 to end mixing, in super cycle inc.
139:     print('Maximum mixing time:',time[-1])
140:
141:     #next 3 lines. Evolution solvent after each supercycle. To a good
142:     #approximation a rotation arount the x-axis, with angular freq.=
143:     #-(3/4)*wDF*Mx(0)
144:     c = np.cos(-(3/4)*wDF*np.outer(H2Oini[0],time))
145:     s = np.sin(-(3/4)*wDF*np.outer(H2Oini[0],time))
146:     MH2O = np.array([np.outer(H2Oini[0],np.ones(maxcyc+1)),
147:                       np.einsum('i,ij->ij',H2Oini[1],c)-
148:                       np.einsum('i,ij->ij',H2Oini[2],s),
149:                       np.einsum('i,ij->ij',H2Oini[2],c)+
150:                       np.einsum('i,ij->ij',H2Oini[1],s)])
151:
152:     MDSS = np.zeros_like(MH2O) #next lines initial MDSS
153:     if iniDSS == "MH2O":
154:         MDSS[:,:,0] = MH2O[:,:,0].copy()
155:     if iniDSS == "iks":
156:         MDSS[0,:,0]+=1.0
157:     if iniDSS == "why":
158:         MDSS[1,:,0]+=1.0
159:     if iniDSS == "zed":
160:         MDSS[2,:,0]+=1.0
161:
162:     rfA  = w1*tau1 #rotation angle rf in 1 time-increment
163:     wDSS  = tau1*vDSS*B0*2*np.pi #rotation angle offset
164:     IA  = tau1*wDF #rotation angle wDF/MH2Ox around x-axis
165:     #next two lines, mixing factors for y and z contributions df (makes very
166:     #little difference to add 0.5 or not). This boils down to a precalculation
167:     #of sin and cos factors for MH2O evolution during pulses
168:     cA  = tau1*wDF*np.cos((np.arange(max(Angle))+0.5)*np.pi/(180*unit))
169:     sA  = tau1*wDF*np.sin((np.arange(max(Angle))+0.5)*np.pi/(180*unit))
170:
171:     lA = len(Angle)
172:     lS = len(SupCyc)
173:
174:     print('Compilation and parallel calculation starts')
175:     MDSS = evolveDSS(MDSS,ngrid,maxcyc,MH2O,SupCyc,Angle,Phase,IA,sA,
```

```
176:                         cA,rfA,wDSS,lA,lS,unit)
177:     #plot for different pfgs before acquistion. 'z' corresponds to, spoil-90
178:     #before the same pfg.
179:     c = 0
180:     for o in order:
181:         co = np.cos(o*2*np.pi*np.arange(ngrid)/(ngrid))/ngrid
182:         si = np.sin(o*2*np.pi*np.arange(ngrid)/(ngrid))/ngrid
183:         MDSSbuX = co.dot(MDSS[0])-si.dot(MDSS[1])
184:         MDSSbuY = co.dot(MDSS[1])+si.dot(MDSS[0])
185:         MDSSbuZx  = co.dot(MDSS[2])
186:         MDSSbuZy =  si.dot(MDSS[2])
187:         if plot:
188:             if plotopt['plane']:
189:                 plt.figure('plane')
190:                 plt.plot(time,MDSSbuX,color = plotcol[c])
191:                 plt.plot(time,MDSSbuY,color = plotcol[c],linestyle='--')
192:             if plotopt['planeabs']:
193:                 plt.figure('planeabs')
194:                 plt.plot(time,np.sqrt(MDSSbuY**2+MDSSbuX**2),color = plotcol[c])
195:             if plotopt['z']:
196:                 plt.figure('z')
197:                 plt.plot(time,MDSSbuZx,color = plotcol[c])
198:                 plt.plot(time,MDSSbuZy,color = plotcol[c],linestyle='--')
199:             if plotopt['zabs']:
200:                 plt.figure('zabs')
201:                 plt.plot(time,np.sqrt(MDSSbuZx**2+MDSSbuZy**2),color = plotcol[c])
202:         c +=1
203:         #can be saved to make own plots, especially to compare
204:         if savedata:
205:             np.savetxt('mag'+str(o)+'.txt',[time,MDSSbuX,MDSSbuY,MDSSbuZx,
206:                                               MDSSbuZy])
207:     plt.show()
208: if __name__ == '__main__':
209:     main()
```