Rapid and safe ASAP acquisition with EXACT NMR

Ikenna Ndukwe, Alexandra Shchukina, Krzysztof Kazimierczuk and Craig P. Butts*

Supplemental Material

The Pulse Sequences

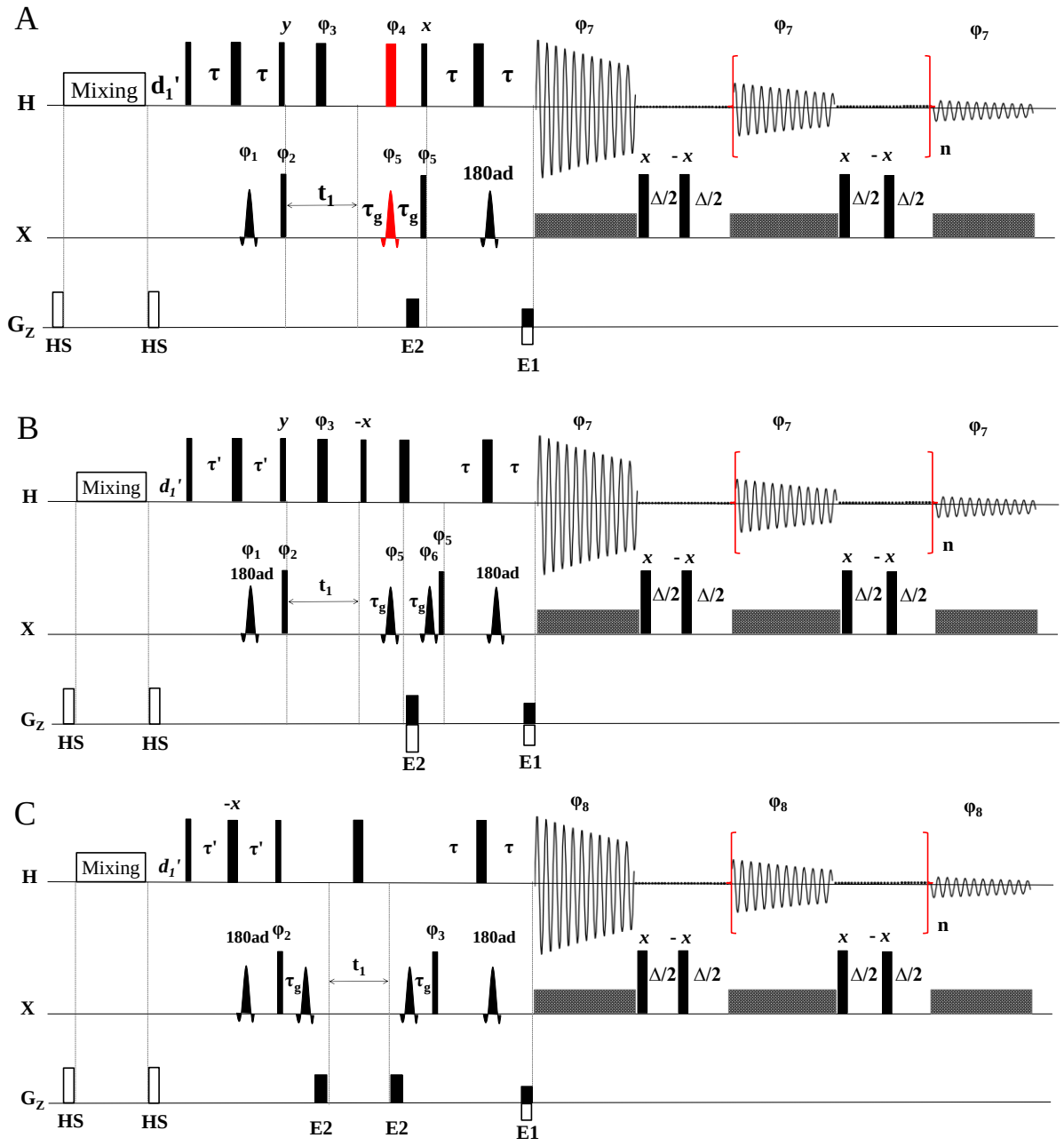


Figure S1: Pulse sequences of (a) EXACT-ASAPHSQC (multiplicity edited) (b) EXACT-ASAPHSQC (non-multiplicity edited) and (c) EXACT-ASAPHMQC experiments. Note that Ernst angle excitation have not been implemented in the EXACT-ASAPHSQC (multiplicity edited) sequence. Wide and narrow filled black boxes are 180° and 90° pulses respectively while the red ^{1}H pulse is 180° for discrimination of CH/CH_3 and CH_2 or 90° for CH only correlations. Black shaped pulses represent ^{13}C cawurst-10 pulses[1] (~102KHz, 196µs) while the red shaped pulse is a REBURP pulse[2] (30KHz, 162µs). The traditional recovery delay has been replaced with a short 60ms period encompassing an ~ 25ms DIPSI-2 mixing sequence[3] flanked by 2.0ms homospoil gradients of power 11.99Gcm^{-1} and 9.20Gcm^{-1} respectively followed by a short delay

d_1' of 31ms. Delays are: $\tau = 1/(4 * {}^1J_{CH})$ where J_{CH} = 146Hz, τ' = 1.2ms, τ_g is the encoding gradient duration and recovery delay except for the EXACT-ASAPHSQC (multiplicity edited) sequence where it equals 2*τ, Δ is user defined and is set to 21ms in the examples presented here. (Δ = 21ms and datachunk lengths were iterated between 15.5, 11.7, 9.6 and 5.2ms).The phase cycle is given as; ϕ_1 = x; ϕ_2 = *x, -x*; ϕ_3 = *x, x, -x, -x*; ϕ_4 = *-x, -x, x, x*; ϕ_5 = *-x, -x, -x, -x, x, x, x, x*; ϕ_6 = *x, x, x, x, -x, -x, -x, -x*; ϕ_7 = *x, -x, x, -x, -x, x, -x, x* and ϕ_8 = *x, -x, -x, x*. Coherence selection: (a) a 2ms encoding gradient (E1) and a 1ms decoding gradient (E2) of power in Gcm^{-1}, E1:E2 (9.9958, 5.0287) while the sign of E2 is switched between echo-antiecho acquisition; (b) 1ms encoding/decoding gradients (E1 and E2) of power in Gcm^{-1} switched between echo-antiecho acquisition, (E1, E2) – (9.9958, 7.4814) and (7.9868, 9.9958) and in (c) with 1ms encoding/decoding gradients of power in Gcm^{-1}, E1:E2 (9.9958:5.0287) and switching the sign of E2 between echo-antiecho acquisition.

Experimental

The EXACT-ASAPHSQC and EXACT-ASAPHMQC experiments (Figures S1) were acquired, on a 500MHz NMR spectrometer equipped with a room temperature probe (298K). ^{1}H and ^{13}C 90 degree pulse widths were 7.6μs@25W and 9.23μs@30W respectively. Spectra were acquired with a 0.105s FID (1050 total data points) acquired over a 10kHz ^{1}H spectrum width centred at 5ppm and 40% sampled in F2 (420 acquired data points). 96 real t1 increments over a 23 KHz ^{13}C spectrum width (centred at 75ppm) was acquired in a single scan using a strychnine sample of 30mg in 0.7ml $CDCl_3$. These experiments were acquired as a phase-sensitive (echo-antiecho) array, taking 37 seconds in total (including 8 dummy scans prior to acquisition). The resulting EXACT FIDs were Fourier Transformed to give a spectrum which appear grossly correct, but an F_2 slice reveal artefacts which can obscure weak peaks in the spectrum (Figures S2a and S2c). The correct spectrum is recovered using Iterative Soft Thresholding (IST)[4-6] with a total of 100 iterations and virtual echo option[7] (Figures S2b and S2d). The 1D IST algorithm from mddnmr program (available from mddnmr.spektrino.com) was used to reconstruct missing points of the burst-sampled FID for each point of F1 dimension. The python script was used as a "wrapper" to prepare data and perform reconstruction with standard compressed sensing (CS) module of mddnmr used normally for 2D NUS spectra. After the reconstruction, spectra were zero filled to 1024*2048 (t_1*t_2).

The EXACT-ASAPHSQC sequence (Figure S1a) was also acquired at 50%, 25% and 12.5% t1 increments (that is 48, 24 and 12 real NUS t1 increments) and the resulting spectra are shown in Figures S3a, S3b and S3c respectively. The corresponding reconstructed spectra – Figures S3d, S3e and S3f – were recovered using IST algorithm after 100, 20 and 15 iterations respectively. The virtual echo option was also used for better reconstruction.[7] The 2D IST algorithm from the mddnmr program (available from mddnmr.spektrino.com) was used to reconstruct all missing points of burst/non-uniformly-sampled 2D FID at once. The python script was used as a "wrapper" to prepare data and perform reconstruction with standard CS module of mddnmr used normally for 3D NUS spectra. After the reconstruction, spectra were zero filled to 1024*2048 (t_1*t_2).

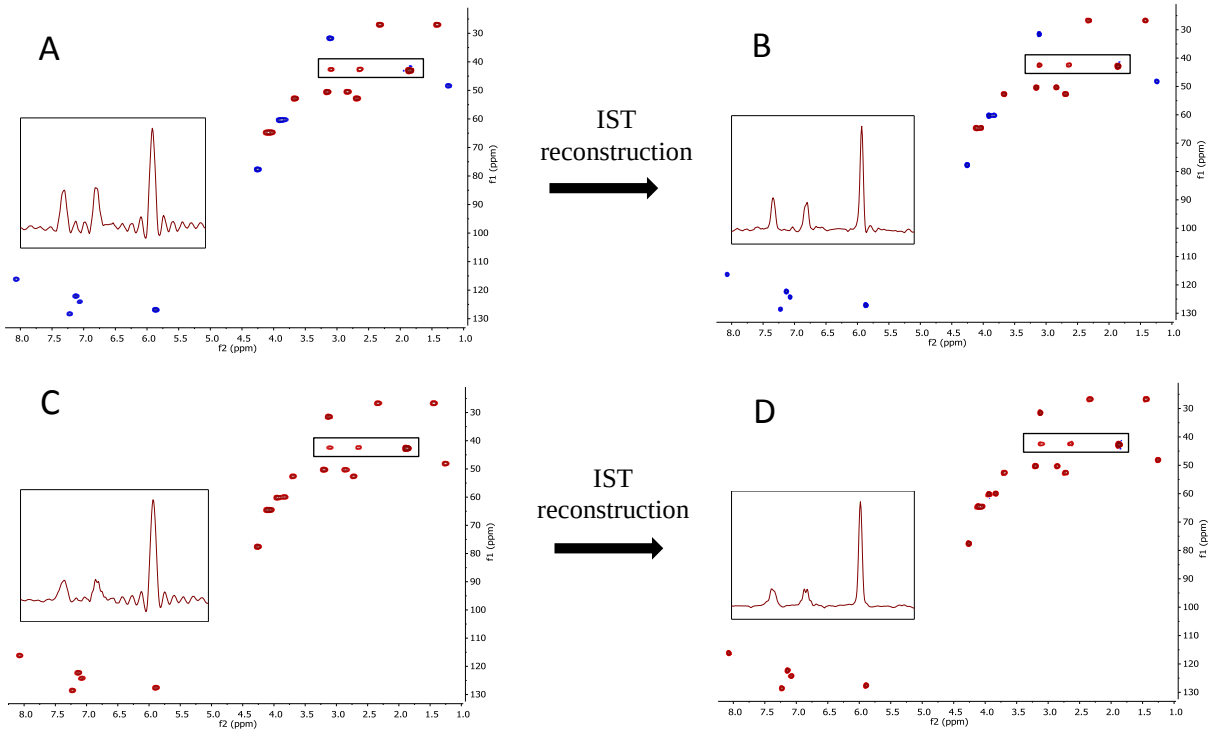


Figure S2: The Fourier transformed spectra of strychnine EXACT datasets (i.e spectra convolved with the sampling scheme's point spread function – see reference[8]) acquired with (a) the EXACT-ASAPHSQC sequence (multiplicity edited) and (c) the EXACT-ASAPHMQC sequence with their respective IST reconstructed spectra (b) and (d). All spectra are processed with Gaussian window function and zero-filled to 1024*2048 (t_1*t_2). 1D traces of three peaks in the same ^{13}C slice are also shown inset revealing artefact suppression after reconstruction with IST.

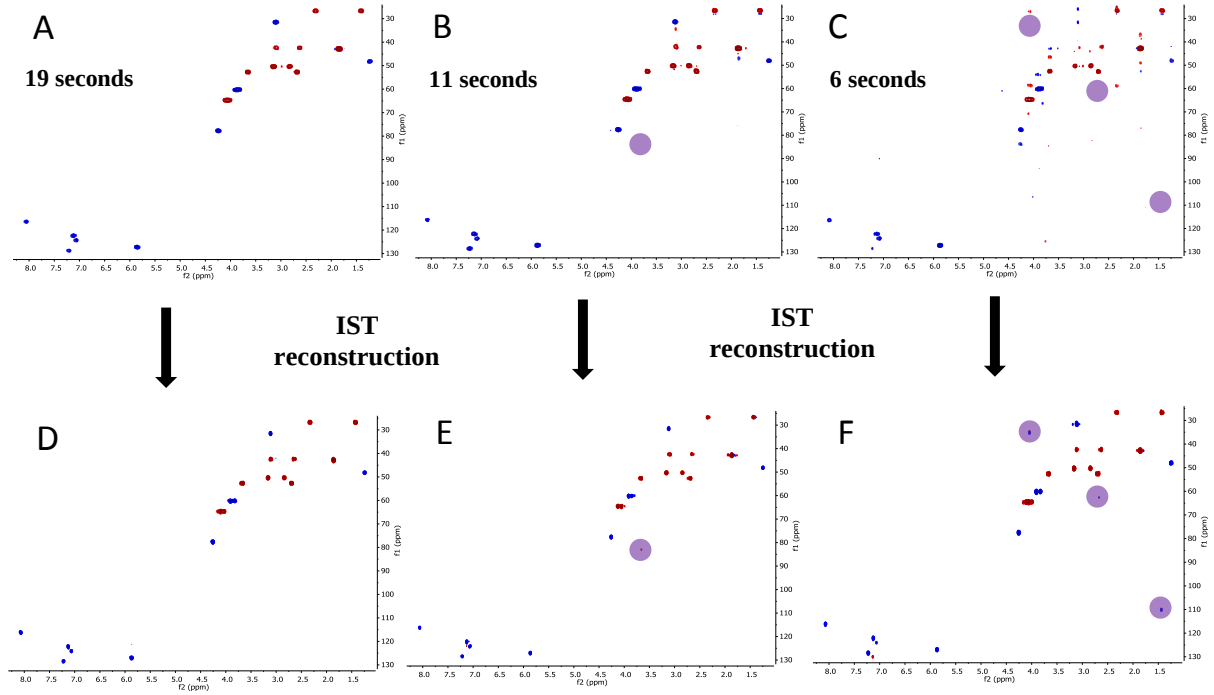


Figure S3: Strychnine spectra resulting from the Fourier transformation of the zero-augmented NUS datasets acquired using the EXACT-ASAPHSQC sequence reported in this work with application of 'standard' random NUS in the indirect dimension at (a) 50 % (b) 25 % and (c) 12.5 % t1 increments. The IST reconstructed spectra are shown in (d), (e) and (f) respectively. Peaks highlighted with purple cycles in the IST spectra are reconstruction artefacts which are not present in the FT spectrum (also highlighted with purple cycles).

Processing Macros

All spectra used in this publication have been processed with the IST algorithm using standard compressed sensing (CS) module of mddnmr (mddnmr.spektrino.com) – which is normally used for 2D or 3D NUS spectra. The python script(s) used as "wrapper" to prepare data and perform reconstruction are all available at http://nmr.cent.uw.edu.pl/index.php/download/category/5-pure-shift (Download file: Python wrappers for mdd).

Duty cycle

Duty cycle is calculated thus:

$$Duty\ Cycle\ (\%) = \frac{AT}{Recycle\ time} * 100\ \%$$

AT = Duration of data sampling periods (does not include gaps)

Recycle time = Relaxation delay + total acquisition time (including gaps)

Sensitivity comparison of multiplicity edited and non-edited EXACT-ASAPHSQC

The absolute peak intensities of correlations in the strychnine spectra of both EXACT-ASAPHSQC experiments (multiplicity edited and non-multiplicity edited) are compared. The respective spectra are acquired and processed with IST algorithm using the parameters outlined in the experimental section above.

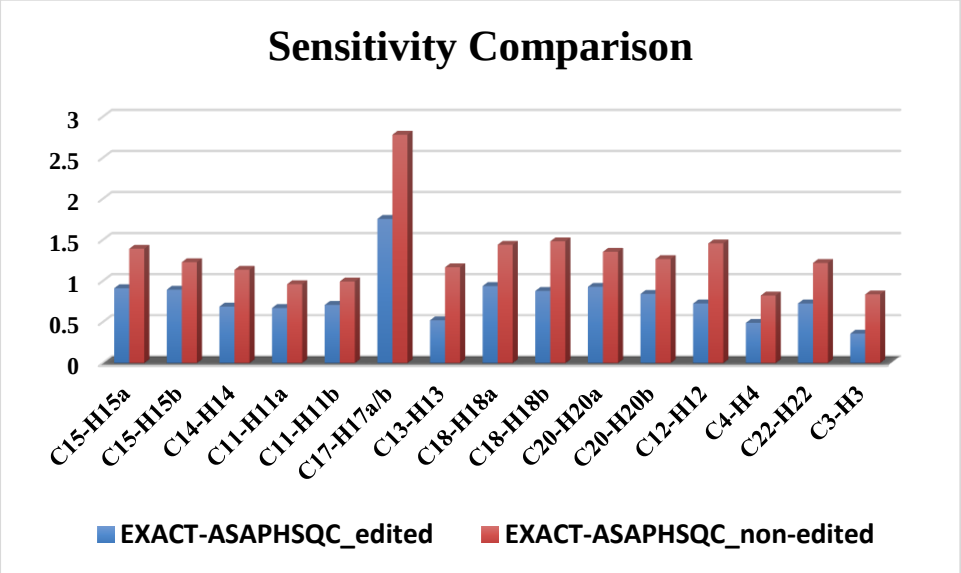
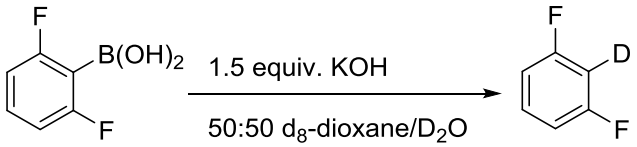


Figure S4: Histogram of the absolute peak intensities of strychnine peaks from multiplicity edited EXACT-ASAPHSQC (blue) and non-edited EXACT-ASAPHSQC (red) reconstructed with IST algorithm.

The acquisition and processing parameters for the kinetic study (equation S1) is the same as that described in the experimental section above and the data obtained are shown in Table S1 below. The times reported are the mid-points of the respective acquisition.



S1

Table S1: Absolute peak intensities for H3/5 from ^{1}H NMR spectra (left) and H3/5-C3/5 for EXACT-ASAPHSQC spectra (right) of the protodeboronation reaction of 2,6-difluorophenylboronic acid with 1.5 equiv. of KOH in 50:50 deuterated dioxane/D_2O. Reported concentrations are derived directly from the absolute intensities following calibration versus starting material immediately prior to initiating the reaction (starting concentration 75mM).

PROTON			EXACT-ASAPHSQC (non-edited)		
Time (s)	Concentration (mM)	Absolute peak Intensities	Time (s)	Concentration (mM)	Absolute peak Intensities
262	66.27	6841.3737	320.5	58.77	1.7850
396	62.02	6402.8833	452.5	58.87	1.7881
530	58.11	5998.9467	592.5	55.45	1.6843
676	54.13	5588.6650	739.5	50.93	1.5469
822	50.42	5205.2449	883.5	47.46	1.4417
965	47.06	4858.5348	1027.5	44.59	1.3544
1108	43.88	4530.1736	1169.5	39.40	1.1969
1251	41.00	4232.8183	1312.5	35.98	1.0930
1393	38.23	3946.2715	1454.5	34.44	1.0460
1534	35.66	3681.3899	1595.5	30.78	0.9349
1676	33.29	3436.9960	1741.5	31.97	0.9712
1824	30.96	3195.8790	1886.5	30.15	0.9159
1967	28.86	2979.5564	2030.5	24.65	0.7488
2112	26.88	2775.0395	2176.5	23.46	0.7125
2257	25.02	2582.8509	2319.5	22.73	0.6904
2401	23.36	2411.4983	2465.5	19.81	0.6018
2547	21.72	2242.5484	2611.5	19.06	0.5790
2693	20.22	2087.6058	2756.5	18.89	0.5739
2839	18.81	1941.5713	2904.5	17.52	0.5321
2987	17.55	1812.0547	3050.5	13.63	0.4140
3132	16.36	1688.7765	3195.5	15.37	0.4670
3277	15.17	1566.3371	3339.5	15.47	0.4699
3420	14.15	1461.0551	3482.5	11.75	0.3569
3564	13.18	1360.5082	3625.5	12.82	0.3895
3705	12.29	1268.6642	3774.5	13.34	0.4053
3856	11.38	1174.6253	3918.5	13.77	0.4182
3999	10.60	1094.7004	4059.5	10.28	0.3123
4140	9.91	1022.9093	4201.5	7.18	0.2182
4282	9.21	951.2468	4342.5	8.60	0.2611

4423	8.67	894.9124	4483.5	6.18	0.1876
4566	8.03	829.0136	4628.5	7.61	0.2313
4711	7.53	776.9528	4775.5	8.44	0.2563
4856	6.97	719.6425	4919.5	5.71	0.1734
5000	6.54	675.1471	5061.5	7.63	0.2317
5143	6.11	631.2429	5206.5	6.17	0.1874
5288	5.68	586.0588	5350.5	5.08	0.1542
5431	5.29	546.5841	5490.5	3.26	0.0989
5571	4.96	512.0950	5629.5	3.24	0.0985

Varian Spectrometer Pulse program

EXACT ASAP-HSQC

```
/* EXACT_ASAPHSQC - ASAP - Acceleration by Sharing Adjacent Polarization
                    E. Kupce and R. Freeman, Magn. Reson. Chem., v.45 pp.2-4 (2007).
                      EXACT - Extended Acquisition Time

                    hsglvl  :  Homospoil gradient level (DAC units)
                    hsgt    :        Homospoil gradient time
                    gzlvlE  :        Encoding gradient level
                    gtE     :        Encoding gradient time
          EDratio :     Encode/Decode ratio
                    gstab   :        Recovery delay
                    j1xh    :        One-bond XH coupling constant
                    pwxlvl  : X-nucleus pulse power
                    pwx     :        X-nucleus 90 deg pulse width
                    d1      :        Relaxation delay
                    d2      :        Evolution delay
                    asap    :    Flag to switch between ASAP and conventional mode
          mix     :     Mixing time for polarization sharing, ca 50 ms (asap)
          adiabatic:    Flag to use adiabatic pulses (y = preferred mode)
                    tpwr_cf :     H1 amplifier compression factor
                    lkgate_flg:       Lock gating flag

Eriks Kupce, Agilent UK 11 Nov. 2006: Original pulse sequence ASAP_HMQC
Bert Heise, Agilent UK Feb. 2008: Chempack compatible version ASAP_HMQC
Bert Heise, Agilent UK Jul. 2010: Lock gating added for better peak shapes ASAP_HMQC

Ikenna E. Ndukwe and Dr. Craig P. Butts, University of Bristol UK Feb. 2016: Implementation of ASAP_HSQC - D.
Schulze-Sunninghausen, J. Becker and B. Luy, J. Am. Chem. Soc., 2014, 136, 1242-1245, and EXACT NMR

Ikenna E. Ndukwe and Dr. Craig P. Butts, University of Bristol UK Apr. 2016: Impementation of multiplicity edited version
of EXACT_ASAPHSQC.
*/

#include <standard.h>
#include <chempack.h>
#ifndef   FIRST_FID
#include <Pbox_psg.h>
#endif

extern int dps_flag;

static shape mixsh, ad180, rtsh;

/*---------------------------------------------
Phase tables for EXACT_ASAPHSQC
---------------------------------------------*/
static int     ph1[1] = {0},   //v1
                        ph2[2] = {0,2},   //v2
                        ph3[4] = {0,0,2,2},   //v3
                        ph4[4] = {2,2,0,0},   //v4
                        ph5[8] = {2,2,2,2,0,0,0,0},   //v5
                        ph6[8] = {0,0,0,0,2,2,2,2},   //v6
                        ph7[8] = {0,2,0,2,2,0,2,0};   //oph


pulsesequence()
{
  double   slpwrT = getval("slpwrT"),
                        slpwT = getval("slpwT"),
                        mixT = getval("mixT"),
                        mix = getval("mix"),
          mult = getval("mult"),
                        pw180 = 0.0,
```

```
                tpwr_cf = getval("tpwr_cf"),
            j1xh = getval("j1xh"),
            hsglvl = getval("hsglvl"),
            hsgt = getval("hsgt"),
            gzlvlE = getval("gzlvlE"),
            gtE = getval("gtE"),
            rof3 = getval("rof3"),
                EDratio = getval("EDratio"),
                gstab = getval("gstab"),
                pwxlvl = getval("pwxlvl"),
                pwx = getval("pwx"),
                taua=getval("taua"),       /* asap delay ca 0.3/J */
                tauA=getval("tauA"),    //compensation for tauB and tauD
        tauB=getval("tauB"),        //effect of rof2
                tauD=getval("tauD"),        //effect of alfa
        EDratio1, EDratio2,tau,taug,tauM,evolcorr,evolcorr2,
        npoints,dcor,rtd,nptot,npnus[4];
char   asap[MAXSTR], adiabatic[MAXSTR],lkgate_flg[MAXSTR],cmd[MAXSTR],slpatT[MAXSTR],rburp[MAXSTR];
int    ncycles=0, icosel=-1,Count=0,n,i,npchunk,
      prgcycle = (int)(getval("prgcycle")+0.5),
      iphase = (int)(getval("phase")+0.5);

getstr("asap",asap);
getstr("adiabatic",adiabatic);
getstr("lkgate_flg",lkgate_flg);
getstr("slpatT",slpatT);
getstr("rburp",rburp);

 if (strcmp(slpatT,"mlev17c") &&
     strcmp(slpatT,"dipsi2") &&
     strcmp(slpatT,"dipsi3") &&
     strcmp(slpatT,"mlev17") &&
     strcmp(slpatT,"mlev16"))
     abort_message("SpinLock pattern %s not supported!.\n", slpatT);

if(j1xh<1.0) j1xh=150.0;
tau = 1 / (4*(getval("j1xh")));
evolcorr = 2*pw+2*rof1;
evolcorr2 = 3*pw+2*rof1;

  tauM = 2*tau + getval("tauC");
  taug = gtE + gstab + 2.0*GRADIENT_DELAY;


if (FIRST_FID)
{
  if(adiabatic[A] == 'y')
    ad180 = pbox_ad180("ad180", pwx, pwxlvl);    /* make adiabatic 180 */
  sprintf(cmd, "Pbox wu2mix -u %s -w \"DIPSI2 20k/0.4m\" -p %.0f -l %.2f -s 2.0",userdir,
  tpwr, 1.0e6*pw*tpwr_cf);
  system(cmd);
  mixsh = getDsh("wu2mix");

  if(rburp[0] == 'y')
     {
       sprintf(cmd, "Pbox Brp_shp -u %s -w \"reburp 30k %f\" -p %.0f -l %.2f -s 2.0",userdir,dof,pwxlvl, 1.0e6*pwx);
       system(cmd);
       rtsh= getRsh("Brp_shp");
     }
}

if(adiabatic[A] == 'y')
  pw180 = ad180.pw;

ncycles = (int) (0.5 + mix/mixsh.pw);

if ((0.5/j1xh) < (gtE+gstab))
```

```
{
  text_error("0.5/1jxh must be greater than gtE+gstab\n");
  psg_abort(1);
}

settable(t1,1,ph1);
settable(t2,2,ph2);
settable(t3,4,ph3);
settable(t4,4,ph4);
settable(t5,8,ph5);
settable(t6,8,ph6);
settable(t7,8,ph7);

getelem(t1,ct,v1);
getelem(t2,ct,v2);
getelem(t3,ct,v3);
getelem(t4,ct,v4);
getelem(t5,ct,v5);
getelem(t6,ct,v6);
getelem(t7,ct,oph);

    npoints = 0.400 * np;
    npoints = (double)((int)((npoints)));

    rtd = 0.021;

    dcor = ((np - npoints)/(2.0*sw)) - (3.0*rtd) + 0.0001;

     i = 0;
     nptot = 0.0;
     npchunk = (int)((37.0*npoints/100.0));
     n = npchunk % 2;
     if (n == 0)
        {
          npnus[0] = (double)((int)((37.0*npoints/100.0)));
          printf("npnus 1 = %f\n",npnus[0]);
          nptot = nptot + npnus[0];
          printf("nptot = %f\n",nptot);
        }
      else
        {
          npnus[0] = ((double)((int)((37.0*npoints/100.0)))) - 1.0;
          printf("npnus 1 = %f\n",npnus[0]);
          nptot = nptot + npnus[0];
          printf("nptot = %f\n",nptot);
        }

      npchunk = (int)((28.0*npoints/100.0));
      n = npchunk % 2;
      if (n == 0)
        {
          npnus[1] = (double)((int)((28.0*npoints/100.0)));
          printf("npnus 2 = %f\n",npnus[1]);
          nptot = nptot + npnus[1];
          printf("nptot = %f\n",nptot);
        }
      else
        {
          npnus[1] = ((double)((int)((28.0*npoints/100.0)))) - 1.0;
          printf("npnus 2 = %f\n",npnus[1]);
          nptot = nptot + npnus[1];
          printf("nptot = %f\n",nptot);
        }

      npchunk = (int)((23.0*npoints/100.0));
      n = npchunk % 2;
      if (n == 0)
```

```
      {
        npnus[2] = (double)((int)((23.0*npoints/100.0)));
        printf("npnus 3 = %f\n",npnus[2]);
        nptot = nptot + npnus[2];
        printf("nptot = %f\n",nptot);
      }
    else
      {
        npnus[2] = ((double)((int)((23.0*npoints/100.0)))) - 1.0;
        printf("npnus 3 = %f\n",npnus[2]);
        nptot = nptot + npnus[2];
        printf("nptot = %f\n",nptot);
      }

    npnus[3] = npoints - nptot;
    printf("npnus 4 = %f\n",npnus[3]);
    nptot = nptot + npnus[3];
    printf("nptot = %f\n",nptot);


if ((phase1 == 2) || (phase1 == 5))
      icosel = 1;

EDratio1 = EDratio/EDratio;
EDratio2 = (EDratio - 1.0)/EDratio;

assign(zero,v6);
assign(ct,v17);
assign(zero,v18);
assign(zero,v19);

if (getflag("prgflg") && (satmode[0] == 'y') && (prgcycle > 1.5))
  {
     hlv(ct,v17);
     mod2(ct,v18); dbl(v18,v18);
     if (prgcycle > 2.5)
       {
          hlv(v17,v17);
          hlv(ct,v19); mod2(v19,v19); dbl(v19,v19);
       }
  }

add(oph,v18,oph);
add(oph,v19,oph);

if (iphase == 2)
  {
    EDratio1 = EDratio/(EDratio + 1.0);
    EDratio2 = (EDratio + 1.0)/(EDratio + 1.0);
  }

initval(2.0*(double)(((int)(d2*getval("sw1")+0.5)%2)),v10);

 if (mult > 0.5)
   {
            add(v2, v10, v2);
            add(oph, v10, oph);
   }
 else
   {
            add(v1,v10,v1);
            add(v2, v10, v2);
            add(oph, v10, oph);
   }


status(A);
```

```
  if (lkgate_flg[0] == 'y')  lk_hold(); /* turn lock sampling off */
  if (asap[A] == 'y')
  {
    mix = 7.0e-5 + 2.0*POWER_DELAY + 2.0*hsgt + mixsh.pw*ncycles;
    delay(1.0e-5);
    obspower(mixsh.pwr);
    zgradpulse(hsglvl,hsgt);
    delay(5.0e-5);

    obsunblank(); xmtron();
    obsprgon(mixsh.name, 1.0/mixsh.dmf, mixsh.dres);
    delay(mixsh.pw*ncycles);
          obsprgoff(); obsblank();
    xmtroff();


    delay(1.0e-5);
    zgradpulse(0.7674*hsglvl,hsgt);
    obspower(tpwr);
    delay(d1 - mix);
   }
   else
   {
    delay(1.0e-5);
    obspower(tpwr);
    zgradpulse(hsglvl,2.0*hsgt);
    rgpulse(pw, zero, rof1, rof1);
    zgradpulse(hsglvl,hsgt);
    rgpulse(pw, one, rof1, rof1);
    zgradpulse(0.5*hsglvl,hsgt);
    delay(d1 - 2.0*hsgt - 4.0*rof1 - 2.0*pw - 1.0e-5 - POWER_DELAY);
   }


 if (satmode[0] == 'y')
  {
    if ((d1-satdly) > 0.02)
          delay(d1-satdly);
    else
          delay(0.02);
    if (getflag("slpsat"))
      {
          shaped_satpulse("relaxD",satdly,zero);
          if (getflag("prgflg"))
            shaped_purge(v6,zero,v18,v19);
      }
    else
      {
          satpulse(satdly,zero,rof1,rof1);
          if (getflag("prgflg"))
            purge(v6,zero,v18,v19);
      }
  }

 if (getflag("wet"))
   wet4(zero,one);

status(B);
  decpower(pwxlvl);

    rgpulse(pw,zero,rof1,rof1);

  if (mult > 0.5)
    {
      delay(tau - pw - 2.0*rof1 + pw180/2.0);
```

```
    rgpulse(2.0*pw,zero,rof1,0.0);
    decshaped_pulse(ad180.name,ad180.pw,v1,0.0,rof1);

    delay(tau - pw - 2.0*rof1 - pw180/2.0);
  }
else
  {
    delay(taua/2.0 - pw - 2.0*rof1 + pw180/2.0);

    rgpulse(2.0*pw,zero,rof1,0.0);
    decshaped_pulse(ad180.name,ad180.pw,v1,0.0,rof1);

    delay(taua/2.0 - pw - 2.0*rof1 - pw180/2.0);
  }

   simpulse(pw,pwx,one,v2,rof1,rof1);

if (mult > 0.5)
  {
        delay(d2/2);
        rgpulse(2.0*pw,v3,rof1,rof1);
        delay(d2/2);

        delay(tauM);

  if (rburp[0] == 'y')
    {
      decpower(rtsh.pwr);
      simshaped_pulse("hard",rtsh.name,mult*pw,rtsh.pw,v4,v5,rof1,rof1);
      decpower(pwxlvl);

      delay(tauM - 2.0*gtE - gstab - 2.0*GRADIENT_DELAY + evolcorr);
      zgradpulse(gzlvlE, 2.0*gtE);
      delay(gstab);
    }
  else
    {
      simshaped_pulse("hard",ad180.name,mult*pw,ad180.pw,v4,v5,rof1,rof1);

      delay(tauM - 2.0*gtE - gstab - 2.0*GRADIENT_DELAY + evolcorr);
      zgradpulse(gzlvlE, 2.0*gtE);
      delay(gstab);
    }

    simpulse(pw, pwx, two, v5, rof1, 0.0);
  }
else
  {
        delay(d2/2 + taug);
        rgpulse(2.0*pw,v3,rof1,rof1);
        delay(d2/2 + taug);

    rgpulse(pw,two,rof1,0.0);
    decshaped_pulse(ad180.name,ad180.pw,v5,0.0,rof1);

        delay(taug + evolcorr2/2.0 + rof1);
        rgpulse(2.0*pw,zero,rof1,rof1);
        zgradpulse(gzlvlE*EDratio1,gtE);
        delay(gstab + evolcorr2/2.0 - rof1);

    decshaped_pulse(ad180.name,ad180.pw,v6,rof1,rof1);
        decrgpulse(pwx,v5,rof1,0.0);
  }

  delay(0.25/j1xh - pw - rof1 + pw180/2.0);

  rgpulse(2.0*pw,zero,rof1,0.0);
```

```
    decshaped_pulse(ad180.name,ad180.pw,zero,0.0,rof1);

    delay(0.25/j1xh - gtE - gstab - 2.0*GRADIENT_DELAY - pw - 2.0*rof1 - pw180/2.0);

  if (mult > 0.5)
    {
      zgradpulse(icosel*2.0*gzlvlE/EDratio, gtE);
      delay(gstab);
    }
  else
    {
      zgradpulse(gzlvlE*EDratio2,gtE);
      delay(gstab);
    }


   if (lkgate_flg[0] == 'y') lk_sample(); /* turn lock sampling on */
   decpower(dpwr);
   delay(rof2 - POWER_DELAY);

        obsblank();
        startacq(alfa);

        status(C);
            acquire(npnus[i],1.0/sw);
            rcvroff();

        status(B);
            decpower(pwxlvl);
            decrgpulse(2.0*pwx,zero,rof1,rof1);
            delay(rtd/2.0 + POWER_DELAY);
                        delay(tauA);
            decrgpulse(2.0*pwx,two,rof1,rof1);
            decpower(dpwr);
            delay(rtd/2.0 - POWER_DELAY);
            obsblank();
                        delay(tauB - 2.0e-6);           //obsblank transition of 2.0e-6
            rcvron();     //this includes rof3
                        delay(tauD - rof3);

/*------------------------------------------------------------------
        Loops for more chunks
------------------------------------------------------------------*/

        for (i = 1; i < 3; i++)
        {

        status(C);
            acquire(npnus[i],1.0/sw);
            rcvroff();

        status(B);
            decpower(pwxlvl);
            decrgpulse(2.0*pwx,zero,rof1,rof1);
            delay(rtd/2.0 + POWER_DELAY);
                        delay(tauA);
            decrgpulse(2.0*pwx,two,rof1,rof1);
            decpower(dpwr);
            delay(rtd/2.0 - POWER_DELAY);
            obsblank();
                        delay(tauB - 2.0e-6);           //obsblank transition of 2.0e-6
            rcvron();     //this includes rof3
                        delay(tauD - rof3);
        }

/*------------------------------------------------------------------
        Acquisition of last chunk
```

```
------------------------------------------------------------------*/
        status(C);
            acquire(npnus[3],1.0/sw);
            rcvroff();

        status(B);
            delay(dcor);
            endacq();
}
```

EXACT ASAP-HMQC

```
/* EXACT_ASAPHMQC - ASAP - Acceleration by Sharing Adjacent Polarization
                    E. Kupce and R. Freeman, Magn. Reson. Chem., v.45 pp.2-4 (2007).
                      EXACT - Extended Acquisition Time

                    hsglvl  :  Homospoil gradient level (DAC units)
                    hsgt    :       Homospoil gradient time
                    gzlvlE  :       Encoding gradient level
                    gtE     :       Encoding gradient time
        EDratio :   Encode/Decode ratio
                    gstab   :       Recovery delay
                    j1xh    :       One-bond XH coupling constant
                    pwxlvl  :  X-nucleus pulse power
                    pwx     :       X-nucleus 90 deg pulse width
                    d1      :       Relaxation delay
                    d2      :       Evolution delay
                    asap    :    Flag to switch between ASAP and conventional mode
        mix     :   Mixing time for polarization sharing, ca 50 ms (asap)
        adiabatic:  Flag to use adiabatic pulses (y = preferred mode)
                    tpwr_cf :   H1 amplifier compression factor
                    lkgate_flg:     Lock gating flag

Eriks Kupce, Agilent UK 11 Nov. 2006: Original pulse sequence
Bert Heise, Agilent UK Feb. 2008: Chempack compatible version
Bert Heise, Agilent UK Jul. 2010: Lock gating added for better peak shapes

Ikenna E. Ndukwe and Dr. Craig P. Butts, University of Bristol UK Feb. 2016: Implementation of EXACT NMR
*/

#include <standard.h>
#include <chempack.h>
#ifndef   FIRST_FID
#include <Pbox_psg.h>
#endif

static shape mixsh, ad180;

static int     ph1[2] = {0,2},
                        ph2[4] = {0,0,2,2},
                        ph3[4] = {0,2,2,0};

pulsesequence()
{
  double mix = getval("mix"),
             pw180 = 0.0,
             tpwr_cf = getval("tpwr_cf"),
        j1xh = getval("j1xh"),
        hsglvl = getval("hsglvl"),
        hsgt = getval("hsgt"),
        gzlvlE = getval("gzlvlE"),
        gtE = getval("gtE"),
        rof3 = getval("rof3"),
        EDratio = getval("EDratio"),
        gstab = getval("gstab"),
```

```
            pwxlvl = getval("pwxlvl"),
            pwx = getval("pwx"),
            taua=getval("taua"),       /* asap delay ca 0.3/J */
            tauA=getval("tauA"),       //compensation for tauB and tauD
            tauB=getval("tauB"),       //effect of rof2
            tauD=getval("tauD"),       //effect of alfa
      npoints,dcor,rtd,nptot,npnus[4];
char   asap[MAXSTR], adiabatic[MAXSTR],lkgate_flg[MAXSTR],cmd[MAXSTR];
int    ncycles=0, icosel=1,Count=0,n,i,npchunk,
      iphase = (int)(getval("phase")+0.5);

getstr("asap",asap);
getstr("adiabatic",adiabatic);
getstr("lkgate_flg",lkgate_flg);
if(j1xh<1.0) j1xh=150.0;

if (FIRST_FID)
{
 if(adiabatic[A] == 'y')
   ad180 = pbox_ad180("ad180", pwx, pwxlvl);    /* make adiabatic 180 */
 sprintf(cmd, "Pbox wu2mix -u %s -w \"WURST2m 20k/0.4m\" -p %.0f -l %.2f -s 2.0",userdir,
 tpwr, 1.0e6*pw*tpwr_cf);
 system(cmd);
 mixsh = getDsh("wu2mix");
}

if(adiabatic[A] == 'y')
 pw180 = ad180.pw;

ncycles = (int) (0.5 + mix/mixsh.pw);

if ((0.5/j1xh) < (gtE+gstab))
{
 text_error("0.5/1jxh must be greater than gtE+gstab\n");
 psg_abort(1);
}

settable(t1,2,ph1);
settable(t2,4,ph2);
settable(t3,4,ph3);

assign(zero,v4);
getelem(t1,ct,v1);
getelem(t3,ct,oph);

   npoints = 0.400 * np;
   npoints = (double)((int)((npoints)));

   rtd = 0.021;

   dcor = ((np - npoints)/(2.0*sw)) - (3.0*rtd) + 0.0001;

    i = 0;
    nptot = 0.0;
    npchunk = (int)((37.0*npoints/100.0));
    n = npchunk % 2;
    if (n == 0)
      {
        npnus[0] = (double)((int)((37.0*npoints/100.0)));
        printf("npnus 1 = %f\n",npnus[0]);
        nptot = nptot + npnus[0];
        printf("nptot = %f\n",nptot);
      }
    else
      {
        npnus[0] = ((double)((int)((37.0*npoints/100.0)))) - 1.0;
        printf("npnus 1 = %f\n",npnus[0]);
```

```
        nptot = nptot + npnus[0];
        printf("nptot = %f\n",nptot);
       }

    npchunk = (int)((28.0*npoints/100.0));
    n = npchunk % 2;
    if (n == 0)
      {
        npnus[1] = (double)((int)((28.0*npoints/100.0)));
        printf("npnus 2 = %f\n",npnus[1]);
        nptot = nptot + npnus[1];
        printf("nptot = %f\n",nptot);
      }
    else
      {
        npnus[1] = ((double)((int)((28.0*npoints/100.0)))) - 1.0;
        printf("npnus 2 = %f\n",npnus[1]);
        nptot = nptot + npnus[1];
        printf("nptot = %f\n",nptot);
      }

    npchunk = (int)((23.0*npoints/100.0));
    n = npchunk % 2;
    if (n == 0)
      {
        npnus[2] = (double)((int)((23.0*npoints/100.0)));
        printf("npnus 3 = %f\n",npnus[2]);
        nptot = nptot + npnus[2];
        printf("nptot = %f\n",nptot);
      }
    else
      {
        npnus[2] = ((double)((int)((23.0*npoints/100.0)))) - 1.0;
        printf("npnus 3 = %f\n",npnus[2]);
        nptot = nptot + npnus[2];
        printf("nptot = %f\n",nptot);
      }

    npnus[3] = npoints - nptot;
    printf("npnus 4 = %f\n",npnus[3]);
    nptot = nptot + npnus[3];
    printf("nptot = %f\n",nptot);

if (iphase == 2)
  icosel = -1;

initval(2.0*(double)((int)(d2*getval("sw1")+0.5)%2),v10);
add(v1,v10,v1);
add(v4,v10,v4);
add(oph,v10,oph);

status(A);

   if (lkgate_flg[0] == 'y')  lk_hold(); /* turn lock sampling off */
   if (asap[A] == 'y')
   {
     mix = 7.0e-5 + 2.0*POWER_DELAY + 2.0*hsgt + mixsh.pw*ncycles;
     delay(1.0e-5);
     obspower(mixsh.pwr);
     zgradpulse(hsglvl,hsgt);
     delay(5.0e-5);

     obsunblank(); xmtron();
     obsprgon(mixsh.name, 1.0/mixsh.dmf, mixsh.dres);
     delay(mixsh.pw*ncycles);
          obsprgoff(); obsblank();
     xmtroff();
```

```
    delay(1.0e-5);
    zgradpulse(hsglvl,hsgt);
    obspower(tpwr);
    delay(d1 - mix);
   }
   else
   {
    delay(1.0e-5);
    obspower(tpwr);
    zgradpulse(hsglvl,2.0*hsgt);
    rgpulse(pw, zero, rof1, rof1);
    zgradpulse(hsglvl,hsgt);
    rgpulse(pw, one, rof1, rof1);
    zgradpulse(0.5*hsglvl,hsgt);
    delay(d1 - 2.0*hsgt - 4.0*rof1 - 2.0*pw - 1.0e-5 - POWER_DELAY);
   }


 if (satmode[0] == 'y')
  {
    if ((d1-satdly) > 0.02)
          delay(d1-satdly);
    else
          delay(0.02);
    if (getflag("slpsat"))
      {
          shaped_satpulse("relaxD",satdly,zero);
          if (getflag("prgflg"))
            shaped_purge(v6,zero,v18,v19);
      }
    else
      {
          satpulse(satdly,zero,rof1,rof1);
          if (getflag("prgflg"))
            purge(v6,zero,v18,v19);
      }
  }

 if (getflag("wet"))
  wet4(zero,one);

status(B);
  decpower(pwxlvl);

   if (asap[0] == 'y')
   {
    rgpulse(pw,zero,rof1,rof1);
    delay(taua/2.0 - pw - 2.0*rof1 + pw180/2.0);
          if(adiabatic[A] == 'y')
          {
      rgpulse(2.0*pw,two,rof1,0.0);
      decshaped_pulse(ad180.name,ad180.pw,zero,0.0,rof1);
          }
          else
      simpulse(2.0*pw,2.0*pwx,two,zero,rof1,rof1);
    delay(taua/2.0 - pw - 2.0*rof1 - pw180/2.0);
    simpulse(pw,pwx,zero,v1,rof1,1.0e-6);
   }
   else
   {
    rgpulse(pw,zero,rof1,rof1);
    delay(taua - rof1 - (2*pw/PI));
    decrgpulse(pwx,v1,rof1,1.0e-6);
   }
   delay(gtE + 2*gstab + 2*GRADIENT_DELAY - (2*pwx/PI) - pwx - 1.0e-6 - rof1);
```

```
if(adiabatic[A] == 'y')
  decshaped_pulse(ad180.name,ad180.pw,v4,rof1,1.0e-6);
else
  decrgpulse(2*pwx,v4,rof1,1.0e-6);
delay(gstab - pwx - 1.0e-6);
zgradpulse(gzlvlE,gtE);
delay(gstab - rof1 - pw);
delay(d2/2);
rgpulse(2.0*pw,zero,rof1,rof1);
delay(d2/2);

delay(gstab - rof1 - pw);
zgradpulse(gzlvlE,gtE);
delay(gstab - pwx - rof1);
if(adiabatic[A] == 'y')
  decshaped_pulse(ad180.name,ad180.pw,zero,rof1,1.0e-6);
else
  decrgpulse(2*pwx,zero,rof1,1.0e-6);
delay(gtE + 2*gstab + 2*GRADIENT_DELAY - (2*pwx/PI) - pwx - 1.0e-6 - rof1);
decrgpulse(pwx,t2,rof1,rof1);
if(asap[0] == 'y')
{
  delay(0.25/j1xh + ad180.pw/2.0);
  if(adiabatic[A] == 'y')
        {
    rgpulse(2.0*pw,zero,rof1,0.0);
    decshaped_pulse(ad180.name,ad180.pw,zero,0.0,rof1);
        }
        else
    simpulse(2.0*pw,2.0*pwx,zero,zero,rof1,rof1);
  delay(0.1*gstab - rof1);
  if(EDratio > 4)
  {
    zgradpulse(icosel*gzlvlE/EDratio*4.0,gtE/2.0);
    delay(0.25/j1xh - ad180.pw/2.0 - gtE/2.0 - 0.1*gstab - 2*GRADIENT_DELAY);
  }
  else
  {
    zgradpulse(icosel*gzlvlE/EDratio*2.0,gtE);
    delay(0.25/j1xh - ad180.pw/2.0 - gtE - 0.1*gstab - 2*GRADIENT_DELAY);
  }
}
else
{
  delay(0.1*gstab - rof1);
  if(EDratio > 4)
  {
    zgradpulse(icosel*gzlvlE/EDratio*4.0,gtE/2.0);
    delay(0.5/j1xh - gtE/2.0 - 0.1*gstab - 2*GRADIENT_DELAY);
  }
  else
  {
    zgradpulse(icosel*gzlvlE/EDratio*2.0,gtE);
    delay(0.5/j1xh - gtE - 0.1*gstab - 2*GRADIENT_DELAY);
  }
}
if (lkgate_flg[0] == 'y') lk_sample(); /* turn lock sampling on */
decpower(dpwr);
delay(rof2 - POWER_DELAY);

        obsblank();
        startacq(alfa);

        status(C);
              acquire(npnus[i],1.0/sw);
              rcvroff();
```

```
        status(B);
                decpower(pwxlvl);
                decrgpulse(2.0*pwx,zero,rof1,rof1);
                delay(rtd/2.0 + POWER_DELAY);
                                delay(tauA);
                decrgpulse(2.0*pwx,two,rof1,rof1);
                decpower(dpwr);
                delay(rtd/2.0 - POWER_DELAY);
                obsblank();
                                delay(tauB - 2.0e-6);           //obsblank transition of 2.0e-6
                rcvron();       //this includes rof3
                                delay(tauD - rof3);

/*------------------------------------------------------------------
        Loops for more chunks
------------------------------------------------------------------*/

        for (i = 1; i < 3; i++)
        {

        status(C);
                acquire(npnus[i],1.0/sw);
                rcvroff();

        status(B);
                decpower(pwxlvl);
                decrgpulse(2.0*pwx,zero,rof1,rof1);
                delay(rtd/2.0 + POWER_DELAY);
                                delay(tauA);
                decrgpulse(2.0*pwx,two,rof1,rof1);
                decpower(dpwr);
                delay(rtd/2.0 - POWER_DELAY);
                obsblank();
                                delay(tauB - 2.0e-6);           //obsblank transition of 2.0e-6
                rcvron();       //this includes rof3
                                delay(tauD - rof3);
        }

/*------------------------------------------------------------------
        Acquisition of last chunk
------------------------------------------------------------------*/
        status(C);
                acquire(npnus[3],1.0/sw);
                rcvroff();

        status(B);
                delay(dcor);
                endacq();
}
```

Reference

1. E. Kupce and R. Freeman, *Journal of Magnetic Resonance, Series A*, 1995, **115**, 273-276.
2. H. Geen and R. Freeman, *Journal of Magnetic Resonance (1969)*, 1991, **93**, 93-141.
3. A. J. Shaka, C. J. Lee and A. Pines, *Journal of Magnetic Resonance (1969)*, 1988, **77**, 274-293.
4. S. G. Hyberts, A. G. Milbradt, A. B. Wagner, H. Arthanari and G. Wagner, *J Biomol NMR*, 2012, **52**, 315-327.
5. K. Kazimierczuk and V. Y. Orekhov, *Angewandte Chemie International Edition*, 2011, **50**, 5556-5559.
6. A. Papoulis, *IEEE Transactions on Circuits and Systems*, 1975, **22**, 735-742.
7. M. Mayzel, K. Kazimierczuk and V. Y. Orekhov, *Chemical Communications*, 2014, **50**, 8947-8950.
8. I. E. Ndukwe, A. Shchukina, K. Kazimierczuk, C. Cobas and C. P. Butts, *ChemPhysChem*, 2016, DOI: 10.1002/cphc.201600541.