Electronic Supplementary Material (ESI) for RSC Advances.

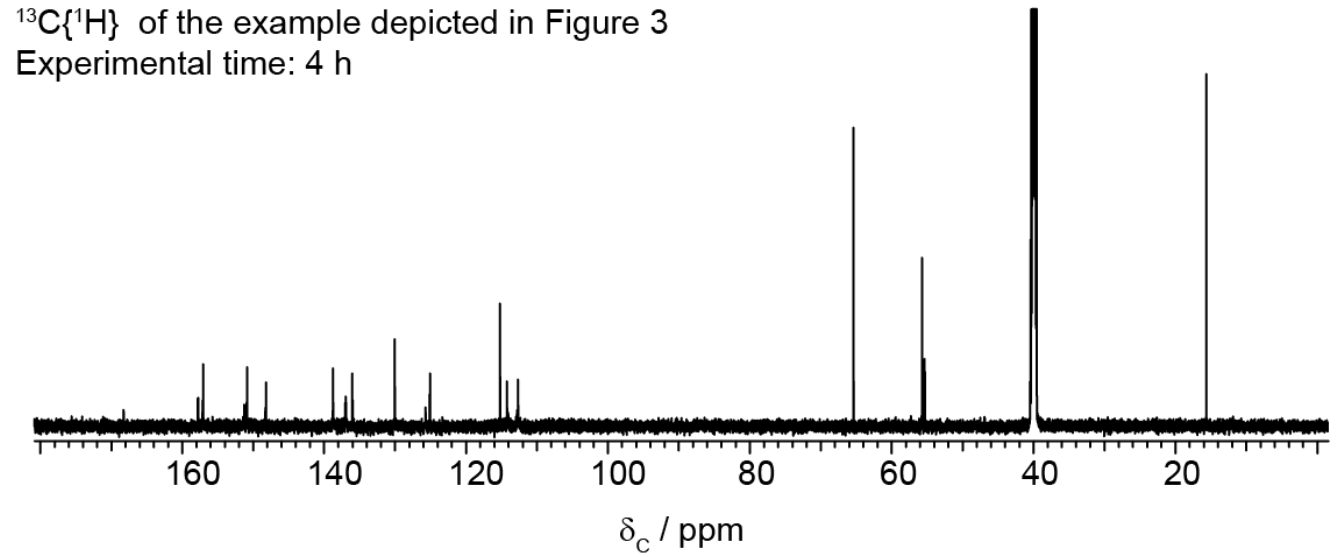
Ultra-high dispersion NMR reveals new levels of detail.

Juan A. Aguilar,[*a] Peter Kiraly,[b] Ralph Adams,[b] Mickaëlle Bonneau,[a] Elizabeth J. Grayson,[a] Mathias Nilsson,[b] Alan M. Kenwright[a] and Gareth A. Morris[b]

Supplementary information content

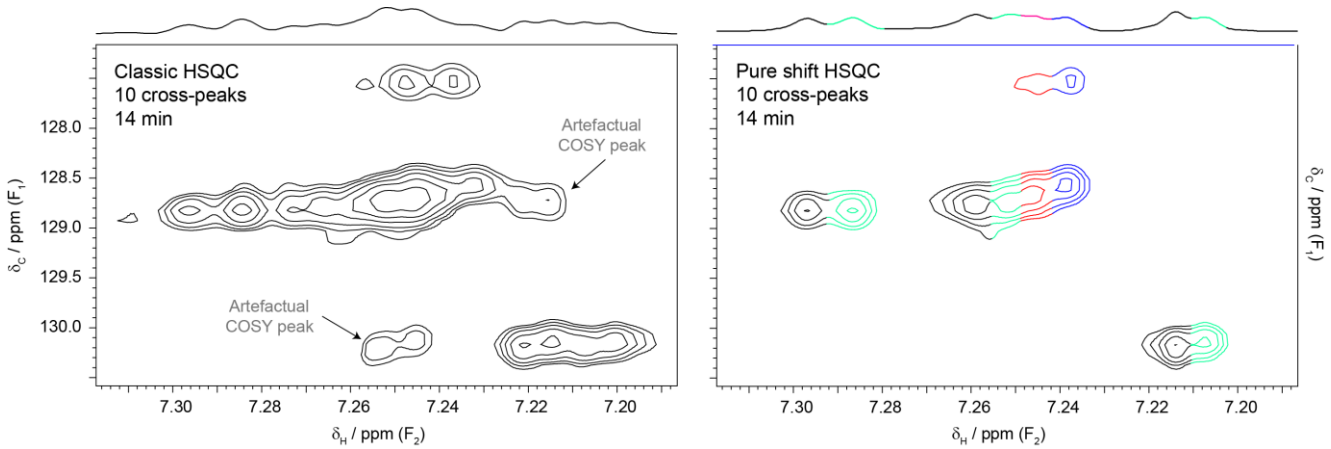
1- The $^{13}C\{^{1}H\}$ acquired to serve as a reference for the example of Figure 3.
2- Pure shift spectra of selected samples submitted to the Durham NMR service.
3- Pulse sequences and macros used in the present publication
- The real-time HSQC sequence for Agilent spectrometers.
- The constant-time COSY pulse sequence for Agilent spectrometers.
- The 1D Zangger-Sterk pulse sequence for Agilent spectrometers.
- The reconstruction macro necessary to assemble the raw data produced by the Zangger-Sterk pulse sequence.

SI-1

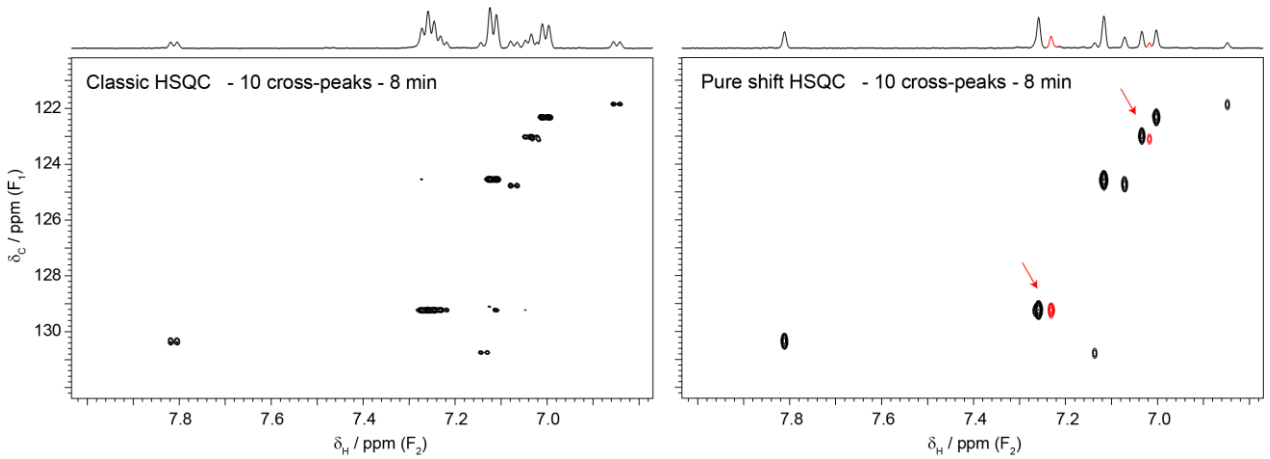

$^{13}C\{^{1}H\}$ of the example depicted in Figure 3
Experimental time: 4 h
160
140
120
100
80
60
40
20
δ_C / ppm

Pure shift spectra of selected samples submitted to the Durham NMR service.

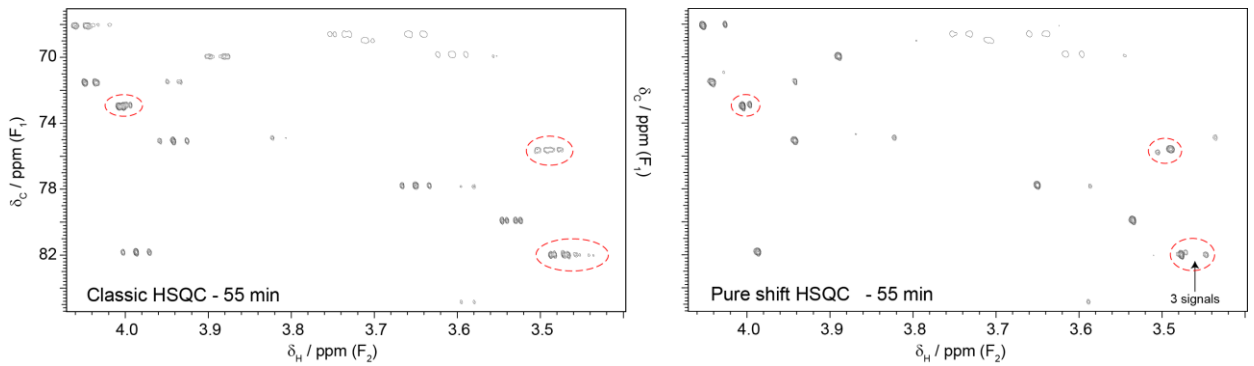
SI-2



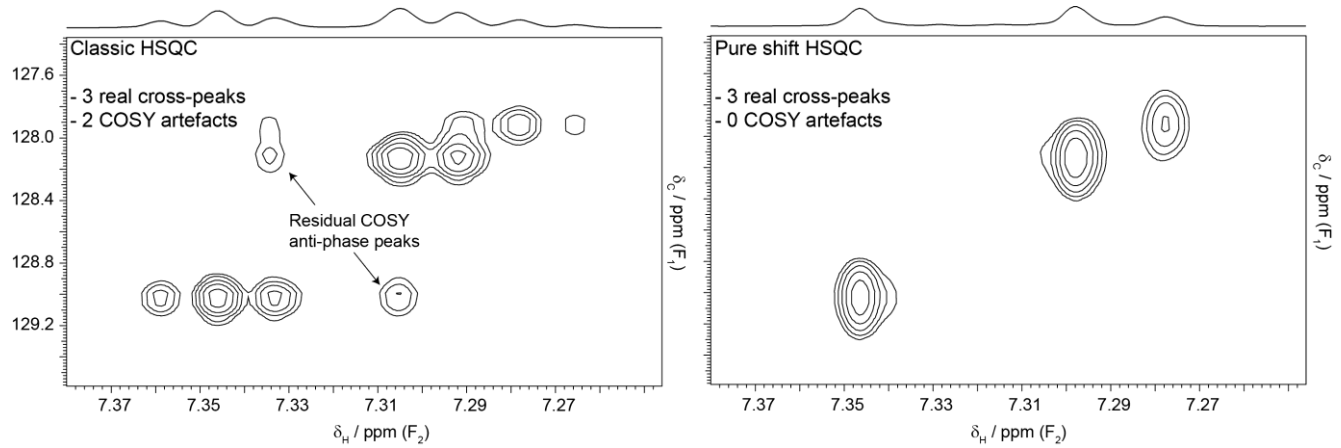
SI-3



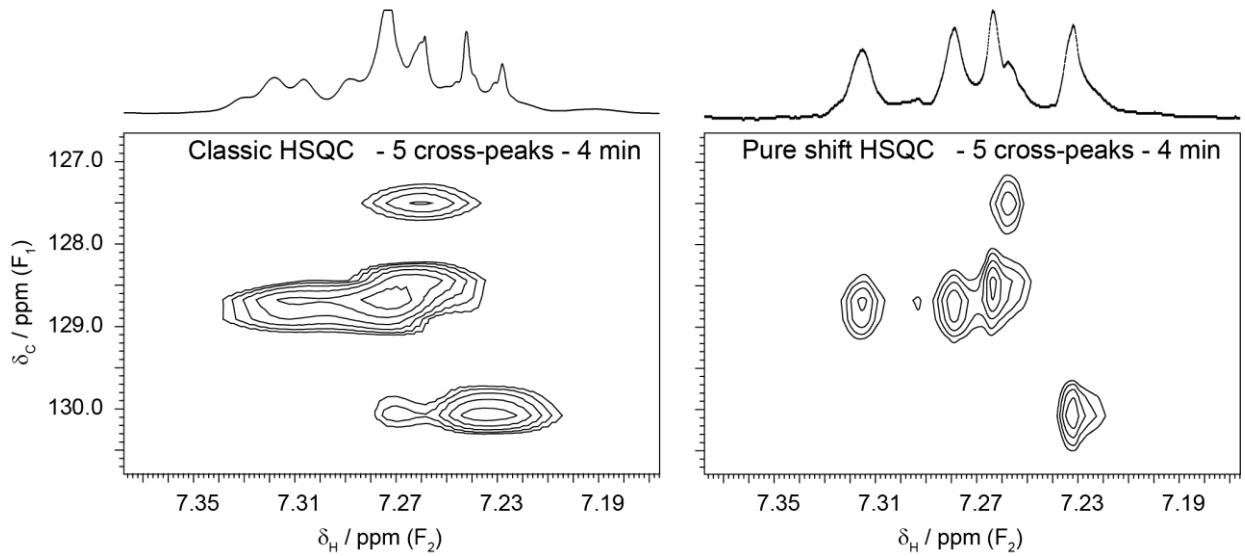
SI-4

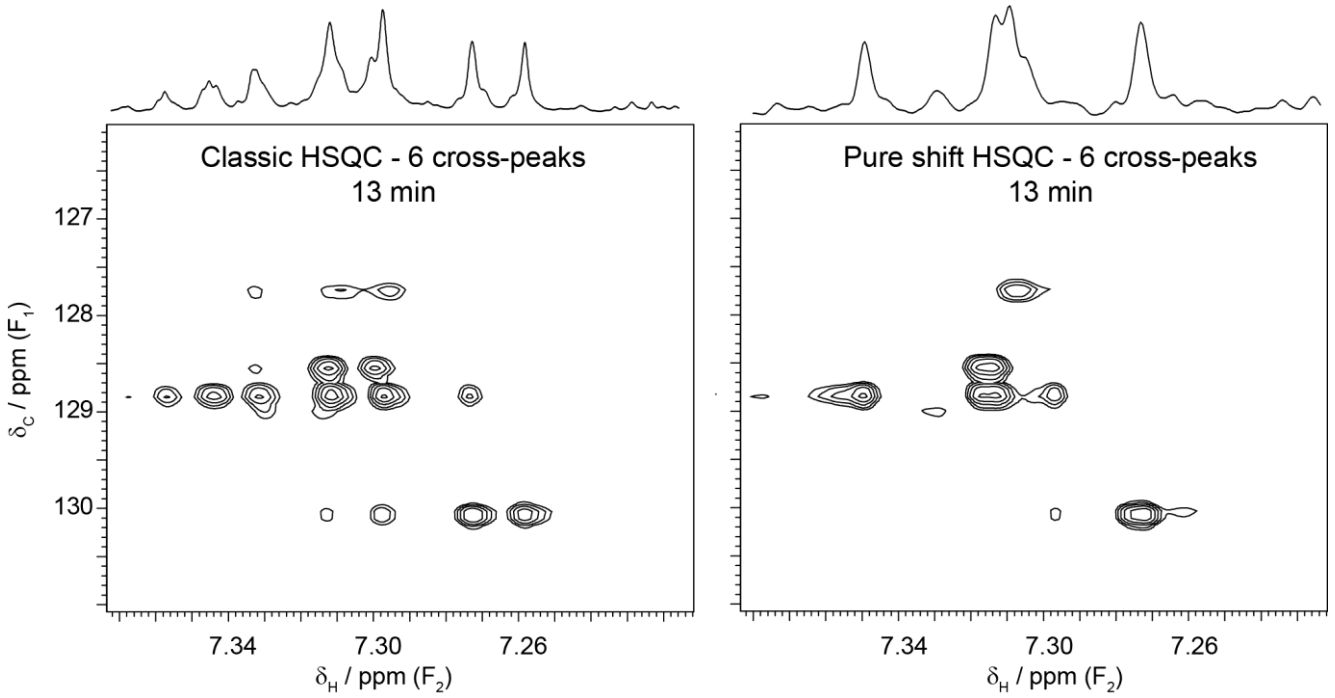


SI-5


Classic HSQC
- 3 real cross-peaks
- 2 COSY artefacts
Residual COSY
anti-phase peaks
127.6
128.0
128.4
128.8
129.2
7.37
7.35
7.33
7.31
7.29
7.27
δH / ppm (F2)
δC / ppm (F1)
Pure shift HSQC
- 3 real cross-peaks
- 0 COSY artefacts
δH / ppm (F2)
δC / ppm (F1)

SI-6


Classic HSQC - 5 cross-peaks - 4 min
127.0
128.0
129.0
130.0
δC / ppm (F1)
7.35
7.31
7.27
7.23
7.19
δH / ppm (F2)
Pure shift HSQC - 5 cross-peaks - 4 min
δH / ppm (F2)


Classic HSQC - 6 cross-peaks
13 min
Pure shift HSQC - 6 cross-peaks
13 min
127
128
129
130
δC / ppm (F1)
7.34
7.30
7.26
δH / ppm (F2)
7.34
7.30
7.26
δH / ppm (F2)

/* The real-time HSQC pulse sequence for Agilent spectrometers STARTS here. Delete this line and save the file as /psglib/rtpsHSQC.c */

```
*------------------------------------------------------------------------
-------------------------------------------------
```

This code is provided as a record of the pulse programmes used to obtain the spectra reported in this publication. There is no warranty (implied or explicit) that it is optimal or bug-free. Anyone using this code does so at their own risk.

```
Developed by the Manchester NMR Group (University of Manchester) and
modified by J. A. Aguilar (Durham University) to use only proton squares
pulses during the real time acquisition when BIPmode='b'

University of Manchester
School of Chemistry
University of Manchester
United Kingdom
May 2013
This is an experimental pulse sequence. Use it accordingly.

2013-07-04 option for adiabatic ('n'/'y') HSQC was added; adjust f1coef
accordingly

Usually 2 transients are enough. Sometimes mirror F1-peaks appear. These
can be easily identified.

------------------------------------------------------------------------
User's Guide for experimental setup:
------------------------------------------------------------------------
1. BIRD = 'n' selects conventional gHSQC
------------------------------------------------------------------------
2. BIRD = 'y' selects real-time pure shift gHSQC (gHSQC-BIRD)
Three options within BIRD are:
BIRDmode ='h' selects hard 13C inversion pulse during BIRD // large off-
resonance effect, not recommended
BIRDmode ='b' selects BIP 13C inversion pulse during BIRD
BIRDmode ='w' selects a pair of wurst adiabatic 13C inversion pulses during
BIRD
For all np should be integer submultiple of npoints
Users control chunking time using npoints so that np/npoints is an integer
Note:
chunk_time=npoints/(2*sw)=at/cycles
cycles=np/npoints, an integer
at=np/(2*sw)=cycles*npoints/(2*sw)=cycles*chunk_time

JAA: mult=0,1,2 As in the regular experiment.
------------------------------------------------------------------------
-------------------------------------------------*/

#include <standard.h>
//#include <chempack.h>
```

```
/*-----------------------------------------
Phase tables for Varian gHSQC
-----------------------------------------*/

static int ph1[4] = {1,1,3,3}, //v1 - proton 90 at the end of first inept
ph2[2] = {0,2}, //v2 - X 90 at the end of first inept
ph3[8] = {0,0,0,0,2,2,2,2}, //v3 - proton 90 in 2nd inept
ph4[16] = {0,0,0,0,0,0,0,0,2,2,2,2,2,2,2,2}, //v4 - X 90 in 2nd inept
ph5[16] = {1,3,3,1,3,1,1,3,3,1,1,3,1,3,3,1}; //oph
/*-----------------------------------------
Phase tables for rtgHSQC-BIRD
-----------------------------------------*/
static int ph11[8] = {1,1,1,1,3,3,3,3}, //v1
ph12[2] = {0,2}, //v2
ph13[16] = {0,0,0,0,0,0,0,0,2,2,2,2,2,2,2,2}, //v3
ph14[32] =
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2}, //v4
ph15[32] =
{1,3,1,3,3,1,3,1,3,1,3,1,1,3,1,3,3,1,3,1,1,3,1,3,1,3,1,3,3,1,3,1}; //oph
static int ph17[4] = {0,0,1,1}, //v7 - 1st 90 of bird and the hard 180
ph18[4] = {1,1,2,2}, //v8 - simpulse 180 of bird
ph19[4] = {2,2,3,3}; //v9 - 2nd 90 of bird

pulsesequence()
{
/*-----------------------------------------------------------
DECLARE AND LOAD VARIABLES
-----------------------------------------------------------*/
//HSQC part

double evolcorr=2.0*pw+4.0e-6,
tau = 1.0/(4.0*(getval("j1xh"))),
taug=2.0*tau,
mult = getval("mult");

int phase1 = (int)(getval("phase")+0.5),
ZZgsign=1.0,
icosel;

//BIRD
double
rof3=getval("rof3"), //delay for receiver off - can be zero if ddrpm='r'
tauA=getval("tauA"), //compensation for tauB and tauC
tauB=getval("tauB"), //effect of rof2
tauC=getval("tauC"), //effect of alfa
tBal=getval("tBal"), //supports inova console if ~1/(fb*1.3)
pwr_XBIP = getval("pwr_XBIP"),
pwr_HBIP = getval("pwr_HBIP"),
pw_XBIP = getval("pw_XBIP"),
pw_HBIP = getval("pw_HBIP"),
npoints=getval("npoints"), // npoints should be an integer multiple of np
cycles=np/npoints;
```

```
cycles = (double)((int)((cycles)));
initval(cycles,v20);

char shp_HBIP[MAXSTR],
shp_XBIP[MAXSTR];
getstr("shp_HBIP",shp_HBIP);
getstr("shp_XBIP",shp_XBIP);

//extensions for AD
double pwx180 = getval("pwx180"),
pwxlvl180 = getval("pwxlvl180"),
pwx180r = getval("pwx180r"),
pwxlvl180r = getval("pwxlvl180r");

char pwx180ad[MAXSTR],
pwx180adR[MAXSTR],
pwx180ref[MAXSTR];
getstr("pwx180ad", pwx180ad);
getstr("pwx180adR", pwx180adR);
getstr("pwx180ref", pwx180ref);

//gradients
double gtE = getval("gtE"), //HSQC encoding
gzlvlE = getval("gzlvlE"),
gstab = getval("gstab"),
gtD = getval("gtD"), //HSQC decoding
gzlvlD = getval("gzlvlD"),
hsglvl = getval("hsglvl"),
hsgt = getval("hsgt"),
hsgstab = getval("hsgstab");

char BIRD[MAXSTR], // Flag to choose gHSQC/rtgHSQC-BIRD ('n'/'y')
BIRDmode[MAXSTR], //Flag to choose hard/bip/wurst2i ('h'/'b'/'w')13C
inversion pulse within BIRD
adiabatic[MAXSTR]; //Flag to use adiabatic refocusing on X-channel
('n'/'y')
getstr("BIRD",BIRD);
getstr("BIRDmode",BIRDmode);
getstr("adiabatic",adiabatic);

char sspul[MAXSTR],
PFGflg[MAXSTR];
getstr("sspul",sspul);
getstr("PFGflg",PFGflg);

//evolcorr and mult declarations
if (adiabatic[0]=='n')
{
evolcorr = 2*pw+4.0e-6;
if (mult > 0.5)
taug = 2*tau;
else
taug = gtE + gstab + 2*GRADIENT_DELAY;
```

```
ZZgsign=-1;
if (mult == 2) ZZgsign=1;
icosel = 1;
}
//AD
if (adiabatic[0]=='y')
{
evolcorr = (4*pwx/PI)+2*pw+8.0e-6;
if (mult > 0.5)
taug = 2*tau; // + getval("tauC");
else
taug = gtE + gstab + 2.0 * GRADIENT_DELAY;
ZZgsign=-1;
if (mult == 2) ZZgsign=1;
icosel = 1;
}


//setup the phase cycle
assign(ct,v10);


if (BIRD[0]=='n')
{
//gHSQC phases
settable(t1,4,ph1);
settable(t2,2,ph2);
settable(t3,8,ph3);
settable(t4,16,ph4);
settable(t5,16,ph5);
}
else
{
//rtgHSQC-BIRD phases
settable(t1,8,ph11);
settable(t2,2,ph12);
settable(t3,16,ph13);
settable(t4,32,ph14);
settable(t5,32,ph15);
settable(t7,4,ph17);
settable(t8,4,ph18);
settable(t9,4,ph19);
getelem(t7, v10, v7);
getelem(t8, v10, v8);
getelem(t9, v10, v9);
}

getelem(t1, v10, v1);
getelem(t2, v10, v2);
getelem(t3, v10, v3);
getelem(t4, v10, v4);
getelem(t5, v10, oph);
```

```
initval(2.0*(double)(((int)(d2*getval("sw1")+0.5)%2)),v5);
if ((phase1 == 2) || (phase1 == 5))
icosel = -1;

add(v2,v5,v2);
add(oph,v5,oph);

/* BEGIN PULSE SEQUENCE */
status(A);

if (sspul[A] == 'y')
{
if (PFGflg[A] == 'y')
{
obspower(tpwr);
delay(5.0e-5);
zgradpulse(hsglvl,hsgt);
rgpulse(pw,zero,rof1,rof1);
zgradpulse(hsglvl,hsgt);
}
else
{
obspower(tpwr-12);
delay(5.0e-5);
rgpulse(500*pw,zero,rof1,rof1);
rgpulse(500*pw,one,rof1,rof1);
}
}

obspower(tpwr);
decpower(pwxlvl);
txphase(zero);
decphase(zero);
obsoffset(tof);
decoffset(dof);

delay(d1);
delay(5.0e-5);

status(B);

/****** null flag starts here *****/

if (getflag("nullflg"))
{
rgpulse(0.5*pw,zero,rof1,rof1);
delay(2.0*tau);
if (adiabatic[0]=='y')
{
decpower(pwxlvl180);
decshaped_pulse(pwx180ad, pwx180, zero, rof1, rof1);
rgpulse(2.0*pw,zero,rof1,rof1);
}
```

```
else { simpulse(2.0*pw,2.0*pwx,zero,zero,rof1,rof1); }
txphase(two);
delay(2.0*tau);
if (adiabatic[0]=='y')
{
decshaped_pulse(pwx180adR, pwx180, zero, rof1, rof1);
decpower(pwxlvl);
}
rgpulse(1.5*pw,two,rof1,rof1);
txphase(zero);
zgradpulse(hsglvl,hsgt);
delay(hsgstab);
}

/****************************gHSQC or gHSQC part of pure shift starts here
************************/

rgpulse(pw,zero,0.0,0.0);
delay(tau);
if (adiabatic[0]=='y')
{
decpower(pwxlvl180);
decshaped_pulse(pwx180ad, pwx180, zero, rof1, rof1);
rgpulse(2.0*pw,zero,rof1,rof1);
}
else { simpulse(2.0*pw,2.0*pwx,zero,zero,rof1,rof1); }
txphase(v1);
delay(tau);
if (adiabatic[0]=='y')
{
decshaped_pulse(pwx180adR, pwx180, zero, rof1, rof1);
decpower(pwxlvl);
}
rgpulse(pw,v1,rof1,rof1);

zgradpulse(hsglvl,2.0*hsgt);
decphase(v2);
delay(hsgstab);

decrgpulse(pwx, v2, rof1, 2.0e-6);
txphase(zero);
decphase(zero);

delay(d2/2.0); // First half of t1 evolution
rgpulse(2.0*pw,zero,2.0e-6,2.0e-6);
delay(d2/2.0); // Second half of t1 evolution

/**/

if (adiabatic[0]=='y')
{
delay(taug - POWER_DELAY);
if (mult > 0.5)
```

```
{
decpower(pwxlvl180r);
decshaped_pulse(pwx180ref, pwx180r, zero, rof1, rof1);
rgpulse(mult * pw, zero, rof1, rof1);
delay(taug - mult * pw - 2.0*rof1 + POWER_DELAY - gtE - gstab - 2.0 *
GRADIENT_DELAY+evolcorr);
zgradpulse(gzlvlE,gtE);
delay(gstab);
decshaped_pulse(pwx180ref, pwx180r, zero, rof1, rof1);
}
else
{
decpower(pwxlvl180);
decshaped_pulse(pwx180ad, pwx180, zero, rof1, rof1);
delay(taug + POWER_DELAY - gtE - gstab - 2.0 * GRADIENT_DELAY+evolcorr);
zgradpulse(gzlvlE,gtE);
delay(gstab);
decshaped_pulse(pwx180ad, pwx180, zero, rof1, rof1);
}
decpower(pwxlvl);
}

if (adiabatic[0]=='n')
{
zgradpulse(gzlvlE,gtE);
delay(taug - gtE - 2.0*GRADIENT_DELAY);
simpulse(mult*pw,2.0*pwx,zero,zero,rof1,rof1);
delay(taug + evolcorr);
}

decrgpulse(pwx,v4,2.0e-6,rof1);
zgradpulse(ZZgsign*0.6*hsglvl,1.2*hsgt);
txphase(v3);
delay(hsgstab);
rgpulse(pw,v3,rof1,rof1);

if (adiabatic[0]=='y')
{
decpower(pwxlvl180);
decshaped_pulse(pwx180adR, pwx180, zero, rof1, rof1);
//decpower(dpwr);
}
delay(tau - (2.0*pw/PI) - 2.0*rof1);

if (adiabatic[0]=='y')
{
rgpulse(2.0*pw,zero,rof1, rof1);
decpower(pwxlvl180);
decshaped_pulse(pwx180ad, pwx180, zero, rof1, rof1);
//decpower(dpwr);
}
else { simpulse(2.0*pw,2.0*pwx,zero,zero,rof1, rof1); }
```

```
zgradpulse(icosel*gzlvlD,gtD);
decpower(dpwr);
delay(tau - gtD - 2.0*GRADIENT_DELAY - POWER_DELAY);

/*******************************gHSQC part stops and BIRD Acquisition
starts here************************/

delay(tBal);
//filter delay (Hoult) for inova; adjust tBal manually for the same effect
//delay(1.0/(getval("fb")*1.3))
if (BIRD[0]=='y')
{
setacqmode(WACQ|NZ); //use this line only for vnmrs console; comment this
out in inova


obsblank();
delay(rof2);
startacq(alfa);
}
/*------------------------------------------------------------------------
-------------------------------------------------------
Observe the 1st half chunk
------------------------------------------------------------------------
-------------------------------------------------------*/
if (BIRD[0]=='y')
{
status(C);
acquire(npoints/2.0,1.0/sw);
rcvroff();
status(B);
obspower(tpwr);
txphase(v7);

/*------------------------------------------------------------------------
Using hard 13C inversion pulse in BIRD
------------------------------------------------------------------------
*/
if (BIRDmode[0]== 'h')
{
rgpulse(pw,v7,rof1,rof1);
decpower(pwxlvl);
delay(2.0*tau);
simpulse(2.0*pw,2.0*pwx,v8,v8,rof1,rof1);
decpower(dpwr);
delay(2.0*tau);
rgpulse(pw,v9,rof1,rof1);
}

/*------------------------------------------------------------------------
--
Using BIP 13C inversion pulse in BIRD
```

```
------------------------------------------------------------------------
-*/
if (BIRDmode[0]== 'b')
{
rgpulse(pw,v7,rof1,rof1);
obspower(tpwr);
if (pwr_XBIP!=pwxlvl) decpower(pwr_XBIP); else decpower(pwxlvl);
delay(2.0*tau);
simshaped_pulse("",shp_XBIP,pw*2.0,pw_XBIP,v8,v8,rof1,rof1);
decpower(dpwr);
delay(2.0*tau);
rgpulse(pw,v9,rof1,rof1);
}

/*------------------------------------------------------------------------
-
Using a pair of wurst2i adiabatic 13C inversion pulses in BIRD
------------------------------------------------------------------------
-*/
if (BIRDmode[0]== 'w')
{
rgpulse(pw,v7,rof1,rof1);
if (pwr_HBIP!=tpwr) obspower(pwr_HBIP);
if (pwxlvl180!=pwxlvl) decpower(pwxlvl180); else decpower(pwxlvl);
txphase(v8); decphase(v8);
delay(2.0*tau);
decshaped_pulse(pwx180ad, pwx180, v8, rof1, rof1);
shaped_pulse(shp_HBIP,pw_HBIP,v8,rof1,rof1);
if (pwr_HBIP!=tpwr) obspower(tpwr);
txphase(v9);
delay(2.0*tau);
decshaped_pulse(pwx180adR, pwx180, v8, rof1, rof1);
decpower(dpwr);
rgpulse(pw,v9,rof1,rof1);
}
txphase(v7);
delay(tauA);
rgpulse(pw*2.0,v7,rof1,rof1); // hard 180 degree refocusing pulse
obsblank();
delay(tauB);
rcvron(); //this includes rof3
delay(tauC);

decr(v20);

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

starthardloop(v20);
status(C);
```

```
acquire(npoints,1.0/sw);
rcvroff();

status(B);
obspower(tpwr);
txphase(v7);

/*------------------------------------------------------------------------
Using hard 13C inversion pulse in BIRD
------------------------------------------------------------------------
*/
if (BIRDmode[0]== 'h')
{
rgpulse(pw,v7,rof1,rof1);
decpower(pwxlvl);
delay(2.0*tau);
simpulse(2.0*pw,2.0*pwx,v8,v8,rof1,rof1);
decpower(dpwr);
delay(2.0*tau);
rgpulse(pw,v9,rof1,rof1);
}

/*------------------------------------------------------------------------
--
Using BIP 13C inversion pulse in BIRD
------------------------------------------------------------------------
-*/
if (BIRDmode[0]== 'b')
{
rgpulse(pw,v7,rof1,rof1);
obspower(tpwr);
if (pwr_XBIP!=pwxlvl) decpower(pwr_XBIP); else decpower(pwxlvl);
delay(2.0*tau);
simshaped_pulse("",shp_XBIP,pw*2.0,pw_XBIP,v8,v8,rof1,rof1);

decpower(dpwr);
delay(2.0*tau);
rgpulse(pw,v9,rof1,rof1);
}

/*------------------------------------------------------------------------
-
Using a pair of wurst2i adiabatic 13C inversion pulses in BIRD
------------------------------------------------------------------------
-*/
if (BIRDmode[0]== 'w')
{
rgpulse(pw,v7,rof1,rof1);
if (pwr_HBIP!=tpwr) obspower(pwr_HBIP);
if (pwxlvl180!=pwxlvl) decpower(pwxlvl180); else decpower(pwxlvl);
txphase(v8); decphase(v8);
delay(2.0*tau);
decshaped_pulse(pwx180ad, pwx180, v8, rof1, rof1);
```

```
shaped_pulse(shp_HBIP,pw_HBIP,v8,rof1,rof1);
if (pwr_HBIP!=tpwr) obspower(tpwr);
txphase(v9);
delay(2.0*tau);
decshaped_pulse(pwx180adR, pwx180, v8, rof1, rof1);
decpower(dpwr);
rgpulse(pw,v9,rof1,rof1);
}

txphase(v7);
delay(tauA);
rgpulse(pw*2.0,v7,rof1,rof1); // hard 180 degree refocusing pulse
obsblank();
delay(tauB);
rcvron(); //this includes rof3
delay(tauC);

endhardloop();

/*------------------------------------------------------------------------
-------------------------------------------------
Acquisition of last half chunk
------------------------------------------------------------------------
-------------------------------------------------*/
status(C);
acquire(npoints/2.0,1.0/sw);
rcvroff();
endacq();
incr(v20);
}

/****** BIRD ends here for all *****/


/************************ ACQ for conventional gHSQC
*********************************/
else
{
status(C);
}

}
/********************** PULSE SEQUENCE ENDS
HERE*************************************/

/* The rtps-HSQC pulse sequence ENDS here. Delete this line and save the file as
/psglib/rtpsHSQC.c */
```

/* The constant-time COSY pulse sequence for Agilent spectrometers STARTS here. Delete this line and save the file as psglib/COSY_CT.c */

/* This code is provided as a record of the pulse programmes used to obtain the spectra reported in this publication. There is no warranty (implied or explicit) that it is optimal or bug-free. Anyone using this code does so at their own risk. */

```
#ifndef LINT
static char SCCSid[] = "@(#)";
#endif


/* Juan A. Aguilar. Durham. 11-07-2013.
Magnitude mode constant-time COSY-60
Usually provides good results when the bandwidth is reduced (less than 5
ppm). Typically a single transient and 512 increments is enough. Make sure
that the resolution along F1 sufficient to be able to tell the difference
between a singlet and a multiplet and that the acquisition time is also
long enough to be able to see the multiplicity. Under these conditions,
good results can be produced in around 20 min. Sensitivity can be improved
by degassing samples, but this if often not necessary.
This is an experimental pulse sequence. Use it accordingly
Paramters:

gzlvl1 : Coherence selection gradient level
gt1 : Gradient time
gstab : Recovery delay
pw : 90 degree pulse length at tpwr
d1 : relaxation delay
cti : Constant time period. Automatically
calculated
*/

#include <standard.h>

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

pulsesequence()
{
double gzlvl1 = getval("gzlvl1"),
gzlvl2 = getval("gzlvl2"),
gt1 = getval("gt1"),
gstab = getval("gstab"),
gstab2 = getval("gstab2"),
cti = getval("cti");

char sspul[MAXSTR];
getstr("sspul",sspul);
```

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

getelem(t1,ct,v1);
getelem(t2,ct,v2);
getelem(t3,ct,oph);

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

cti=2.0*(ni*0.5/sw1+gt1*2.0+gstab*2.0+0.0001);

status(A);
delay(d1);

obspower(tpwr);
obsoffset(tof);
delay(rof1);

status(B);

rgpulse(pw, v1, rof1, rof1);
delay(cti*0.5 - d2*0.5+gstab+gt1);
zgradpulse(gzlvl1,gt1);
delay(gstab);
rgpulse(pw*2.0, zero, rof1, rof1);

zgradpulse(gzlvl1,gt1);
delay(gstab);
delay(cti*0.5 + d2*0.5);
zgradpulse(gzlvl1*0.3,gt1);
delay(gstab);
rgpulse(pw*0.666667, v2, rof1, rof2);
zgradpulse(gzlvl1*0.3,gt1);
delay(gstab);

status(C);
}
```

/* The constant-time COSY pulse sequence for Agilent spectrometers ENDS here. Delete this line and save the file as psglib/COSY_CT.c */

/* The Zangger-Sterk pulse sequence for Agilent spectrometers STARTS here. Delete this line and save the file as /psglib/pureshiftZS.c */

/* This code is provided as a record of the pulse programmes used to obtain the spectra reported in this publication. There is no warranty (implied or explicit). Anyone using this code does so at their own risk. */

```
#ifndef LINT
static char SCCSid[] = "@(#)GMZSr.c 19.1 01/13/06 Copyright (c) 1991-1996
Varian Assoc.,Inc. All Rights Reserved";
#endif
/*
Usually good results can be produced under 10 min using two transients and
32 to 48 increments provided that the bandwidth to be analysed is
relatively reduced (a few ppm). */

/* GMZSnew - simple Zangger-Sterk pure shift with hard 180 before soft */


#include <standard.h>
#include <Pbox_psg.h>

pulsesequence()
{
double p1lvl,
droppts = getval("droppts"), /* number of dummy points to acquire */
gzlvl1=getval("gzlvl1"), /* CTP selection gradient */
gzlvl2=getval("gzlvl2"), /* slice select gradient */
gzlvl_ss=getval("gzlvl_ss"), /* steady-state crusher gradient */
gtss=getval("gtss"), /* steady-state gradient pulse width */
gt1=getval("gt1"), /* CTP gradient pulse width */
gstab=getval("gstab"), /* gradient stabilisation delay */
selpw=getval("selpw"), /* pulse length for the soft 180 */
selpwr=getval("selpwr"); /* power level for the soft 180 */

char sspul[MAXSTR],
selshape[MAXSTR]; /* pulse file for the soft 180 */

getstr("sspul",sspul);
getstr("selshape",selshape);

/*Check that gstab > 1/sw */
if (gstab<(droppts)/sw)
{
abort_message("gstab should be greater than droppts/sw\n");
}

/*Check that sw1 is an integer submultiple of sw */
if (fabs((sw/sw1)-(double)((int)((sw/sw1)+0.5)))> 0.01)
{
text_message("WARNING: sw1 should be an integer submultiple of sw\n");
}
```

```
/*Check that power deposition in gradient coil is not excessive*/
if ( (gzlvl2*gzlvl2*selpw/(gradstepsz*gradstepsz))> 0.01) /* maximum 1% of
full dissipation */
{
abort_message("Slice select gradient gzlvl2 is dangerously high\n");
}



/* LOAD AND INITIALIZE VARIABLES */
getstr("sspul", sspul);

/* CHECK CONDITIONS */

/* CALCULATE PHASES */
mod4(ct,v1);
mod2(ct,v3);
sub(v3,one,v2);
add(v3,one,v4);
add(v4,one,v5);
mod4(v2,v2);
mod4(v3,v3);
mod4(v4,v4);
hlv(ct,v6);
hlv(v6,v7);
hlv(v7,v7);
mod4(v6,v6);
mod4(v7,v7);
add(v6,v7,oph);
dbl(oph,oph);
add(oph,v1,oph);
mod4(oph,oph); /* v1 + 2(v6+v7) */

/* BEGIN ACTUAL PULSE SEQUENCE CODE */
status(A);
if (sspul[0] == 'y')
{
zgradpulse(gzlvl_ss,gtss);
obspower(tpwr);
rgpulse(1000*1e-6, zero, rof1, 0.0e-6);
rgpulse(1000*1e-6, one, 0.0e-6, rof1);
zgradpulse(gzlvl_ss,gtss*0.6);
}
obspower(tpwr);
hsdelay(d1);
status(B);

rgpulse(pw,v1,rof1,rof2);

delay(d2/2.0);
delay((0.25/sw1)-gt1-gstab);
zgradpulse(gzlvl1*0.5,gt1);
delay(gstab);
```

```
rgpulse(pw*2,v7,rof1,rof2); /* Hard 180*/
delay(0.25/sw1);
delay(gstab);
zgradpulse(-gzlvl1*0.5,gt1); /* second CTP pulse*/
delay(gstab);
obspower(selpwr); /* POWER CHANGE (SOFT) */
rgradient('z',gzlvl2); /* SLICE selection*/
shaped_pulse(selshape,selpw, v6, rof1, rof2); /* SLICE selection - Soft 180
*/
rgradient('z',0.0); /* SLICE selection*/
delay(gstab);
zgradpulse(-gzlvl1,gt1); /* third CTP pulse*/
rcvron();
obsblank();
delay(gstab-droppts/sw);
obspower(tpwr); /* Power level change (HARD) */
delay(d2/2.0);

/* detection */
status(C);
}
```

/* The Zangger-Sterk pulse sequence for Agilent spectrometers END here. Delete this line and save the file as /psglib/pureshiftZS.c */

/* The assembler macro for Zangger-Sterk pulse sequences for Agilent spectrometers STARTS here. Delete this line and save the file as /maclib/pureshift_proc */

```
if (($#>0)) then
write('error','Usage: pureshift_proc; takes no arguments')
abort
endif

jexp:$exp,$expname

cptmp('pureshift')
$nfid=ni
if (lsfid>0) then
$droppts=lsfid+1
else
$droppts=1
ENDIF
exists('droppts','parameter'):$ex
IF $ex>0 then
$droppts=droppts
ENDIF
exists('sw1','parameter'):$ex
IF $ex<1 then "make sure 2D parameters available"
par2d
ENDIF
exists('nchunk','parameter'):$ex
IF $ex>0 then "backward compatibility"
sw1=sw*2/nchunk
groupcopy('current','processed','acquisition')
ENDIF

$npoint=trunc((sw/sw1)+0.5)
$chunk1=$npoint/2
exists('chunk1','parameter'):$ex
IF $ex>0 then
$chunk1=chunk1
ENDIF
if $chunk1=0 then
$chunk1=$npoint
endif
$tmpfile=userdir+'/'+$expname+'/homodec_writefid'
$tmpfile2=userdir+'/'+$expname+'/homodec_fid'
beepoff

$imag=0.0
$real=0.0

exists($tmpfile,'file'):$ex1
IF $ex1>0 then
shell('rm',$tmpfile)
ENDIF

$i=1
REPEAT
writefid($tmpfile,$i)
lookup('file',$tmpfile)
$k=1
repeat
lookup('read'):$temp "read dummy points"
lookup('read'):$temp "read dummy points"
$k=$k+1
until $k>$droppts

$j=1
IF $i<2 THEN
REPEAT
lookup('read'):$imag[$j]
lookup('read'):$real[$j]
$j=$j+1
UNTIL ($j>($chunk1))
ELSE
```

```
  REPEAT
  lookup('read'):$imag[($i-2)*$npoint+$j+$chunk1]
  lookup('read'):$real[($i-2)*$npoint+$j+$chunk1]
  $j=$j+1
 5 UNTIL ($j>($npoint))
  ENDIF
  exists($tmpfile,'file'):$ex1
  IF $ex1>0 then
  shell('rm',$tmpfile)
10 ENDIF
  $i=$i+1
  UNTIL ($i>$nfid)


  exists($tmpfile2,'file'):$ex1
15 IF $ex1>0 then
  shell('rm',$tmpfile2)
  ENDIF


  $i=1
20 REPEAT
  write('file',$tmpfile2,'%d %d',$imag[$i],$real[$i])
  $i=$i+1
  UNTIL ($i>(($nfid-1)*$npoint+$chunk1))


25 rm(curexp+'/acqfil/fid')
  shell('sleep 1')
  makefid($tmpfile2)


  setvalue('np',2.0*($npoint*($nfid-1)+$chunk1))
30 setvalue('fn',np)
  setvalue('at', 0.5*np/sw)
  groupcopy('current','processed','acquisition')
  exists($tmpfile2,'file'):$ex1
  IF $ex1>0 then
35 shell('rm',$tmpfile2)
  ENDIF


  exists('nchunk','parameter'):$ex
  IF $ex>0 then "backward compatibility"
40 destroy('nchunk','current')
  destroy('nchunk','processed')
  ENDIF
  lb='n' gf=at/2 lsfid=0
  fn=4*np ni=0
45 groupcopy('current','processed','acquisition')
  wft aph full vsadj
```

/* The assembler macro ENDS here. Delete this line */

50

55