Electronic Supplementary Information

Robust single-scan ultraselective NMR

Margherita Bazzoni,[a] Armand Régheasse,[a] Patrick Giraudeau,[a] Aurélie Bernard,[a] Ralph W. Adams,[b] Gareth A. Morris,[b] Mathias Nilsson,[b] Peter Kiraly,[*c] Jean-Nicolas Dumez[a, †]

[a] Nantes Université, CNRS, CEISAM, France

[b] Department of Chemistry, University of Manchester, Oxford Road, Manchester, M13 9PL

[c] JEOL UK Ltd., Bankside, Long Hanborough, OX29 8LJ, UK

† Jean-Nicolas Dumez made major contributions to this work prior to his passing.

*Email: peter.kiraly@jeoluk.com

Contents

1. Experimental section

All pulse programs and original data files are available from DOI: 10.48420/31502338

All chemicals were obtained commercially and used without further purification.

Sample 1: 25 mM solution of cyclosporin in C_6D_6.

NMR experiments on the cyclosporin sample were carried out on a Bruker (Avance III) spectrometer working at a ^{1}H Larmor frequency of 500.13 MHz and equipped with a triple-axis gradient broadband inverse-detection probe. Maximum gradients along the X, Y, and Z axes were 47.0, 48.2, and 63.0 G cm^{-1}, respectively. Data were collected using TopSpin version 3.6.2, and analysed using MestReNova. Comparisons of SNR were performed using JASON version 6.1, without apodization and with noise estimated from the 10 to 11 ppm range of each spectrum.

The JND-GEMSTONE experiments were performed using a relaxation delay of 5 s, 2 scans, and 1 dummy scan (unless specified otherwise). WURST adiabatic pulses of 37.5 ms (when total encoding time = 300 ms) and a bandwidth of 5 kHz were used (unless specified otherwise). An rSNOB shaped selective pulse of 16.78 ms duration was used as the selective 180^o refocusing pulse. The encoding gradients (G_e) were along the Z PFG axis, with a strength of 0.56 G cm^{-1} (to match the 5 kHz bandwidth of the chirp pulse), the coherence selection gradient pulses around the selective refocusing pulse were along the Y PFG axis with a 15.9 G cm^{-1} strength. Coherence selection gradient pulses around the even-numbered adiabatic pulses were: X, 6.9 G cm^{-1}; X, 9.4 G cm^{-1}; Y, 7.9 G cm^{-1}; Y, 10.6 G cm^{-1}.

GEMSTONES experiments (4 blocks) were performed using a relaxation delay of 5 s, 2 scans, and 1 dummy scan (unless specified otherwise). WURST adiabatic pulses of 37.5 ms and a

bandwidth of 15 kHz were used (unless specified otherwise). An rSNOB shaped selective pulse of 16.78 ms duration was used as the selective 180^{o} refocusing pulse. The encoding gradients were along the Z PFG axis, with a 1.7 G cm^{-1} strength, the coherence selection gradient pulses were along the X and Y PFG axes. The coherence selection gradient pulses around the refocusing pulse for the 4 blocks were (axis, strength): X, 8.9 G cm^{-1}; Y, 11.1 G cm^{-1}; X, 9.9 G cm^{-1}; X, 8.0 G cm^{-1}. Coherence selection gradient pulses around the second adiabatic pulse of each block were for the 4 blocks (axis, strength): Y, 8.1 G cm^{-1}; X, 11.7 G cm^{-1}; X, 12.7 G cm^{-1}; Y, 9.1 G cm^{-1}.

JND-GEMSTONE(S) TOCSY: The experiments were acquired using a relaxation delay of 5 s, 8 scans, and 2 dummy scans. For TOCSY experiments a mixing time of 60 ms was used, unless differently specified. JND-GEMSTONE TOCSY were based on the block described above, and used a total of 8 adiabatic pulses. GEMSTONES TOCSY experiments were based on the published six-block version of the sequence and used a total of 12 adiabatic pulses. In both case the total encoding time was 300 ms.

Sample 2: 25 mM solution of 17β-estradiol in dmso-d_6. This sample was not freshly prepared and was not sealed, therefore it showed a significant H_2O signal.

NMR experiments on the estradiol sample were carried out using a 400 JEOL ECZL spectrometer equipped with a Royal HFX triple resonance probe and z-only PFG system capable of a maximum gradient of 30 G cm^{-1}. The acquisition software was Delta version 6.4, data processing was carried out using JASON version 5.

The JND-GEMSTONE module for COSY, TOCSY, NOESY and ROESY experiments was set up with a selectivity of 6.25 Hz (corresponding to 160 ms total encoding time provided by

the 8 chirp pulses, each of 20 ms). The chirp bandwidth was set to 10 kHz, matching with an encoding gradient (G_e) strength of 1.2 G/cm. A 23.7 ms (useful bandwidth 80 Hz) rSNOB pulse was used for selective refocusing, and the same for the classic selective experiments. All coherence selection gradient pulses had to be applied along the z-PFG axis, using the following amplitudes: 9.4, 5.9, 11.2, and 7.9 G/cm, and inverting the sign after the refocusing rSNOB pulse. Spectra were averaged using alternated signs of gradient pulses. All gradient recovery delays were set to 1 ms, except after the ZQS element where a longer 3 ms was used. Correlation experiments implemented with GEMSTONE included CLIP COSY, TOCSY with standard ZQS elements before (50 ms chirp pulse covering 25 kHz) and after (30 ms chirp pulse covering 25 kHz) and a DIPSI2 mixing time of 80.1 ms, and easyROESY with a mixing time of 200 ms.

2. The Jean-Nicolas Dumez experiment

2.1 Selectivity

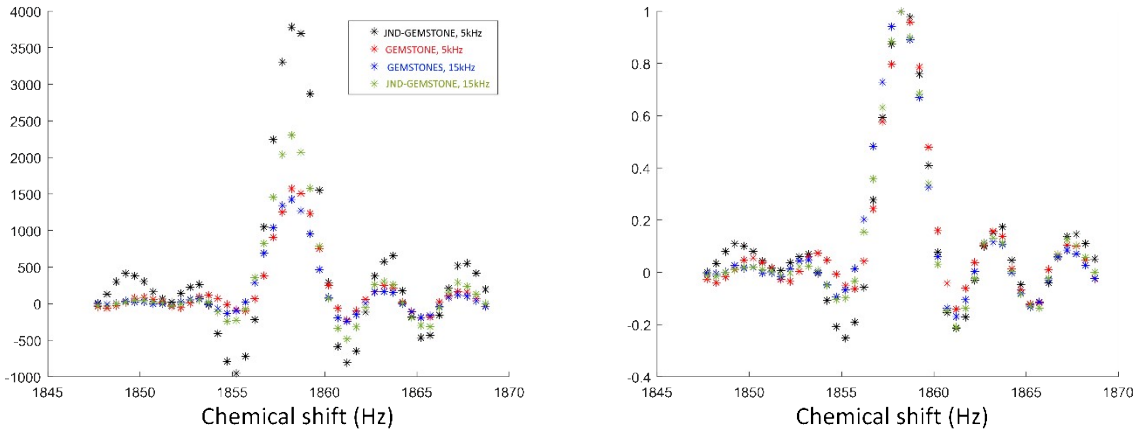

Fig. S1 Selectivity profile obtained from single-scan experiments, for each series of experiments (JND-GEMSTONE 5 kHz in black, GEMSTONE 5 kHz in red, GEMSTONES 15 kHz in blue and JND-GEMSTONE 15 kHz in green) the phase correction for the on-resonance experiment was applied to each dataset. The profiles were obtained using the cyclosporin signal at 3.2 ppm, displacing the selected offset from the actual one over a range of ±11 Hz in 0.5 Hz steps.

In Fig. S1 we can see that the central lobe is consistent for all sequences and parameters used, with a minor broadening for GEMSTONES (blue) because of the larger number of smoothed pulse envelope rises and falls. At the side lobes a small difference is observed for JND-GEMSTONE 5 kHz (black), the only curve that has the first side lobe almost symmetric about resonance. From theory, we expect symmetric selectivity profiles, and the experimental imperfections leading to the observed deviation are still not fully explained. However, the central lobe of the profiles is consistent for the different sequences and differences are mostly observed for the side lobes that are in any case an undesirable feature.

2.2 *J*-modulation

A major advantage of the JND-GEMSTONE pulse sequence element with respect to GEMSTONE(S) is the possibility to obtain clean lineshapes even using low bandwidths for the adiabatic pulses. In GEMSTONE(S) this would lead to distortions due to *J*-modulation. Examples of this are presented in Fig. S2.

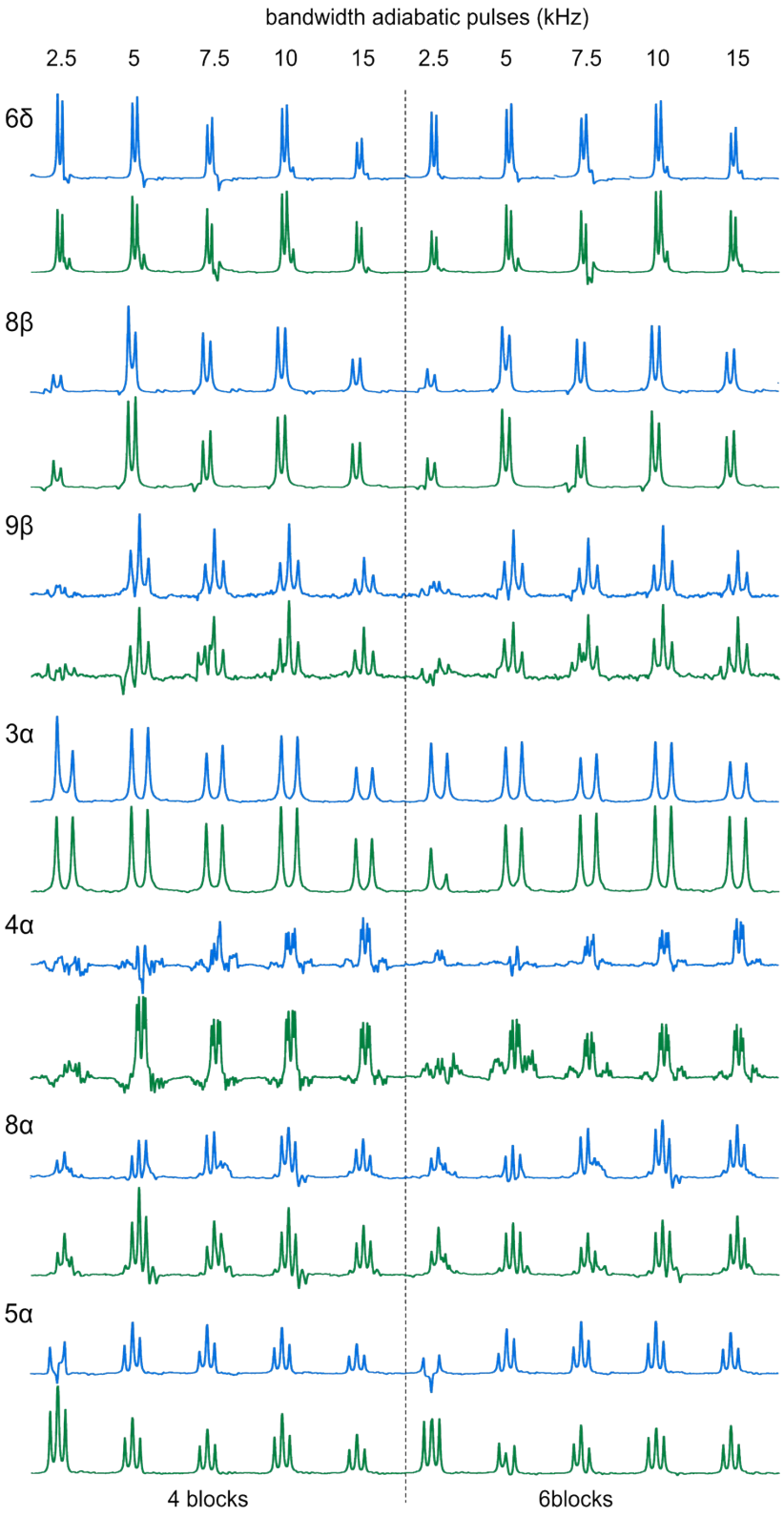

Fig. S2 Expansions of the ^{1}H spectrum of cyclosporin in C_6D_6. The expansion shows the signals of the different protons shown in the labels. Spectra are shown for the GEMSTONES (blue) and JND-GEMSTONE (green) pulse sequences, using an increasing bandwidth of the frequency-swept pulses (and matching gradient strength G_e). The minimum bandwidth needed for clean results is different for the different spin systems when using the GEMSTONE(S) sequence.

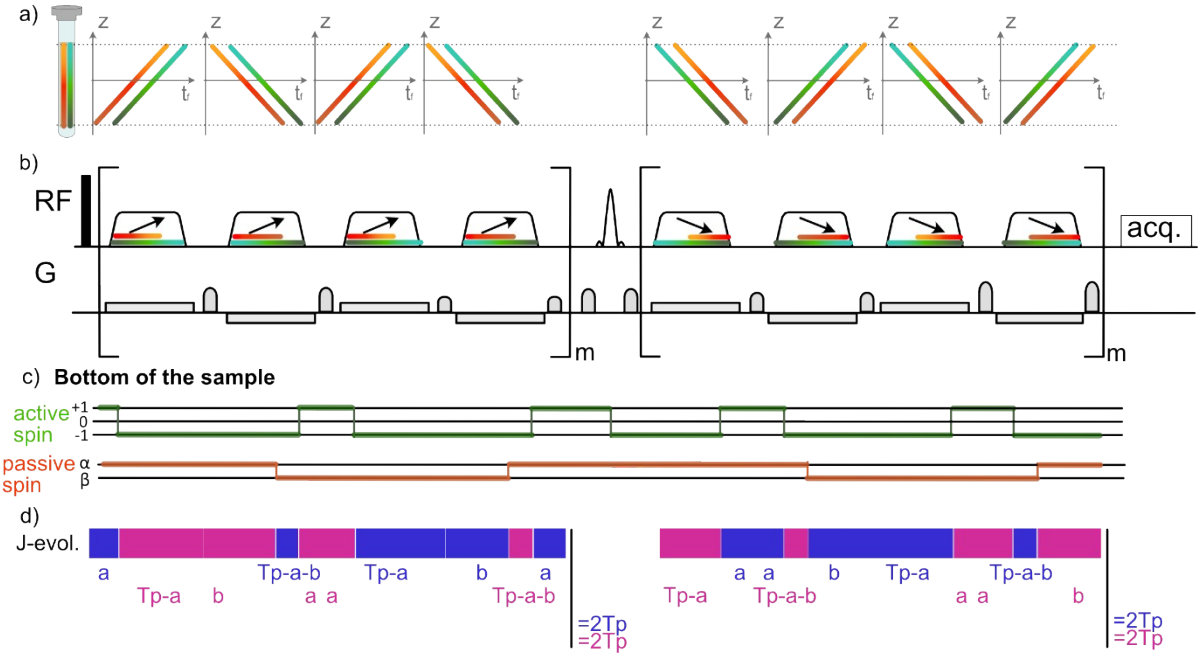

Fig. S3 Analysis of *J*-modulation during the JND-GEMSTONE pulse sequence, when using a low encoding gradient and a matched bandwidth for the chirp (meaning that the bandwidth is equal to frequency dispersion induced by the gradient for the active spins). Panel (a) shows the relationship between flip-time (t_f) and position for the active spins in green, and for the passive spins in red, for either positive or negative encoding gradient. The flip-time is related to the spin frequency by equation (1) below. The colour intensity corresponds to the z position in the sample, as shown by the vertical line in the NMR tube. The dashed horizontal lines show the bounds of the radiofrequency coil region. Panel (b) shows the JND-GEMSTONE pulse sequence. The overlaid horizontal coloured bars show the times at which active and passive spins are flipped by the frequency swept pulses. The dashed vertical line shows the bounds of the frequency range that is swept by the chirp pulses. It can be seen that only part of the passive spins are inverted every time. Panel (c) shows coherence transfer pathways for the active spin, and the evolution of the state of the passive spin (arbitrarily chosen to start in state α, the pathway starting from β would be symmetrical), for spins located at the bottom of the sample. Panel (d) shows the effective sign of the modulation by the J coupling, based on the coherence transfer pathway (CTP) of the active spin and the state of the passive spin – the sign changes when one or other of the spins is flipped. It can be seen that in this case the passive spins are inverted by one chirp pulse out of two. The variables a and b correspond to the flip times for the active and passive spins respectively, in the case of a positive sweep rate and positive encoding gradient. Overall, for this pulse sequence, *J*-evolution is refocused despite the fact that passive spins experience only one chirp pulse out of two. Similar analysis for using only one (as in GEMSTONE and GEMSTONES), or two chirp pulses shows that *J*-modulation does occur.

The time t_f at which a given spin is flipped depends on the pulse frequency sweep rate and initial frequency and its resonant frequency ω:

$$\omega(t_f) = \omega_{rf}^0 + Rt_f \qquad (1),$$

where R is the pulse sweep rate and ω_{rf}^0 is its initial frequency.

3. Mean diffusional attenuation calculation over the swept region of the GEMSTONE sequence

3.1 Methodology

Analytical calculations were performed with Mathematica 11.3 and Matlab R2021a to derive the equations for diffusion and mean diffusion attenuation.

The derivation was performed for ensembles of spin-1/2 for the sequences shown in Fig. S4. The two chirp pulses of opposite sweep direction were chosen to sweep over a bandwidth of 15 kHz in a duration of 100 ms. A gradient of amplitude 1.7 G cm^{-1} was applied with positive sign with forward sweep chirp pulse and with negative sign with reverse sweep chirp pulse, resulting in an encoding length of 16.6 mm. The selective π pulse was described as an instantaneous π rotation of the peak of interest, in the middle of a 16 ms delay. A time grid of 2000 points was distributed linearly over a length of 100 ms to plot the result of calculation. A diffusion coefficient of 4.6×10^{-10} m^2/s was used for the calculation. Relaxation effects were not taken into account in this simulation.

3.2 GEMSTONE calculation

Normalized diffusional attenuation for a combination of chirp pulse with positive gradient strength, is given by

$$\frac{S}{S_0} = e^{-Db(z)} \quad (1),$$

where S/S_0 is the normalized signal, D is the diffusion coefficient, and

$$\boldsymbol{b(z)} = \int \big(\boldsymbol{K(z,t)}\big)^2 \boldsymbol{dt}, \quad (2)$$

, where $K(z,t)$ is the phase variation of the magnetization along the spatial encoding axis and is given by

$$\boldsymbol{K(z,t) = \frac{\partial \phi(z,t)}{\partial z}}, \tag{3}$$

, where $\phi(z,t)$ represents the phase of the magnetization as a function of time during the sequence.

Hence the expression for $b(z)$ can be calculated for different sections of the pulse sequence using Eq 2 and 3.

GEMSTONE

GEMSTONE sequence consists of two blocks, one with a combination of chirp pulse with positive gradient and another with the negative gradient.

(i) Before $t_{flip,1}$,

$$\boldsymbol{K_1(z,t) = -\gamma G_e t, 0 \leq t < t_{flip,1}} \tag{4}$$

(ii) During the interval $t_{flip,1} \leq t < T_p$

$$\boldsymbol{K_2(z,t) = -\gamma G_e (t - 2t_{flip,1})} \tag{5}$$

(iii) During the first half of the selective pulse $0 \leq t < T_{sel}/2$

$$\boldsymbol{K_3(z) = -\gamma G_e (T_p - 2t_{flip,1})} \tag{6}$$

(iv) During second half of the selective pulse $T_{sel}/2 \leq t < T_{sel}$

$$\boldsymbol{K_4(z) = \gamma G_e (T_p - 2t_{flip,1})} \tag{7}$$

(v) After the selective π pulse, a chirp pulse of reverse sweep with negative gradient strength is applied. It results in the phase variation in the interval $0 \le t < t_{flip,2}$

$$\boldsymbol{K_5(z,t) = \gamma G_e(t + T_p - 2t_{flip,1})} \tag{8}$$

(vi) In the final interval of the sequence, $t_{flip,2} \le t < T_p$

$$\boldsymbol{K_6(z,t) = \gamma G_e(t - T_p)} \tag{9}$$

The instantaneous flip times $t_{flip,1}$ and $t_{flip,2}$ in this case will be given by

$$\boldsymbol{t_{flip,1}(z) = \frac{-\gamma G_e z - \omega_{i1}}{R_1}} \tag{10}$$

$$\boldsymbol{t_{flip,2}(z) = \frac{\gamma G_e z - \omega_{i2}}{R_2}} \tag{11}$$

,where $\omega_{i1} = -\frac{2\pi BW}{2}$, $\omega_{i2} = \frac{2\pi BW}{2} = -\omega_{i1}$, $R_1 = \frac{2\pi BW}{T_e}$, and $R_2 = -\frac{2\pi BW}{T_e} = -R_1$

With these parameters, equations (10) and (11) become identical and hence $t_{flip,1}(z) = t_{flip,2}(z)$

Now $b(z)$ for the sequence can be calculated as

$$\begin{aligned}\boldsymbol{b} = &\int_0^{t_{flip,1}} \big(\boldsymbol{K_1(z,t)}\big)^2 \boldsymbol{dt_1} + \int_{t_{flip,1}}^{T_p} \big(\boldsymbol{K_2(z,t)}\big)^2 \boldsymbol{dt_2} + \int_0^{T_{sel}/2} \big(\boldsymbol{K_3(z)}\big)^2 \boldsymbol{dt_3} + \int_{T_{sel}/2}^{T_{sel}} \big(\boldsymbol{K_4(z)}\big)^2 \boldsymbol{dt_4} \\ &+ \int_0^{t_{flip,2}} \big(\boldsymbol{K_5(z,t)}\big)^2 \boldsymbol{dt_3} + \int_{t_{flip,2}}^{T_p} \big(\boldsymbol{K_6(z,t)}\big)^2 \boldsymbol{dt_4}\end{aligned} \tag{12}$$

Combining equations (4), (5), (6), (7), (8), (9), and (12) we get

$$\boldsymbol{b = \gamma^2 G_e^2 T_p^3 \left(\frac{1}{6} + \frac{2(\gamma G_e z)^2}{(2\pi BW)^2}\right) + \gamma^2 G_e^2 T_p^2 \frac{4T_{sel}(\gamma G_e z)^2}{(2\pi BW)^2}} \tag{13}$$

If we replace $-\frac{2\pi BW}{\gamma G_e} = L$ in equation (13), it gives

$$\boldsymbol{b = \gamma^2 G_e^2 T_p^3 \left(\frac{1}{6} + \frac{2z^2}{L^2}\right) + \gamma^2 G_e^2 T_p^2 T_{sel} \frac{4z^2}{L^2}} \tag{14}$$

$$\boldsymbol{b = \gamma^2 G_e^2 T_p^3 \left(\frac{1}{6}\right) + \gamma^2 G_e^2 T_p^2 \frac{2z^2}{L^2} \left(T_p + 2T_{sel}\right)} \tag{15}$$

Hence from equations (1) and (15) we get

$$\frac{S}{S_0} = e^{-D\left(\frac{1}{6}\gamma^2 G_e^2 T_p^3 + \gamma^2 G_e^2 T_p^2 \frac{2z^2}{L^2}\left(T_p + 2T_{sel}\right)\right)} \tag{16}$$

$$\boldsymbol{\frac{S}{S_0} = e^{-\frac{1}{6}\gamma^2 G_e^2 T_p^3 D} e^{-\gamma^2 G_e^2 T_p^2 \frac{4z^2}{L^2} D\left(T_{sel} + \frac{T_p}{2}\right)}} \tag{17}$$

When we consider multiple blocks, equation (16) can be modified to give signal attenuation

$$\boldsymbol{\frac{S}{S_0} = e^{-\frac{1}{6}\gamma^2 G_e^2 \left(\frac{T_e}{2n}\right)^3 D} e^{-\gamma^2 G_e^2 \left(\frac{T_e}{2n}\right)^2 \frac{4z^2}{L^2} D\left(T_{sel} + \frac{T_e}{2n}\right)} e^{-\frac{1}{6}\gamma^2 G_e^2 \left(\frac{T_e}{2n}\right)^3 D} e^{-\gamma^2 G_e^2 \left(\frac{T_e}{2n}\right)^2 \frac{4z^2}{L^2} D\left(T_{sel} + \frac{T_e}{4n}\right)} \ldots} \tag{18}$$

$$\boldsymbol{\frac{S}{S_0} = e^{-\frac{1}{6}\gamma^2 G_e^2 \left(\frac{T_e}{2n}\right)^3 Dn} e^{-n\gamma^2 G_e^2 \left(\frac{T_e}{2n}\right)^2 \frac{4z^2}{L^2} D\left(T_{sel} + \frac{T_e}{4n}\right)}} \tag{19}$$

$$\boldsymbol{\frac{S}{S_0} = e^{-\frac{1}{6}\gamma^2 G_e^2 \left(\frac{T_e}{2n}\right)^2 \frac{T_e}{2} D} e^{-\gamma^2 G_e^2 \left(\frac{T_e}{2}\right)^2 \frac{4z^2}{L^2} D\left(nT_{sel} + \frac{T_e}{4}\right)}} \tag{20}$$

Hence, mean attenuation over the spatial encoded region can be calculated as

$$A_n(z) = \frac{1}{L} \int_{-L/2}^{L/2} e^{-\frac{1}{6}\gamma^2 G_e^2 \left(\frac{T_e}{2n}\right)^2 \frac{T_e}{2} D} e^{-4\gamma^2 G_e^2 \left(\frac{T_e}{2n}\right)^2 \frac{z^2}{L^2} D\left(nT_{sel} + \frac{T_e}{4}\right)} dz \tag{21}$$

$$\boldsymbol{A_n(z) = e^{-\frac{1}{6}\gamma^2 G_e^2 \left(\frac{T_e}{2n}\right)^2 \frac{T_e}{2} D} \frac{1}{L} \int_{-L/2}^{L/2} e^{-4\gamma^2 G_e^2 \left(\frac{T_e}{2n}\right)^2 \frac{z^2}{L^2} D\left(nT_{sel} + \frac{T_e}{4}\right)} dz} \tag{22}$$

Let $\beta_n = \sqrt{4\gamma^2 G_e^2 \left(\frac{T_e}{2n}\right)^2 D\left(nT_{sel} + \frac{T_e}{4}\right)}$ and $u = z/L$, hence $du = \frac{dz}{L}$

$$A_n(t) = e^{-\frac{\beta_1^2}{24\left(\frac{2nT_{sel}}{T_e} + \frac{1}{2}\right)}} \int_{-1/2}^{1/2} e^{-\beta_1^2 u^2} du = e^{-\frac{\beta_1^2}{24\left(\frac{2nT_{sel}}{T_e} + \frac{1}{2}\right)}} \times 2 \int_0^1 e^{-\beta_1^2 u^2} du \tag{23}$$

Let $\beta_n u = v$, $du = \frac{dv}{\beta_1}$

$$A_n = e^{-\frac{\beta_n^2}{24\left(\frac{2nT_{sel}}{T_e} + \frac{1}{2}\right)}} \frac{1}{\beta_n} \times 2 \int_0^{\beta_n} e^{-v^2} dv = 2e^{-\frac{\beta_n^2}{24\left(\frac{2nT_{sel}}{T_e} + \frac{1}{2}\right)}} \frac{1}{\beta_n} \frac{\sqrt{\pi}}{2} \, erf(\beta_n) \tag{24}$$

,where $erf(\beta_n) = \frac{2}{\sqrt{\pi}} \int_0^{\beta_n} e^{-v^2} dv$.

JND-GEMSTONE

JND-GEMSTONE sequence consists of a selective π pulse sandwiched between two identical blocks, and the block has a combination of two chirp pulses, first with positive gradient and the second with the negative gradient.

(i) Before $t_{flip,1}$,

$$K_1(z,t) = -\gamma G_e t, 0 \le t < t_{flip,1} \tag{25}$$

(ii) During the interval $t_{flip,1} \le t < T_p$

$$\boldsymbol{K_2(z,t) = -\gamma G_e\left(t - t_{flip,1}\right) + \gamma G t_{flip,1} = -\gamma G_e\left(t - 2t_{flip,1}\right)} \tag{26}$$

(iii) After the first chirp pulse+gradient, and before $t_{flip,2}$ i.e. $0 \le t < t_{flip,2}$

$$\boldsymbol{K_3(z,t) = \gamma G_e t - \gamma G_e\left(T_p - 2t_{flip,1}\right) = \gamma G_e\left(t - T_p + 2t_{flip,1}\right)} \tag{27}$$

(iv) During the interval $t_{flip,2} \le t < T_p$

$$\boldsymbol{K_4(z,t) = \gamma G_e\left(t - t_{flip,2}\right) - \gamma G_e\left(t_{flip,2} - T_p + 2t_{flip,1}\right) = \gamma G_e\left(t - 2t_{flip,2} + T_p - 2t_{flip,1}\right)} \tag{28}$$

(v) During the first half of the π pulse $0 \le t < T_{sel}/2$

$$\boldsymbol{K_5(z) = \gamma G_e(2T_p - 2t_{flip,2} - 2t_{flip,1})} \tag{29}$$

(vi) During the second half of the π pulse $T_{sel}/2 \le t < T_{sel}$

$$\boldsymbol{K_6(z) = \gamma G_e(2T_p - 2t_{flip,2} - 2t_{flip,1})} \tag{30}$$

The instantaneous flip times $t_{flip,1}$ and $t_{flip,2}$ are given by

$$t_{flip,1}(z) = \frac{-\gamma G_e z - \omega_{i1}}{R_1} \tag{31}$$

$$t_{flip,2}(z) = \frac{\gamma G_e z - \omega_{i2}}{R_2} \tag{32}$$

, where $\omega_{i1} = -\frac{2\pi BW}{2}$, $\omega_{i2} = -\frac{2\pi BW}{2}, = \omega_{i1}$, $R_1 = \frac{2\pi BW}{T_p}$, and $R_2 = \frac{2\pi BW}{T_p} = R_1$.

Using these values and replacing $-\frac{2\pi BW}{\gamma G_e}$ with L, the instantaneous flip time can also be written as

$$\boldsymbol{t_{flip,1}(z) = T_p\left(\frac{z}{L} + \frac{1}{2}\right)} \tag{33}$$

$$\boldsymbol{t_{flip,2}(z) = T_p\left(\frac{1}{2} - \frac{z}{L}\right)} \tag{34}$$

With these definitions we have $t_{flip,1} + t_{flip,2} = T_p$ and hence the phase variation shown in equations (29) and (30) becomes zero. It implies that there is no position-dependent phase encoding during the selective π pulse.

(vii) After the selective π pulse a chirp pulse of reverse sweep with positive gradient strength is applied. It results in the phase variation in the interval $0 \le t < t_{flip,3}$ as follows

$$\boldsymbol{K_7(z,t) = -\gamma G_e t} \tag{35}$$

(viii) During the interval $t_{flip,3} \le t < T_p$

$$\boldsymbol{K_8(z,t) = -\gamma G_e\left(t - t_{flip,3}\right) + \gamma G_e t_{flip,3} = -\gamma G_e(t - 2t_{flip,3})} \tag{36}$$

(ix) After the third reverse swept chirp pulse and before $t_{flip,4}$ $0 \le t < t_{flip,4}$

$$\boldsymbol{K_9(z,t) = \gamma G_e t - \gamma G_e\left(T_p - 2t_{flip,3}\right) = \gamma G_e\left(t - T_p + 2t_{flip,3}\right)} \tag{37}$$

(x) During the final interval, $t_{flip,4} \le t < T_p$

$$\boldsymbol{K_{10}(z,t) = \gamma G_e\left(t - t_{flip,4}\right) - \gamma G_e\left(t_{flip,4} - T_p + 2t_{flip,3}\right)} \tag{38}$$

The instantaneous flip times $t_{flip,3}$ and $t_{flip,4}$ in this case will be given by

$$\boldsymbol{t_{flip,3}(z) = \frac{-\gamma G_e z - \omega_{i3}}{R_3} = \frac{-\gamma G_e z + \omega_{i2}(=\omega_{i1})}{-R_1(=-R_2)} = \frac{\gamma G_e z - \omega_{i2}}{R_2} = t_{flip,2}(z)} \tag{39}$$

$$\boldsymbol{t_{flip,4}(z) = \frac{\gamma G_e z - \omega_{i4}}{R_4} = t_{flip,1}(z)} \tag{40}$$

, where $\omega_{i3} = \dfrac{2\pi BW}{2} = -\omega_{i1}$, $\omega_{i4} = \dfrac{2\pi BW}{2} = -\omega_{i1}$, $R_3 = -\dfrac{2\pi BW}{T_p} = -R_1$,

and $R_4 = -\dfrac{2\pi BW}{T_p} = -R_1$

With these parameters $t_{flip,3}(z)$ and $t_{flip,4}(z)$ becomes identical to $t_{flip,2}$ and $t_{flip,1}$, respectively.

Now $b(z)$ for the sequence can be calculated as

$$\begin{aligned}\boldsymbol{b} = &\int_0^{t_{flip,1}} \left(\boldsymbol{K_1(z,t)}\right)^2 \boldsymbol{dt_1} + \int_{t_{flip,1}}^{T_p} \left(\boldsymbol{K_2(z,t)}\right)^2 \boldsymbol{dt_2} + \int_0^{t_{flip,2}} \left(\boldsymbol{K_3(z,t)}\right)^2 \boldsymbol{dt_3} + \int_{t_{flip,2}}^{T_p} \left(\boldsymbol{K_4(z,t)}\right)^2 \boldsymbol{dt_4} \\ &+ \int_0^{T_{sel}/2} \left(\boldsymbol{K_4(z)}\right)^2 \boldsymbol{dt_5} + \int_{T_{sel}/2}^{T_{sel}} \left(\boldsymbol{K_5(z)}\right)^2 \boldsymbol{dt_6} + \int_0^{t_{flip,3}} \left(\boldsymbol{K_7(z,t)}\right)^2 \boldsymbol{dt_7} \\ &+ \int_{t_{flip,3}}^{T_p} \left(\boldsymbol{K_8(z,t)}\right)^2 \boldsymbol{dt_8} + \int_0^{t_{flip,4}} \left(\boldsymbol{K_9(z,t)}\right)^2 \boldsymbol{dt_9} + \int_{t_{flip,4}}^{T_p} \left(\boldsymbol{K_{10}(z,t)}\right)^2 \boldsymbol{dt_{10}}\end{aligned} \tag{41}$$

Using the phase variations from $K_1(z,t)$ to $K_{10}(z,t)$ in equation (41), b comes out to be

$$\boldsymbol{b = \gamma^2 G_e^2 T_p^3 \left(\frac{1}{3} + \frac{4(\gamma G_e z)^2}{(2\pi BW)^2}\right)} \tag{42}$$

If we replace $-\dfrac{2\pi BW}{\gamma G_e} =, L$ equation (42) gets converted to

$$\boldsymbol{b = \gamma^2 G_e^2 T_p^3 \left(\frac{1}{3} + \frac{4z^2}{L^2}\right)} \tag{43}$$

Now from equations (1) and (43)

$$\frac{S}{S_0} = e^{-D\gamma^2 G_e^2 T_p^3\left(\frac{1}{3}+\frac{4z^2}{L^2}\right)} = e^{-\frac{D\gamma^2 G_e^2 T_p^3}{3}} e^{-D\gamma^2 G_e^2 T_p^3 \frac{4z^2}{L^2}} \tag{44}$$

When we consider multiple blocks, equation (44) can be modified to give a signal attenuation

$$\frac{S}{S_0} = e^{-\frac{D\gamma^2 G_e^2 \left(\frac{T_e}{4m}\right)^3}{3}} e^{-D\gamma^2 G_e^2 \left(\frac{T_e}{4m}\right)^3 \frac{4z^2}{L^2}} e^{-\frac{D\gamma^2 G_e^2 \left(\frac{T_e}{4m}\right)^3}{3}} e^{-D\gamma^2 G_e^2 \left(\frac{T_e}{4m}\right)^3 \frac{4z^2}{L^2}} \ldots \tag{45}$$

$$\frac{S}{S_0} = e^{-\frac{D\gamma^2 G_e^2 \frac{T_e}{4}\left(\frac{T_e}{4m}\right)^2}{3}} e^{-D\gamma^2 G_e^2 \frac{T_e}{4}\left(\frac{T_e}{4m}\right)^2 \frac{4z^2}{L^2}} \tag{46}$$

Now, the mean attenuation over the spatial encoded region can be calculated as

$$A_m(z) = \frac{1}{L}\int_{-L/2}^{L/2} e^{-\frac{1}{3}D\gamma^2 G_e^2 \frac{T_e}{4}\left(\frac{T_e}{4m}\right)^2} e^{-D\gamma^2 G_e^2 T_e \left(\frac{T_e}{4m}\right)^2 \frac{4z^2}{L^2}} dz \tag{47}$$

$$A_m(z) = 2e^{-\frac{1}{3}D\gamma^2 G_e^2 \frac{T_e}{4}\left(\frac{T_e}{4m}\right)^2} \frac{1}{L}\int_0^L e^{-D\gamma^2 G_e^2 \frac{T_e}{4}\left(\frac{T_e}{4m}\right)^2 \frac{4z^2}{L^2}} dz \tag{48}$$

Let $\beta_m = \sqrt{4\gamma^2 G_e^2 \left(\frac{\mathrm{T_e}}{4\mathrm{m}}\right)^2 \frac{\mathrm{T_e}}{4} D}$ and $u = z/L$, hence $du = \frac{dz}{L}$

$$A_m(u) = 2e^{-\frac{\beta_m^2}{12}} \int_0^1 e^{-\beta_m^2 u^2} du \tag{49}$$

Let $\beta_m u = v$, $du = \frac{dv}{\beta_m}$

$$A_m = \frac{2}{\beta} e^{-\frac{\beta_m^2}{12}} \int_0^{\beta_m} e^{-v^2} dv \tag{50}$$

Since $erf(\beta_m) = \frac{2}{\sqrt{\pi}}\int_0^{\beta_m} e^{-v^2} dv$,

$$A_m = \frac{2}{\beta_m} e^{-\frac{\beta_m^2}{12}} \frac{\sqrt{\pi}}{2} erf(\beta_m) \tag{51}$$

3.3 Numerical Application

A comparison of diffusional attenuation for GEMSTONES and JND-GEMSTONE is shown in Fig. S5 for the calculated mean attenuations for one, four, and eight blocks with respect to the pulse duration. This calculation captures qualitatively the effect observed experimentally, although the agreement is not quantitative and other contributions will have to be accounted for.

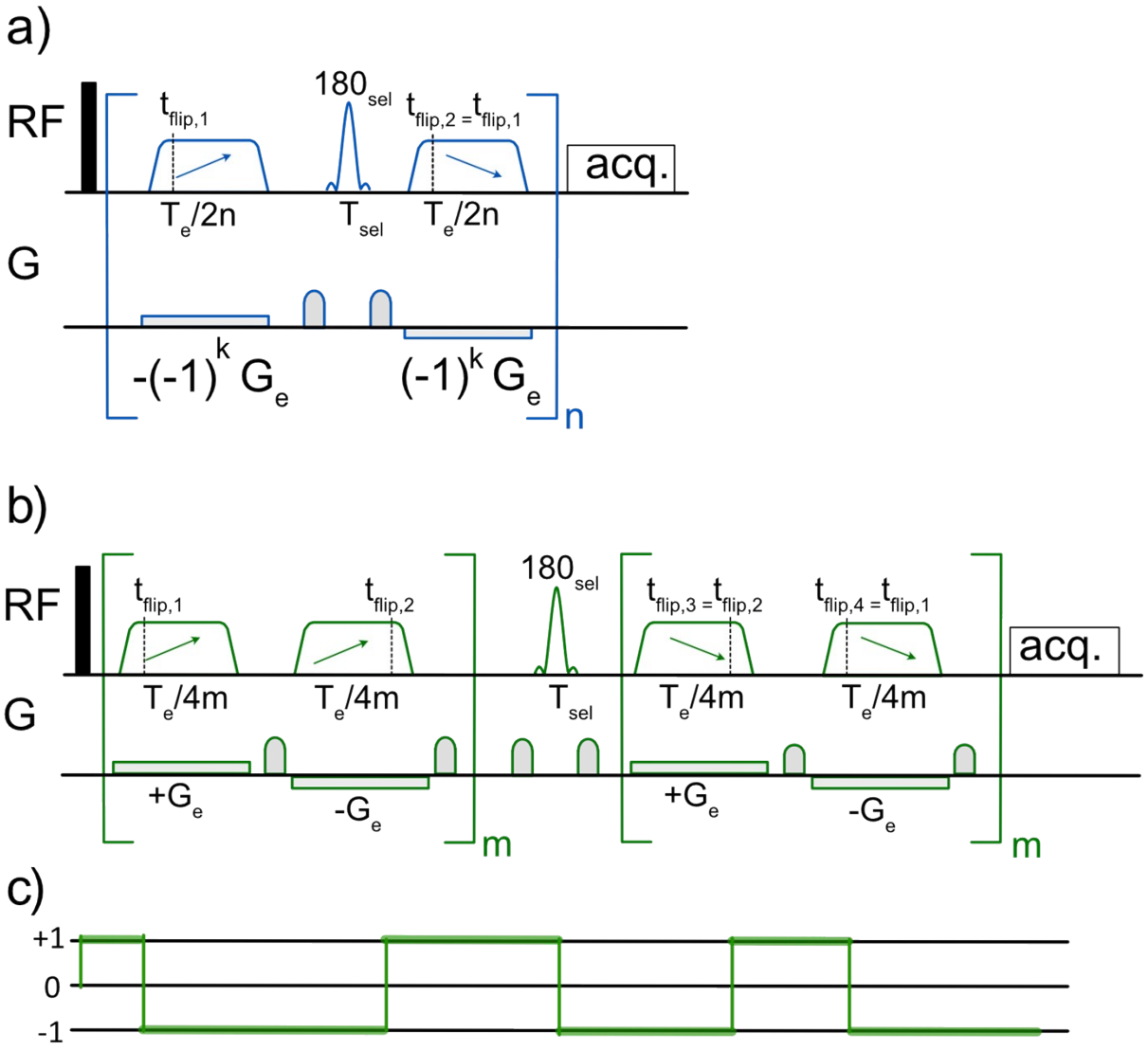

Fig. S4 a) The GEMSTONES pulse sequence element, b) The proposed JND-GEMSTONE element, c) the coherence transfer pathway for JND-GEMSTONE.

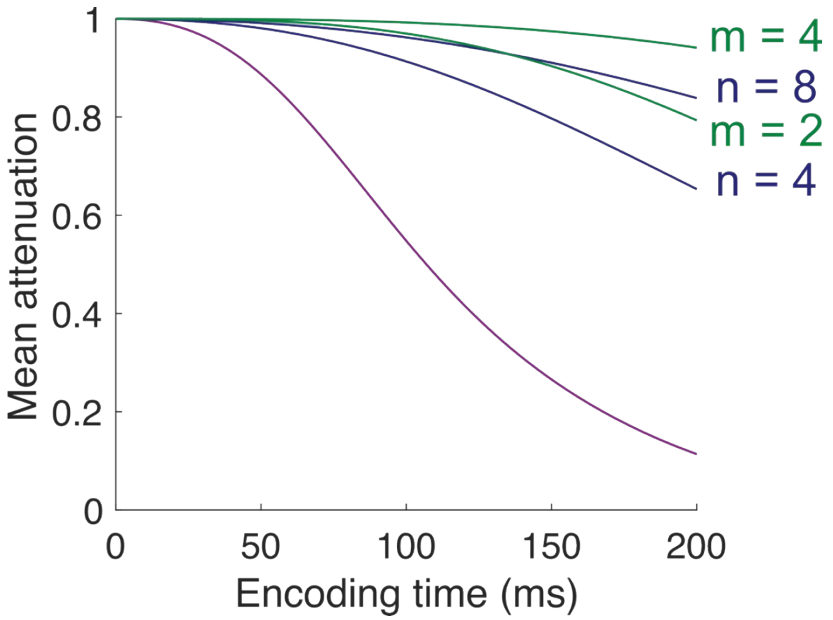

Fig. S5 Variation of mean attenuation as function of pulse length for the single block GEMSTONE sequence (purple), for the GEMSTONES sequence with n = 4 and 8 (blue), and for the JND-GEMSTONE sequence with m = 2 and 4. Calculations used a diffusion coefficient of $4.6*10^{-10}$ m^2/s (as obtained from DOSY for cyclosporin in C_6D_6), G_e = 1.70 G cm^{-1}, $T_e/2$ = 100 ms and a chirp pulse bandwidth of 15 kHz.

4. Applications in correlation experiments

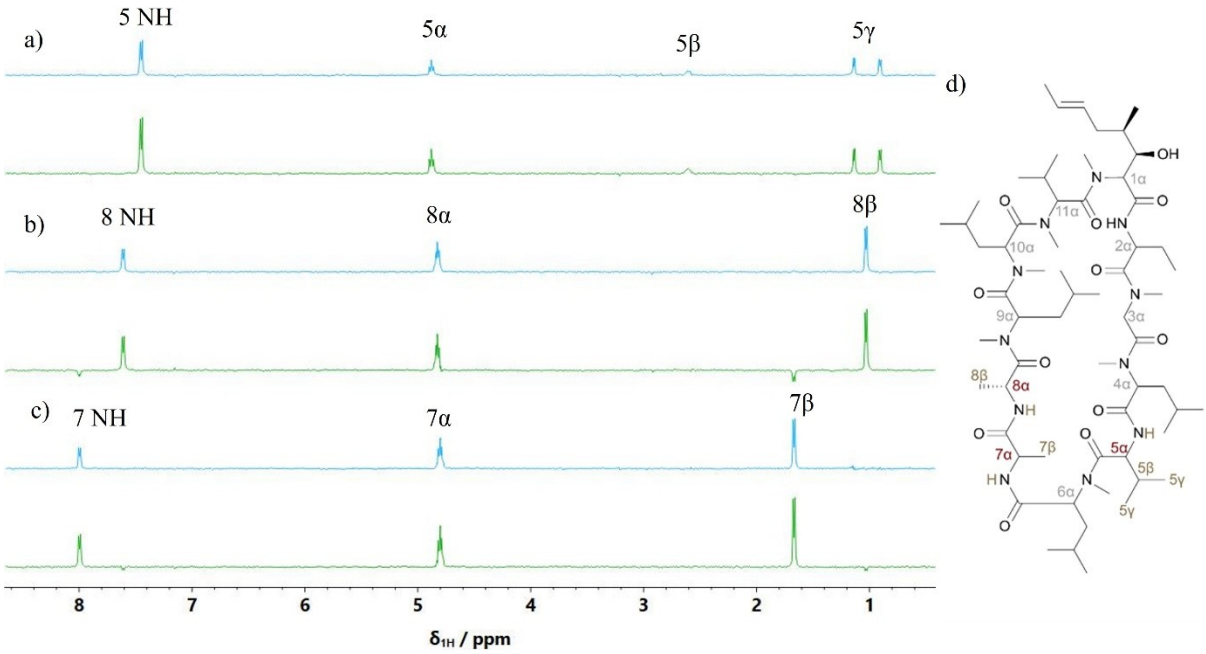

Fig. S6 a)-c) Selective TOCSY experiments using the GEMSTONES (blue, n = 6) and JND-GEMSTONE (green, m = 1) pulse sequences applied to three amino acids from cyclosporin. For each of the three amino acids, the H_α proton (blue) was selectively excited and coherence transferred. The chirp bandwidths were 5 kHz for JND-GEMSTONE and 15 kHz for GEMSTONES, to avoid *J*-modulation. These are the optimal values in the two experiments. JND-GEMSTONE allows the use of smaller chirp bandwidths, but in this example there is no significant difference in sensitivity as both GEMSTONES and JND-GEMSTONE minimize the signal losses by diffusion for a large molecule such as cyclosporin. The encoding and TOCSY mixing times were 300 and 60 ms, respectively. d) Structure of cyclosporin

The signal to noise ratio was measured for each peak using a 500 Hz (1 ppm) noise region. The sensitivity of using JND-GEMSTONE is consistently better, but as expected differs between resonances. Residual J modulation effects can distort such comparisons, but JND-GEMSTONE should always provide the better sensitivity as it avoids repeated application of the selective (rSNOB) pulse.

Table S1 Signal to noise measured from spectra in Fig.S6a

	5 NH	**5α**	**5β**	**5γ**	**5γ**
GEMSTONES	66.5	29.2	8.5	33.2	30.5
JND-GEMSTONE	98.9	43.2	8.9	44.3	42.2
difference	1.5	1.5	1.1	1.4	1.4

Table S2 Signal to noise measured from spectra in Fig.S6b

	8 NH	**8α**	**8β**
GEMSTONES	42.6	55.4	85.3
JND-GEMSTONE	55.5	59.3	99.4
difference	1.4	1.1	1.2

Table S3 Signal to noise measured from spectra in Fig.S6c

	7 NH	**7α**	**7β**
GEMSTONES	39.4	58.1	93.9
JND-GEMSTONE	55.0	69.4	116.7
difference	1.4	1.2	1.3

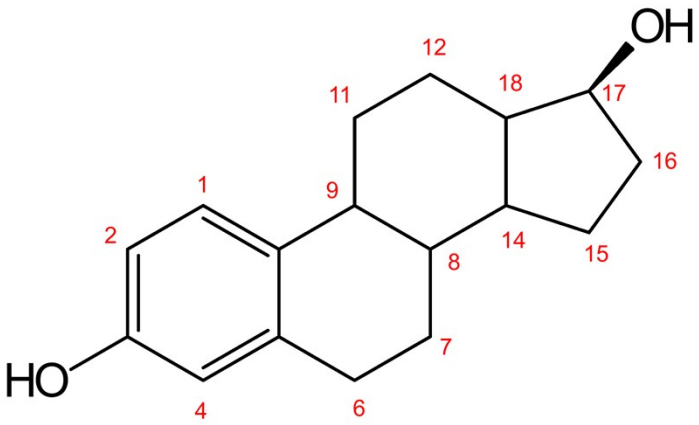

Fig. S7 Structure of 17β-estradiol; labels refer to proton numbering for peak assignments below.

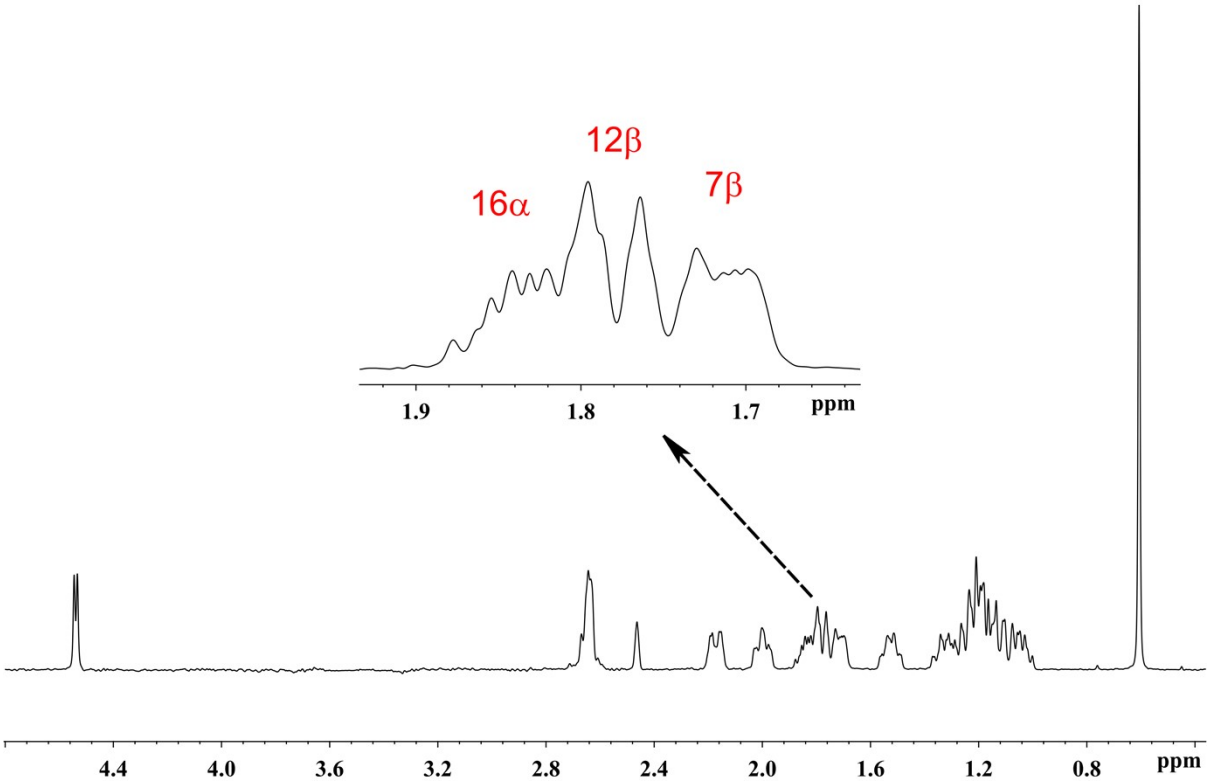

Fig. S8 ^{1}H NMR spectrum of 17β-estradiol in dmso-d_6. The inset shows the overlapping region used for testing application of the new JND-GEMSTONE pulse sequence element with COSY, TOCSY, NOE, and ROE. Experiments were carried out to selectively observe only the correlations from H-12β. A large (about x250 times the signals of interest) water peak at 3.5 ppm was removed by using a time-domain solvent filter in the data processing in JASON, based on the work of D. Marion, M. Ikura and A. Bax, *J. Magn. Reson.* 1989, **84**, 425-430.

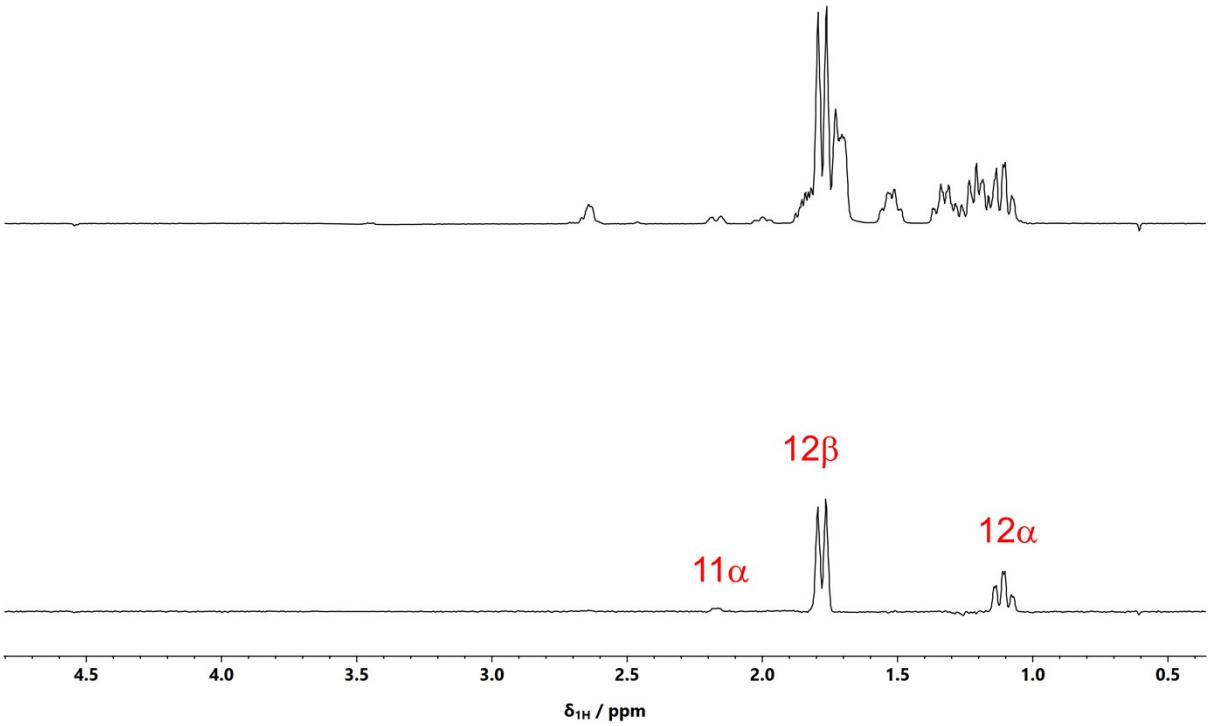

Fig. S9 Conventional (top) and JND-GEMSTONE (bottom) selective CLIP COSY spectra of 17β-estradiol in dmso-d_6.

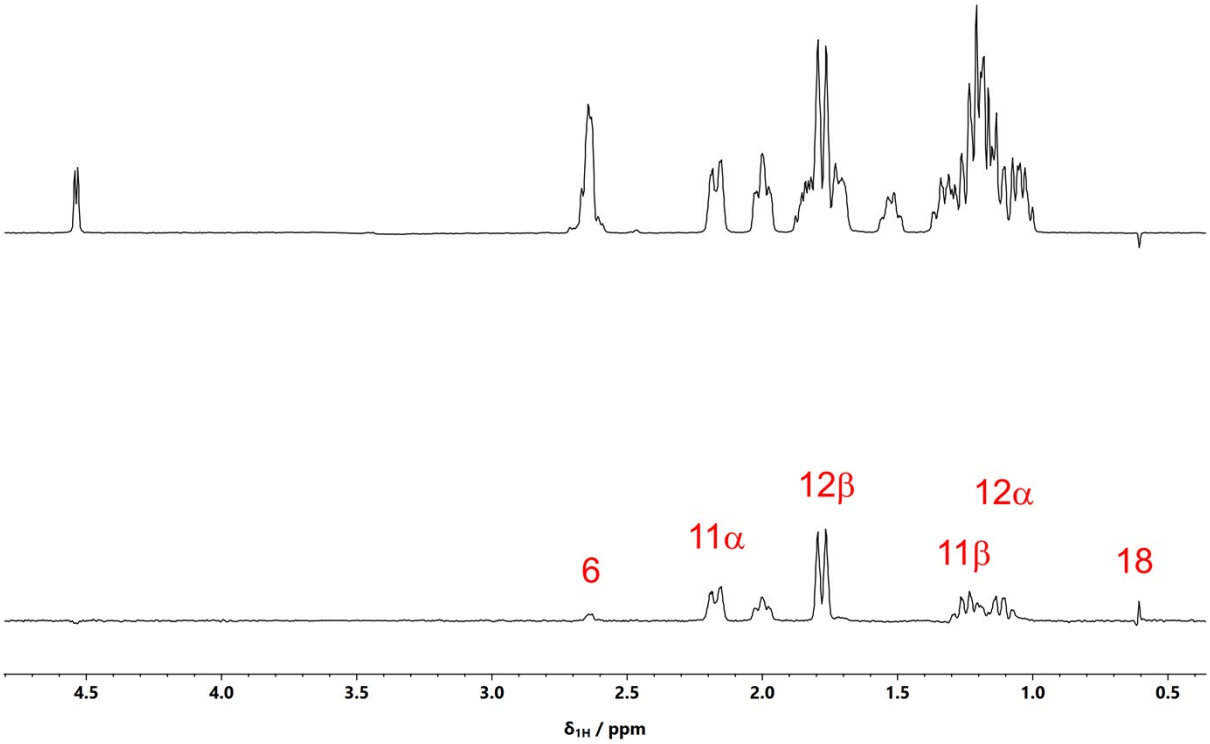

Fig. S10 Conventional (top) and JND-GEMSTONE (bottom) selective TOCSY spectra of 17β-estradiol in dmso-d_6. Additional signals appear in the conventional spectrum due to insufficient selectivity.

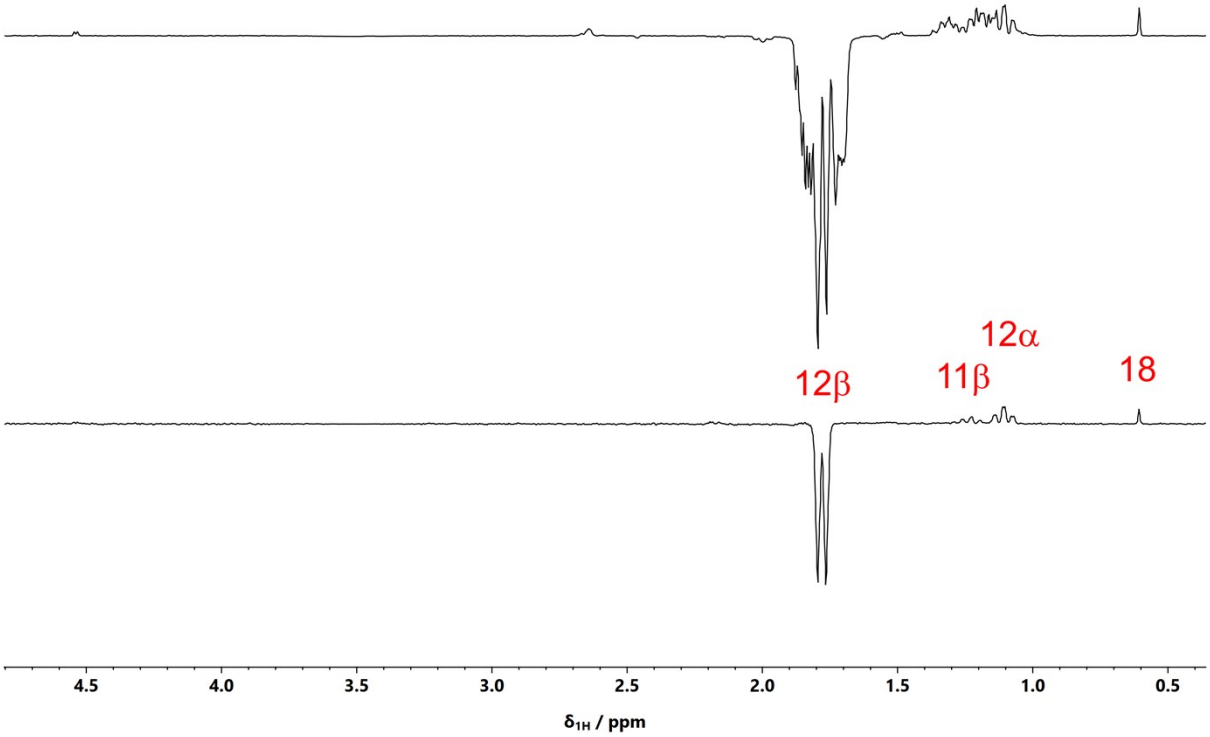

Fig. S11 Conventional (top) and JND-GEMSTONE (bottom) selective NOESY spectra of 17β-estradiol in dmso-d_6. Additional signals appear in the conventional spectrum due to insufficient selectivity.

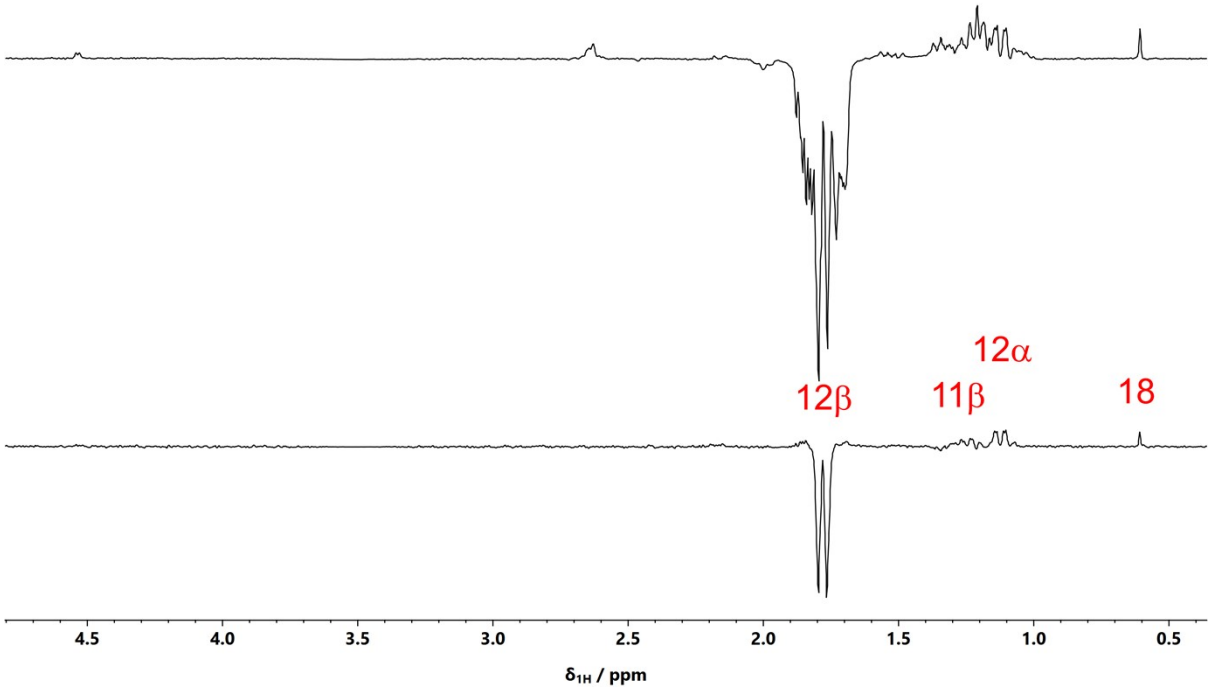

Fig. S12 Conventional (top) and JND-GEMSTONE (bottom) selective easyROESY spectra of 17β-estradiol in dmso-d_6. Additional signals appear in the conventional spectrum due to insufficient selectivity.

5. Pulse programs

All pulse programs and original data files are available from DOI: 10.48420/31502338

It is recommended to use the electronic copies of pulse programs to reproduce the experiments described, but text format is provided below.

5.1 JND-GEMSTONE for Bruker spectrometers

```
;single scan JND-GEMSTONE experiment
;avance-version

;Ultra-Selective NMR Experiment with Preserved Sensitivity
;Based on:
;P. Kiraly, N. Kern, M. P. Plesniak, M. Nilsson, D. J. Procter, G.A. Morris, and R.
W. Adams
; Angew.Chem. (2020) https://doi.org/10.1002/anie.202011642
;
;M. Bazzoni, R. Mishra, and J.-N. Dumez
;Angew. Chem. Int. Ed. 2023, e202314598
;
;$CLASS=HighRes
;$DIM=1D
;$TYPE=
;$SUBTYPE=
;$COMMENT=


#include <Avance.incl>
#include <Grad.incl>
#include <Delay.incl>

define list<gradient> gl2 = { 0.87 1.17 0.73 1.23 }

"FACTOR1=(d9/(p6*115.112))/2"
"l1=FACTOR1*2"

define loopcounter n
"n=cnst49"

"spoff29=0"
"spoff30=0"


"d11=30m"
"d49=cnst49*1u"


"acqt0=-p1*2/PI"


1 ze
2 30m
  20u BLKGRAD
  20u pl1:f1
  d1 pl1:f1
  20u UNBLKGRAD

  (p1 ph1):f1

;first encoding block 2*n chirp
3 d17 igrad gl2
  (p41:sp41 ph6):f1 (p41:gp11)
  d17
  p17:gp13 * gl2
```

```
  d16
  (p41:sp41 ph7):f1 (p41:gp12)
  d16
  p17:gp13 * gl2
lo to 3 times n

; central selective 180 pulse
  d16
  p16:gp1
  d16
  p12:sp2:f1 ph2:r
  d16
  p16:gp1
  d16

;second encoding block 2*n chirp
 4 d17 igrad gl2
  (p42:sp42 ph6):f1 (p42:gp11)
  d17
  p17:gp14*gl2
  d16
  (p42:sp42 ph7):f1 (p42:gp12)
  d16
  p17:gp14*gl2
  lo to 4 times n

  go=2 ph31
  30m mc #0 to 2 F0(zd)
       20u BLKGRAD

d49

exit

ph1=0
ph2=0 1

ph6=0
ph7=0
ph31=0 2


;pl0 : 0W
;pl1 : f1 channel - power level for pulse (default)
;sp2: f1 channel - shaped pulse
;p1 : f1 channel -  90 degree high power pulse
;p12: f1 channel - 180 degree shaped pulse
;p16: homospoil/gradient pulse                      [1 msec]
;p41: adiabatic pulse
;p42: reverse sweep adiabatic pulse
;d1 : relaxation delay; 1-5 * T1

;d16: delay for homospoil/gradient recovery
;d17: adjustable delay surrounding the encoding pulses should be d16+p17
;cnst49: number of chirp pair per side of the sel180

;n: loop for GEMSTONE cycle (=cnst49)

;ns: even number, total number of scans: NS * TD0
;ds: 4

;phcor 2 : phasedifference between power levels sp1 and pl1

;for z-only gradients:
;gpz11: matched to BW
;gpz12: -gpz11

;use gradient files:
```

```
;gpnam11: RECT.1
;gpnam12: RECT.1


;for sweepwidth of adiabatic shape and adjusting gpz0
;   see supplementary material of M.J. Thrippleton & J. Keeler,
;   Angew. Chem. Int. Ed. 42, 3938-3941 (2003)
```

5.2 JND-GEMSTONE TOCSY for Bruker spectrometers

```
;single scan JND-GEMSTONE experiment
;avance-version

;Ultra-Selective NMR Experiment with Preserved Sensitivity
;
; P. Kiraly, N. Kern, M. P. Plesniak, M. Nilsson, D. J. Procter, G.A. Morris, and R.
W. Adams
; Angew.Chem. (2020) https://doi.org/10.1002/anie.202011642
;
;M. Bazzoni, R. Mishra, and J.-N. Dumez
;Angew. Chem. Int. Ed. 2023, e202314598
;
;$CLASS=HighRes
;$DIM=1D
;$TYPE=
;$SUBTYPE=
;$COMMENT=


#include <Avance.incl>
#include <Grad.incl>
#include <Delay.incl>

define list <gradient> gl2={0.87 1.17 0.73 1.23 }

"FACTOR1=(d9/(p6*115.112))/2"
"l1=FACTOR1*2"

define loopcounter n
"n=cnst49"

"spoff29=0"
"spoff30=0"


"d11=30m"
"d49=cnst49*1u"


"acqt0=-p1*2/PI"


1 ze
2 30m
  20u BLKGRAD
  20u pl1:f1
  d1 pl1:f1
  20u UNBLKGRAD

  (p1 ph1):f1
```

```
;first encoding block 2*n chirp
3 d17
  (p41:sp41 ph6):f1 (p41:gp11)
  d17
  p17:gp13*gl2
  d16
  (p41:sp41 ph7):f1 (p41:gp12)
  d16
  p17:gp13*gl2
  igrad gl2
lo to 3 times n

; central selective 180 pulse
  d16
  p16:gp1
  d16
  p12:sp2:f1 ph2:r
  d16
  p16:gp1
  d16

;second encoding block 2*n chirp
 4 d17
  (p42:sp42 ph6):f1 (p42:gp11)
  d17
  p17:gp14*gl2
  d16
  (p42:sp42 ph7):f1 (p42:gp12)
  d16
  p17:gp14*gl2
  igrad gl2
  lo to 4 times n

;tocsy block
  d16 pl1:f1
  p1 ph3
  10u gron0
  (p32:sp29 ph3):f1
  20u groff
  d16 pl10:f1

                                          ;begin DIPSI2
5 p6*3.556 ph23
  p6*4.556 ph25
  p6*3.222 ph23
  p6*3.167 ph25
  p6*0.333 ph23
  p6*2.722 ph25
  p6*4.167 ph23
  p6*2.944 ph25
  p6*4.111 ph23

  p6*3.556 ph25
  p6*4.556 ph23
  p6*3.222 ph25
  p6*3.167 ph23
  p6*0.333 ph25
  p6*2.722 ph23
  p6*4.167 ph25
  p6*2.944 ph23
  p6*4.111 ph25

  p6*3.556 ph25
  p6*4.556 ph23
  p6*3.222 ph25
  p6*3.167 ph23
  p6*0.333 ph25
```

```
  p6*2.722 ph23
  p6*4.167 ph25
  p6*2.944 ph23
  p6*4.111 ph25

  p6*3.556 ph23
  p6*4.556 ph25
  p6*3.222 ph23
  p6*3.167 ph25
  p6*0.333 ph23
  p6*2.722 ph25
  p6*4.167 ph23
  p6*2.944 ph25
  p6*4.111 ph23
  lo to 5 times l1
                                        ;end DIPSI2

  p16:gp2
  d16
  10u gron0*1.333
  (p32*0.75:sp29 ph3):f1
  20u groff

  d16 pl1:f1
  p1 ph3

;acquisition
  go=2 ph31
  30m mc #0 to 2 F0(zd)
       20u BLKGRAD

d49

exit

ph1=0 2 0 2
ph2=0 0 1 1
ph3=0
ph6=0
ph7=0
ph23=3
ph25=1
ph31=0 2 2 0


;pl0 : 0W
;pl1 : f1 channel - power level for pulse (default)
;pl10: f1 channel - power level for TOCSY-spinlock
;sp2: f1 channel - shaped pulse
;sp29: f1 channel - shaped pulse (adiabatic)
;p1 : f1 channel -  90 degree high power pulse
;p6 : f1 channel -  90 degree low power pulse
;p12: f1 channel - 180 degree shaped pulse
;p16: homospoil/gradient pulse                         [1 msec]
;p32: f1 channel - 180 degree shaped pulse (adiabatic)    [20 msec]
;     smoothed chirp (sweepwidth, 20% smoothing, 10000 points)
;p41: adiabatic pulse
;p42: reverse sweep adiabatic pulse
;d1 : relaxation delay; 1-5 * T1
;d9 : TOCSY mixing time
;d16: delay for homospoil/gradient recovery
;d17: adjustable delay surrounding the encoding pulses, should be d16+p17
;cnst49: number of chirp pair per side of the sel180

;n: loop for GEMSTONE cycle (=cnst49)

;ns: even number, total number of scans: NS * TD0
;ds: 2
```

```
;phcor 2 : phasedifference between power levels sp1 and pl1

;for z-only gradients:
;gpz11: matched to BW
;gpz12: -gpz11

;gp0: grad TOCSY block
;gp2: grad TOCSY block

;use gradient files:
;gpnam11: RECT.1
;gpnam12: RECT.1


;for sweepwidth of adiabatic shape and adjusting gpz0
;   see supplementary material of M.J. Thrippleton & J. Keeler,
;   Angew. Chem. Int. Ed. 42, 3938-3941 (2003)
```

5.3 JND-GEMSTONE CLIP COSY for JEOL spectrometers

```
------------------------------------------------------------------------------
--                                                                          --
--                          Experiment Source Code                          --
--              Delta NMR Experiment & Machine Control Interface            --
--                                                                          --
--                     Copyright (c) 2021 JEOL Ltd                          --
--                            All Rights Reserved                           --
--                                                                          --
------------------------------------------------------------------------------
-- HELP.eng: GEMSTONE
-- Category: 1D, liquid_advanced, gemstone
-- File name : fsGEMSTONE_CLIP_COSY
--
-- Sequence name : kp_fsGEMSTONE_CLIP_COSY_16ph2
--
-- Reference :
--  P. Kiraly et al., Angew. Chem. Int. Ed. 2021, 60, 666-669. DOI:
10.1002/anie.202011642
-- P. Kiraly et al., Chem. Commun. , 2021, 57, 2368-2371. DOI: 10.1039/D0CC08033K
--  M. Bazzoni et al., Angew. Chem. Int. Ed. 2023, 62, e202314598. DOI:
10.1002/anie.202314598
-- D. Taylor et al., Chem. Commun., 2023, 59, 6734. DOI: 10.1039/d3cc01333b
--
-- Parameters
-- x_pulse          : 90[deg] pulse width
-- x_atn            : attenuator of x_pulse
--
-- obs_sel_180      : 180[deg] selective pulse
-- obs_sel_offset   : offset for obs_sel_180
-- obs_sel_atn      : attenuator of obs_sel_180
-- obs_sel_shape    : pulse shape of obs_sel_180
--
-- relaxation_delay : inter-pulse delay
-- repetition_time  : pulse repetition_time (= relaxation_delay+x_acq_time)
--
-- Note :
-- scans = 4*n
--
-- fs-GEMSTONE sequence with CLIP COSY transfer
--  x90-(ad180/+Ge  ad180/-Ge  ad180/+Ge  ad180/-Ge)loop  G1-sel180-G1  (ad180r/+Ge
ad180r/-Ge ad180r/+Ge ad180r/-Ge)loop- x90-zqs1-x90-d-x180-d-x90-d-x180-d-x90-zqs2-
x90-acq
--
--
```

```
--

header
    filename        => "fsGEMSTONE_CLIP_COSY";
    sample_id       => "";
    comment         => "fs-GEMSTONE_CLIP_COSY";
    process         =  "kp_basic1D.list";
    include "header";
end header;

instrument
    include "instrument";
    recvr_gain          => 40;
    spin_state          => "SPIN OFF";
end instrument;

acquisition
    x_domain         => "Proton";
    x_offset         => 1460.0[Hz];
    x_sweep          => 10[kHz];
    include "x_points_default_1d_calc";
    x_points        => 8192; --x_points_default;

    scans            => 4, help "scans = 4*n";
    x_prescans       => 4;
    mod_return       => 4;
    include "acquisition";
end acquisition;

pulse
    collect COMPLEX,OBS;

    experiment              =  "kp_fsGEMSTONE_CLIP_COSY_16ph2.jxp";
    obs_domain              =  x_domain;
    obs_offset              =  x_offset;
    obsfreq                 =  _get_freq(obs_domain);

comment_1               =? "*** Pulse ***";
    x_pulse                 => x90, help "observe 90[deg] pulse";
    x_atn                   =? xatn, help "attenuator for x_pulse";
    obs_pulse               =  x_pulse;
    obs_atn                 =  x_atn;


comment_16              =? "*** GEMSTONE: adiabatic 180 pulses ***";
    gemstone_selectivity      => 10[Hz], 0.5[Hz]->50[Hz]:0.001[Hz], help "bandwidth
of sinc excitation profile";
    gemstone_encoding        => 1/gemstone_selectivity, 20[ms] -> 2000[ms] : 1[ms],
help "total chemical shift encoding time";
    gs_target_pw              => 20[ms], (10[ms], 20[ms], 30[ms], 40[ms], 50[ms]),
help "adiabatic pulse duration desired, typically 20ms";

    gs_calc1                = gemstone_encoding / (8*gs_target_pw);
    gs_calc2                = lower(gs_calc1 + 0.001);
    gs_calc4                 = if gs_calc2=0 then gs_target_pw else gs_target_pw *
gs_calc1 / gs_calc2;

    gs_calc1m               = gemstone_encoding / (8 * 10[ms]);
    gs_calc2m               = lower(gs_calc1m + 0.001);
    gs_calc4m                = if gs_calc2m=0 then 10[ms] else 10[ms] * gs_calc1m /
gs_calc2m;

    gs_loop_number          =? if gs_calc1>=0.999 then gs_calc2 else gs_calc2m;
    gs_pulse_duration       =? if gs_calc1>=0.999 then
                                  if gs_calc4 >= gs_target_pw and gs_calc4 <=
2*gs_target_pw then gs_calc4 else gs_target_pw
                               else
```

```
                                    if  gs_calc4m  >=  9.999[ms]  and  gs_calc4m  <=
20.001[ms] then gs_calc4m else 10[ms];
    gs_encodingTotal         =? 8 * gs_pulse_duration * gs_loop_number;


    gs_loop                 => gs_loop_number, 0 -> 32 : 1, help "number of cycles";


    obs_ad180_shape           => "CHIRP", ad_shape_names, help "smoothed CHIRP 180
pulse shape";
    obs_ad180_m_pulse         => gs_pulse_duration, 10[ms] -> 100[ms] : 1[ms], help
"adiabatic pulse duration = gs_encodingTotal / (8*gs_loop_number)";
    obs_ad180_m_fsweep       => 5[kHz], help "adiabatic pulse sweep width";
    obs_ad180_sweep_ppm      =  obs_ad180_m_fsweep / obsfreq * 1[Mppm];
    obs_ad180_chirp_smooth   => 10[%], 0[%] -> 50[%] : 1[%], help "smoothing";
    obs_ad180_m_q0           => 11, 3 -> 20 : 1, help "adiabaticity factor typically
11";
    obs_ad180_m_ph_rev         =  1;
    obs_ad180_b1_calc        =  sqrt (obs_ad180_m_q0 * obs_ad180_m_fsweep / (6.28319
* obs_ad180_m_pulse));
    obs_ad180_m_b1max          => obs_ad180_b1_calc;
    obs_ad180_pw90           =  1 / (4 * obs_ad180_m_b1max);
    obs_ad180_atn_calc       =  obs_atn + 20[dB] * log (obs_ad180_pw90 / obs_pulse);
    obs_ad180_atn            => obs_ad180_atn_calc;

    obs_ad180r_m_fsweep        =  obs_ad180_m_fsweep;
    obs_ad180r_m_pulse         =  obs_ad180_m_pulse;

    obs_ad180r_chirp_smooth  =  obs_ad180_chirp_smooth;
    obs_ad180r_m_q0            =  obs_ad180_m_q0;
    obs_ad180r_m_ph_rev        = -1, (1, -1), help "phase reverse";
    obs_ad180r_atn           =  obs_ad180_atn;


comment_17               =? "*** GEMSTONE: gradients ***";
    grad_slPos_amp             => -6.0[mT/m], help "Amplitude of spatial encoding
gradient during 1st chirp";
    grad_slTweak                => 1.0, 0.0 -> 2.0 : 0.001, help "tweak gradient
amplitude";
    grad_slNeg_amp           =? -1 * grad_slPos_amp * grad_slTweak, help "Amplitude
of spatial encoding gradient during 2nd chirp";

    grad_recover_sl          => 1[ms], help "gradient recovery time in loop";
    grad_sl                  =? obs_ad180_m_pulse;
    grad_shape_sl          =? "SQUARE", help "Gradient shape for spatial encoding";

    grad_ctp                 => TRUE, help "Use optional CTP gradient pulses";
    grad_ctp_sl             => 1.0[ms], help "optional CTP gradient pulse durations";
    grad_ctp_amp1            => 94[mT/m], help "CTP gradient pulse amplitude in the
loop";
    grad_ctp_amp2            => 59[mT/m], help "CTP gradient pulse amplitude in the
loop";
    grad_ctp_amp3           => 112[mT/m], help "CTP gradient pulse amplitude in the
loop";
    grad_ctp_amp4            => 79[mT/m], help "CTP gradient pulse amplitude in the
loop";
      grad_ctp_amp1n =? -grad_ctp_amp1;
      grad_ctp_amp2n =? -grad_ctp_amp2;
      grad_ctp_amp3n =? -grad_ctp_amp3;
      grad_ctp_amp4n =? -grad_ctp_amp4;

comment_18              =? "*** GEMSTONE: Active Spin Refocusing 180 Pulse ***";
    obs_asr180_shape         => "RSNOB_180", std_shape_names, help "selective pulse
shape";
    soft_shape_input         =  obs_asr180_shape;

    --include "soft_pulse_calc";
```

```
    soft_bw_input                => 100[Hz], help "bandwidth in [ppm] or [Hz] of
obs_sel_180";
    xfreq                    =  _get_freq(x_domain);
    soft_angle               =? _get_s_mparam(soft_shape_input, "m_angle", 90[deg]);
    int_soft_pc              =  _get_s_mparam(soft_shape_input, "m_integ", 100[%]);
    int_r_soft_pc             =  int_soft_pc / _get_s_mparam("gauss_90", "m_integ",
100[%]);
    unit_soft_pc             =  if soft_bw_input * 0 = 0[ppm]
                                then 1
                                else if soft_bw_input * 0 = 0[Hz]
                                then 2
                                else 3;
    soft_bw_hz               =  if unit_soft_pc = 1
                                then soft_bw_input * xfreq / 1[Mppm]
                                else if unit_soft_pc = 2
                                then soft_bw_input
                                else 100[Hz];
    type_soft_pc             =  _get_s_mparam(soft_shape_input, "m_type", "std");
    is_std_pc                =  if type_soft_pc = "std" and unit_soft_pc < 3
                                then TRUE
                                else FALSE;
    soft_pw_calc             =? if is_std_pc
                                then  _get_s_mparam(soft_shape_input,  "bw",  1)  /
soft_bw_hz
                                else 1[us];
    soft_atn_calc            =? if is_std_pc
                                then xatn_soft + 20[dB] * log (soft_pw_calc / x90_soft
* 90[deg] / soft_angle * int_r_soft_pc)
                                else 79[dB];

    obs_asr180               => soft_pw_calc, help "selective 180 pulse duration";
    obs_asr180_atn            => soft_atn_calc, help "attenuator of selective pulse,
obs_asr_180";

comment_8                =? "*** Pulse Field Gradient ***";
    gradient_max              =? z_gradient_max, help "Maximum amplitude for a given
probe as defined in the probe file";
    grad_1                   => 1[ms], help "CTP gradient pulse duration";
    grad_1_amp               => 160[mT/m], help "CTP gradient pulse amplitude";
       grad_1_ampn =? -grad_1_amp;
    grad_shape                => "S_RECTANGLE", fg_shape_names, help "shape type of
CTP gradient pulses";
    grad_recover             => 1.0[ms], help "gradient recovery time";

    grad_spoil               => 1[ms], help "duration of grad_1 during mixing";
    grad_spoil_amp              => 133[mT/m], help "Enter amplitude in Tesla/meter
units";
    grad_spoiln_amp =? -grad_spoil_amp;
    grad_recover_zqs         => 3.0[ms], help "gradient recovery time";

comment_50               =? "*** CLIP-COSY ***";
    delta                    => 20[ms], help "transfer delay for CLIP-COSY";

comment_41               =? "*** Zero-Quantum Dephasing ***";
    obs_chp_shape            =? "CHIRP", ad_shape_names, help "shape pulse";
    obs_chp1_m_pulse           => 50[ms], (30[ms], 40[ms], 50[ms]), help "ZQfilter
pulse width";
    obs_chp1_m_fsweep        =? if obs_chp1_m_pulse = 30[ms]
                                then x_sweep * 2
                                else if obs_chp1_m_pulse = 40[ms]
                                then x_sweep * 2
                                else x_sweep * 2;
    obs_chp1_sweep_ppm       =? round (obs_chp1_m_fsweep * 1[Mppm] / obsfreq), help
"obs_chp1_ad_sweep_ppm";
    obs_chp1_chirp_smooth    =  10[%];
    obs_chp1_m_q0              =  11, help "q >= 5 On resonance adiabatic Quality
factor";
    obs_chp1_m_ph_rev        =  1, (1, -1), help "phase reverse";
```

```
    obs_chp1_b1_calc         =  sqrt (obs_chp1_m_q0 * obs_chp1_m_fsweep / (6.28319 *
obs_chp1_m_pulse));
    obs_chp1_m_b1max         =? obs_chp1_b1_calc;
    obs_chp1_pw90            =  1 / (4 * obs_chp1_m_b1max);
    obs_chp1_atn_calc        =  if obs_atn + 20[dB] * log (obs_chp1_pw90 / obs_pulse)
< obs_atn + 6[dB]
                                then obs_atn + 6[dB]
                                else  obs_atn  +  20[dB]  *  log  (obs_chp1_pw90  /
obs_pulse);
    obs_chp1_atn             =? obs_chp1_atn_calc;
    obs_chp2_m_pulse           => 30[ms], (30[ms], 40[ms], 50[ms]), help "ZQfilter
pulse width";
    obs_chp2_m_fsweep        =? if obs_chp2_m_pulse = 30[ms]
                                then x_sweep * 2
                                else if obs_chp2_m_pulse = 40[ms]
                                then x_sweep * 2
                                else x_sweep * 2;
    obs_chp2_sweep_ppm       =? round (obs_chp2_m_fsweep * 1[Mppm] / obsfreq), help
"obs_chirp2_ad_sweep_ppm";
    obs_chp2_chirp_smooth    =  10[%];
    obs_chp2_m_q0              =  11, help "q >= 5 On resonance adiabatic Quality
factor";
    obs_chp2_m_ph_rev        =  1, (1, -1), help "phase reverse";
    obs_chp2_b1_calc         =  sqrt (obs_chp2_m_q0 * obs_chp2_m_fsweep / (6.28319 *
obs_chp2_m_pulse));
    obs_chp2_m_b1max         =? obs_chp2_b1_calc;
    obs_chp2_pw90            =  1 / (4 * obs_chp2_m_b1max);
    obs_chp2_atn_calc        =  if obs_atn + 20[dB] * log (obs_chp2_pw90 / obs_pulse)
< obs_atn + 6[dB]
                                then obs_atn + 6[dB]
                                else  obs_atn  +  20[dB]  *  log  (obs_chp2_pw90  /
obs_pulse);
    obs_chp2_atn             =? obs_chp2_atn_calc;
    g_factor                 =  1, help "obs_gamma/1H_gamma";
    coil_factor              =  1, help "coil_length/20mm";
    f_factor_zqf1                    =   obs_chp1_m_fsweep  /  40[kHz],  help  "ref
10ppm*8/500MHz=40kHz";
    grad_zqf1_amp_calc       =  47[mT/m] * f_factor_zqf1 / coil_factor / g_factor,
help "gradient amplitude of grad_sl_ps";
    grad_zqf1_amp                =? grad_zqf1_amp_calc, help "Enter amplitude in
Tesla/meter units";
    f_factor_zqf2                    =   obs_chp2_m_fsweep  /  40[kHz],  help  "ref
10ppm*8/500MHz=40kHz";
    grad_zqf2_amp_calc       =  47[mT/m] * f_factor_zqf2 / coil_factor / g_factor,
help "gradient amplitude of grad_sl_ps";
    grad_zqf2_amp                =? grad_zqf2_amp_calc, help "Enter amplitude in
Tesla/meter units";
    grad_zqf1n_amp =? -grad_zqf1_amp;
    grad_zqf2n_amp =? -grad_zqf2_amp;

comment_7 =? "*** Pulse Delay ***";
    initial_wait     =  1[s];
    include "relaxation_delay_1d_calc";
    relaxation_delay => 2[s]; -- relaxation_delay_default, help "relaxation delay";
    repetition_time          =?     relaxation_delay     +     x_acq_time,     help
"relaxation_delay+x_acq_time";

comment_900              =? "*** lock hold ***";
    lock_hold                => TRUE, help "select TRUE or FALSE for lock hold";

    include "pulse";

    -- not optimized, taken from ref
    phase_90  = {16(0), 16(180)};
    phase_asr = {4(0), 4(90), 4(180), 4(270)};
    phase_1 = {2(0), 2(180)};
    phase_2 = {0, 180};
    phase_zero = {0};
```

```
    phase_one = {90};
    --phase_two = {180};
    phase_three = {270};
    --scans      1   2
    phase_acq = {2(0, 180, 180, 0, 180, 0, 0, 180), 2(180, 0, 0, 180, 0, 180, 180,
0)};

    phase_ad1 = {0};
    phase_ad2 = {0}; -- {16(0),16(90)};
    phase_ad3 = {0};
    phase_ad4 = {0}; -- {4(0), 4(90)};

    phase_ad1r= {0};
    phase_ad2r= {0}; -- {8(0),8(90)};
    phase_ad3r= {0};
    phase_ad4r= {0}; -- {2(0), 2(90)};

module_config = "";

begin
    initial_wait;
    relaxation_delay;

    when lock_hold do
        on LOCKHOLD;
    end when;
    1[us];

    x_pulse, (obs.gate, obs.phs.phase_90, obs.atn.x_atn);

    loop gs_loop times
        parallel
            justify center
                obs_ad180_m_pulse,(obs.gate,obs.phs.phase_ad1,obs.atn.obs_ad180_atn,
obs.shape.{obs_ad180_shape,"obs_ad180"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slPos_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;

        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp1, fgz.shape.grad_shape);
            grad_recover;
        end when;
        parallel
            justify center
                obs_ad180_m_pulse,(obs.gate,obs.phs.phase_ad2,obs.atn.obs_ad180_atn,
obs.shape.{obs_ad180_shape,"obs_ad180"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slNeg_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp1, fgz.shape.grad_shape);
            grad_recover;
        end when;

        parallel
            justify center
                obs_ad180_m_pulse,(obs.gate,obs.phs.phase_ad3,obs.atn.obs_ad180_atn,
obs.shape.{obs_ad180_shape,"obs_ad180"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slPos_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;
```

```
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp2, fgz.shape.grad_shape);
            grad_recover;
        end when;
        parallel
            justify center
                obs_ad180_m_pulse,(obs.gate,obs.phs.phase_ad4,obs.atn.obs_ad180_atn,
obs.shape.{obs_ad180_shape,"obs_ad180"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slNeg_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp2, fgz.shape.grad_shape);
            grad_recover;
        end when;
    end loop;

    grad_1, (fgz.gate, fgz.amp.grad_1_amp, fgz.shape.grad_shape);
    grad_recover;
    obs_asr180,      (obs.gate,      obs.phs.phase_asr,      obs.atn.obs_asr180_atn,
obs.shape.obs_asr180_shape);
    grad_1, (fgz.gate, fgz.amp.grad_1_amp, fgz.shape.grad_shape);
    grad_recover;

    loop gs_loop times
        parallel
            justify center
                obs_ad180r_m_pulse,          (obs.gate,          obs.phs.phase_ad1r,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180r"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slPos_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;

        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp3, fgz.shape.grad_shape);
            grad_recover;
        end when;
        parallel
            justify center
                obs_ad180r_m_pulse,          (obs.gate,          obs.phs.phase_ad2r,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180r"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slNeg_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp3, fgz.shape.grad_shape);
            grad_recover;
        end when;

        parallel
            justify center
                obs_ad180r_m_pulse,          (obs.gate,          obs.phs.phase_ad3r,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180r"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slPos_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;

        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp4, fgz.shape.grad_shape);
```

```
            grad_recover;
        end when;
        parallel
            justify center
                obs_ad180r_m_pulse,          (obs.gate,          obs.phs.phase_ad4r,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180r"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slNeg_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp4, fgz.shape.grad_shape);
            grad_recover;
        end when;
    end loop;

    x_pulse, (obs.gate, obs.phs.phase_1, obs.atn.x_atn);
    10[us];
    on (FGZ.GATE, FGZ.AMP.grad_zqf1_amp);
    obs_chp1_m_pulse,    (OBS.GATE,    OBS.PHS.phase_zero,    OBS.ATN.obs_chp1_atn,
OBS.SHAPE.{obs_chp_shape, "obs_chp1"});
    off (FGZ.GATE, FGZ.AMP.grad_zqf1_amp);
    grad_recover;

    x_pulse, (obs.gate, obs.phs.phase_zero, obs.atn.x_atn);
    delta / 2;
    2 * x_pulse, (OBS.GATE, OBS.PHS.phase_one, OBS.ATN.x_atn);
    delta / 2;
    x_pulse, (OBS.GATE, OBS.PHS.phase_one, OBS.ATN.x_atn);
    delta / 2;
    2 * x_pulse, (OBS.GATE, OBS.PHS.phase_three, OBS.ATN.x_atn);
    delta / 2;
    x_pulse, (OBS.GATE, OBS.PHS.phase_zero, OBS.ATN.x_atn);

    grad_spoil, (fgz.gate, fgz.amp.grad_spoil_amp, fgz.shape.grad_shape);
    grad_recover;
    on (FGZ.GATE, FGZ.AMP.grad_zqf2_amp);
    obs_chp2_m_pulse,    (OBS.GATE,    OBS.PHS.phase_zero,    OBS.ATN.obs_chp2_atn,
OBS.SHAPE.{obs_chp_shape, "obs_chp2"});
    off (FGZ.GATE, FGZ.AMP.grad_zqf2_amp);
    grad_recover;
    grad_spoil, (fgz.gate, fgz.amp.grad_spoil_amp*0.77, fgz.shape.grad_shape);
    grad_recover;

    grad_recover_zqs;
    x_pulse, (obs.gate, obs.phs.phase_2, obs.atn.x_atn);

    acq( dead_time, delay, phase_acq );
    10[us];
    when lock_hold do
        off LOCKHOLD;
    end when;
    10[us];

       relaxation_delay;

    when lock_hold do
        on LOCKHOLD;
    end when;
    1[us];

    x_pulse, (obs.gate, obs.phs.phase_90, obs.atn.x_atn);

    loop gs_loop times
        parallel
            justify center
```

```
                obs_ad180_m_pulse,(obs.gate,obs.phs.phase_ad1,obs.atn.obs_ad180_atn,
obs.shape.{obs_ad180_shape,"obs_ad180"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slNeg_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;

        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp1n, fgz.shape.grad_shape);
            grad_recover;
        end when;
        parallel
            justify center
                obs_ad180_m_pulse,(obs.gate,obs.phs.phase_ad2,obs.atn.obs_ad180_atn,
obs.shape.{obs_ad180_shape,"obs_ad180"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slPos_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp1n, fgz.shape.grad_shape);
            grad_recover;
        end when;

        parallel
            justify center
                obs_ad180_m_pulse,(obs.gate,obs.phs.phase_ad3,obs.atn.obs_ad180_atn,
obs.shape.{obs_ad180_shape,"obs_ad180"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slNeg_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;

        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp2n, fgz.shape.grad_shape);
            grad_recover;
        end when;
        parallel
            justify center
                obs_ad180_m_pulse,(obs.gate,obs.phs.phase_ad4,obs.atn.obs_ad180_atn,
obs.shape.{obs_ad180_shape,"obs_ad180"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slPos_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp2n, fgz.shape.grad_shape);
            grad_recover;
        end when;
    end loop;

    grad_1, (fgz.gate, fgz.amp.grad_1_amp, fgz.shape.grad_shape);
    grad_recover;
    obs_asr180,      (obs.gate,      obs.phs.phase_asr,      obs.atn.obs_asr180_atn,
obs.shape.obs_asr180_shape);
    grad_1, (fgz.gate, fgz.amp.grad_1_amp, fgz.shape.grad_shape);
    grad_recover;

    loop gs_loop times
        parallel
            justify center
                obs_ad180r_m_pulse,          (obs.gate,          obs.phs.phase_ad1r,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180r"});
            justify center
```

```
                grad_sl, (fgz.gate, fgz.amp.grad_slNeg_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;

        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp3n, fgz.shape.grad_shape);
            grad_recover;
        end when;
        parallel
            justify center
                obs_ad180r_m_pulse,          (obs.gate,          obs.phs.phase_ad2r,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180r"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slPos_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp3n, fgz.shape.grad_shape);
            grad_recover;
        end when;

        parallel
            justify center
                obs_ad180r_m_pulse,          (obs.gate,          obs.phs.phase_ad3r,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180r"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slNeg_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;

        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp4n, fgz.shape.grad_shape);
            grad_recover;
        end when;
        parallel
            justify center
                obs_ad180r_m_pulse,          (obs.gate,          obs.phs.phase_ad4r,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180r"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slPos_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp4n, fgz.shape.grad_shape);
            grad_recover;
        end when;
    end loop;

    x_pulse, (obs.gate, obs.phs.phase_1, obs.atn.x_atn);
    10[us];
    on (FGZ.GATE, FGZ.AMP.grad_zqf1n_amp);
    obs_chp1_m_pulse,    (OBS.GATE,    OBS.PHS.phase_zero,    OBS.ATN.obs_chp1_atn,
OBS.SHAPE.{obs_chp_shape, "obs_chp1"});
    off (FGZ.GATE, FGZ.AMP.grad_zqf1n_amp);
    grad_recover;

    x_pulse, (obs.gate, obs.phs.phase_zero, obs.atn.x_atn);
    delta / 2;
    2 * x_pulse, (OBS.GATE, OBS.PHS.phase_one, OBS.ATN.x_atn);
    delta / 2;
    x_pulse, (OBS.GATE, OBS.PHS.phase_one, OBS.ATN.x_atn);
    delta / 2;
    2 * x_pulse, (OBS.GATE, OBS.PHS.phase_three, OBS.ATN.x_atn);
    delta / 2;
```

```
    x_pulse, (OBS.GATE, OBS.PHS.phase_zero, OBS.ATN.x_atn);

    grad_spoil, (fgz.gate, fgz.amp.grad_spoiln_amp, fgz.shape.grad_shape);
    grad_recover;
    on (FGZ.GATE, FGZ.AMP.grad_zqf2n_amp);
    obs_chp2_m_pulse,    (OBS.GATE,    OBS.PHS.phase_zero,    OBS.ATN.obs_chp2_atn,
OBS.SHAPE.{obs_chp_shape, "obs_chp2"});
    off (FGZ.GATE, FGZ.AMP.grad_zqf2n_amp);
    grad_recover;
    grad_spoil, (fgz.gate, fgz.amp.grad_spoiln_amp*0.77, fgz.shape.grad_shape);
    grad_recover;

    grad_recover_zqs;
    x_pulse, (obs.gate, obs.phs.phase_2, obs.atn.x_atn);

    acq( dead_time, delay, phase_acq );
    10[us];
    when lock_hold do
        off LOCKHOLD;
    end when;
    10[us];

end pulse;
```

5.4 JND-GEMSTONE TOCSY for JEOL spectrometers

```
-------------------------------------------------------------------------------
--                                                                           --
--                          Experiment Source Code                           --
--              Delta NMR Experiment & Machine Control Interface             --
--                                                                           --
--                     Copyright (c) 2021 JEOL Ltd                           --
--                           All Rights Reserved                             --
--                                                                           --
-------------------------------------------------------------------------------
-- HELP.eng: GEMSTONE
-- Category: 1D, liquid_advanced, gemstone
-- File name : fsGEMSTONE_TOCSY
--
-- Sequence name : kp_fsGEMSTONE_TOCSY_16ph32
--
-- Reference :
--  P. Kiraly et al., Angew. Chem. Int. Ed. 2021, 60, 666-669. DOI:
10.1002/anie.202011642
-- P. Kiraly et al., Chem. Commun. , 2021, 57, 2368-2371. DOI: 10.1039/D0CC08033K
--  M. Bazzoni et al., Angew. Chem. Int. Ed. 2023, 62, e202314598. DOI:
10.1002/anie.202314598
--
-- Parameters
-- x_pulse          : 90[deg] pulse width
-- x_atn            : attenuator of x_pulse
--
-- obs_sel_180      : 180[deg] selective pulse
-- obs_sel_offset   : offset for obs_sel_180
-- obs_sel_atn      : attenuator of obs_sel_180
-- obs_sel_shape    : pulse shape of obs_sel_180
--
-- relaxation_delay : inter-pulse delay
-- repetition_time  : pulse repetition_time (= relaxation_delay+x_acq_time)
--
-- Note :
-- scans = 1*n
--
-- fs-GEMSTONE sequence
```

```
--  x90-(ad180/+Ge  ad180/-Ge  ad180/+Ge  ad180/-Ge)loop  G1-sel180-G1  (ad180r/+Ge
ad180r/-Ge ad180r/+Ge ad180r/-Ge)loop- x90-mix_time(tocsy)-x90-acq
--
--
--

header
    filename        => "fsGEMSTONE_TOCSY";
    sample_id       => "";
    comment         => "fs-GEMSTONE_TOCSY";
    process         =  "kp_basic1D.list";
    include "header";
end header;

instrument
    include "instrument";
    recvr_gain          => 40;
    spin_state          => "SPIN OFF";
end instrument;

acquisition
    x_domain         => "Proton";
    x_offset         => 1460.0[Hz];
    x_sweep          => 10[kHz];
    include "x_points_default_1d_calc";
    x_points        => 8192; --x_points_default;

    scans            => 8, help "scans = 8*n";
    x_prescans       => 8;
    mod_return       => 8;
    include "acquisition";
end acquisition;

pulse
    collect COMPLEX,OBS;

    macro dipsi2_rc( pulse_90, phase, sl_atn, tau ) is
        on OBS.ATN.sl_atn;
        pulse_90 * 180 / 90, (OBS.GATE, OBS.PHS.phase);
        tau;
        pulse_90 * 140 / 90, (OBS.GATE, OBS.PHS.phase);
        pulse_90 * 320 / 90, (OBS.GATE, OBS.PHS.phase + 180);
        tau;
        pulse_90 * 90 / 90, (OBS.GATE, OBS.PHS.phase + 180);
        pulse_90 * 270 / 90, (OBS.GATE, OBS.PHS.phase);
        tau;
        pulse_90 * 20 / 90, (OBS.GATE, OBS.PHS.phase);
        pulse_90 * 200 / 90, (OBS.GATE, OBS.PHS.phase + 180);
        tau;
        pulse_90 * 85 / 90, (OBS.GATE, OBS.PHS.phase + 180);
        pulse_90 * 30 / 90, (OBS.GATE, OBS.PHS.phase);
        pulse_90 * 125 / 90, (OBS.GATE, OBS.PHS.phase + 180);
        tau;
        pulse_90 * 120 / 90, (OBS.GATE, OBS.PHS.phase + 180);
        pulse_90 * 300 / 90, (OBS.GATE, OBS.PHS.phase);
        tau;
        pulse_90 * 75 / 90, (OBS.GATE, OBS.PHS.phase);
        pulse_90 * 255 / 90, (OBS.GATE, OBS.PHS.phase + 180);
        tau;
        pulse_90 * 10 / 90, (OBS.GATE, OBS.PHS.phase + 180);
        pulse_90 * 190 / 90, (OBS.GATE, OBS.PHS.phase);
        tau;
        pulse_90 * 180 / 90, (OBS.GATE, OBS.PHS.phase);
        loop 2 times
            tau;
            pulse_90 * 180 / 90, (OBS.GATE, OBS.PHS.phase + 180);
            tau;
            pulse_90 * 140 / 90, (OBS.GATE, OBS.PHS.phase + 180);
```

```
            pulse_90 * 320 / 90, (OBS.GATE, OBS.PHS.phase);
            tau;
            pulse_90 * 90 / 90, (OBS.GATE, OBS.PHS.phase);
            pulse_90 * 270 / 90, (OBS.GATE, OBS.PHS.phase + 180);
            tau;
            pulse_90 * 20 / 90, (OBS.GATE, OBS.PHS.phase + 180);
            pulse_90 * 200 / 90, (OBS.GATE, OBS.PHS.phase);
            tau;
            pulse_90 * 85 / 90, (OBS.GATE, OBS.PHS.phase);
            pulse_90 * 30 / 90, (OBS.GATE, OBS.PHS.phase + 180);
            pulse_90 * 125 / 90, (OBS.GATE, OBS.PHS.phase);
            tau;
            pulse_90 * 120 / 90, (OBS.GATE, OBS.PHS.phase);
            pulse_90 * 300 / 90, (OBS.GATE, OBS.PHS.phase + 180);
            tau;
            pulse_90 * 75 / 90, (OBS.GATE, OBS.PHS.phase + 180);
            pulse_90 * 255 / 90, (OBS.GATE, OBS.PHS.phase);
            tau;
            pulse_90 * 10 / 90, (OBS.GATE, OBS.PHS.phase);
            pulse_90 * 190 / 90, (OBS.GATE, OBS.PHS.phase + 180);
            tau;
            pulse_90 * 180 / 90, (OBS.GATE, OBS.PHS.phase + 180);
        end loop;
        tau;
        pulse_90 * 180 / 90, (OBS.GATE, OBS.PHS.phase);
        tau;
        pulse_90 * 140 / 90, (OBS.GATE, OBS.PHS.phase);
        pulse_90 * 320 / 90, (OBS.GATE, OBS.PHS.phase + 180);
        tau;
        pulse_90 * 90 / 90, (OBS.GATE, OBS.PHS.phase + 180);
        pulse_90 * 270 / 90, (OBS.GATE, OBS.PHS.phase);
        tau;
        pulse_90 * 20 / 90, (OBS.GATE, OBS.PHS.phase);
        pulse_90 * 200 / 90, (OBS.GATE, OBS.PHS.phase + 180);
        tau;
        pulse_90 * 85 / 90, (OBS.GATE, OBS.PHS.phase + 180);
        pulse_90 * 30 / 90, (OBS.GATE, OBS.PHS.phase);
        pulse_90 * 125 / 90, (OBS.GATE, OBS.PHS.phase + 180);
        tau;
        pulse_90 * 120 / 90, (OBS.GATE, OBS.PHS.phase + 180);
        pulse_90 * 300 / 90, (OBS.GATE, OBS.PHS.phase);
        tau;
        pulse_90 * 75 / 90, (OBS.GATE, OBS.PHS.phase);
        pulse_90 * 255 / 90, (OBS.GATE, OBS.PHS.phase + 180);
        tau;
        pulse_90 * 10 / 90, (OBS.GATE, OBS.PHS.phase + 180);
        pulse_90 * 190 / 90, (OBS.GATE, OBS.PHS.phase);
        tau;
        pulse_90 * 180 / 90, (OBS.GATE, OBS.PHS.phase);
        tau;
        off OBS.ATN.sl_atn;
    end dipsi2_rc;

    experiment              =  "kp_fsGEMSTONE_TOCSY_16ph32.jxp";
    obs_domain              =  x_domain;
    obs_offset              =  x_offset;
    obsfreq                 =  _get_freq(obs_domain);

comment_1               =? "*** Pulse ***";
    x_pulse                 => x90, help "observe 90[deg] pulse";
    x_atn                   =? xatn, help "attenuator for x_pulse";
    obs_pulse               =  x_pulse;
    obs_atn                 =  x_atn;


comment_16              =? "*** GEMSTONE: adiabatic 180 pulses ***";
    gemstone_selectivity     => 10[Hz], 0.5[Hz]->50[Hz]:0.001[Hz], help "bandwidth
of sinc excitation profile";
```

```
    gemstone_encoding        => 1/gemstone_selectivity, 20[ms] -> 2000[ms] : 1[ms],
help "total chemical shift encoding time";
    gs_target_pw              => 20[ms], (10[ms], 20[ms], 30[ms], 40[ms], 50[ms]),
help "adiabatic pulse duration desired, typically 20ms";

    gs_calc1                 = gemstone_encoding / (8*gs_target_pw);
    gs_calc2                 = lower(gs_calc1 + 0.001);
    gs_calc4                  = if gs_calc2=0 then gs_target_pw else gs_target_pw *
gs_calc1 / gs_calc2;

    gs_calc1m                = gemstone_encoding / (8 * 10[ms]);
    gs_calc2m                = lower(gs_calc1m + 0.001);
    gs_calc4m                = if gs_calc2m=0 then 10[ms] else 10[ms] * gs_calc1m /
gs_calc2m;

    gs_loop_number           =? if gs_calc1>=0.999 then gs_calc2 else gs_calc2m;
    gs_pulse_duration        =? if gs_calc1>=0.999 then
                                   if gs_calc4 >= gs_target_pw and gs_calc4 <=
2*gs_target_pw then gs_calc4 else gs_target_pw
                                else
                                   if gs_calc4m >= 9.999[ms] and gs_calc4m <=
20.001[ms] then gs_calc4m else 10[ms];
    gs_encodingTotal         =? 8 * gs_pulse_duration * gs_loop_number;


    gs_loop                 => gs_loop_number, 0 -> 32 : 1, help "number of cycles";


    obs_ad180_shape           => "CHIRP", ad_shape_names, help "smoothed CHIRP 180
pulse shape";
    obs_ad180_m_pulse         => gs_pulse_duration, 10[ms] -> 100[ms] : 1[ms], help
"adiabatic pulse duration = gs_encodingTotal / (8*gs_loop_number)";
    obs_ad180_m_fsweep       => 5[kHz], help "adiabatic pulse sweep width";
    obs_ad180_sweep_ppm      =  obs_ad180_m_fsweep / obsfreq * 1[Mppm];
    obs_ad180_chirp_smooth   => 10[%], 0[%] -> 50[%] : 1[%], help "smoothing";
    obs_ad180_m_q0            => 11, 3 -> 20 : 1, help "adiabaticity factor typically
11";
    obs_ad180_m_ph_rev         =  1;
    obs_ad180_b1_calc        =  sqrt (obs_ad180_m_q0 * obs_ad180_m_fsweep / (6.28319
* obs_ad180_m_pulse));
    obs_ad180_m_b1max          => obs_ad180_b1_calc;
    obs_ad180_pw90           =  1 / (4 * obs_ad180_m_b1max);
    obs_ad180_atn_calc       =  obs_atn + 20[dB] * log (obs_ad180_pw90 / obs_pulse);
    obs_ad180_atn            => obs_ad180_atn_calc;

    obs_ad180r_m_fsweep        =  obs_ad180_m_fsweep;
    obs_ad180r_m_pulse         =  obs_ad180_m_pulse;

    obs_ad180r_chirp_smooth  =  obs_ad180_chirp_smooth;
    obs_ad180r_m_q0            =  obs_ad180_m_q0;
    obs_ad180r_m_ph_rev        = -1, (1, -1), help "phase reverse";
    obs_ad180r_atn           =  obs_ad180_atn;


comment_17               =? "*** GEMSTONE: gradients ***";
    grad_slPos_amp              => -6.0[mT/m], help "Amplitude of spatial encoding
gradient during 1st chirp";
    grad_slTweak                 => 1.0, 0.0 -> 2.0 : 0.001, help "tweak gradient
amplitude";
    grad_slNeg_amp           =? -1 * grad_slPos_amp * grad_slTweak, help "Amplitude
of spatial encoding gradient during 2nd chirp";

    grad_recover_sl          => 1[ms], help "gradient recovery time in loop";
    grad_sl                  =? obs_ad180_m_pulse;
    grad_shape_sl          =? "SQUARE", help "Gradient shape for spatial encoding";

    grad_ctp                 => TRUE, help "Use optional CTP gradient pulses";
    grad_ctp_sl            => 1.0[ms], help "optional CTP gradient pulse durations";
```

```
    grad_ctp_amp1            => 94[mT/m], help "CTP gradient pulse amplitude in the
loop";
    grad_ctp_amp2            => 59[mT/m], help "CTP gradient pulse amplitude in the
loop";
    grad_ctp_amp3           => 112[mT/m], help "CTP gradient pulse amplitude in the
loop";
    grad_ctp_amp4            => 79[mT/m], help "CTP gradient pulse amplitude in the
loop";
      grad_ctp_amp1n =? -grad_ctp_amp1;
      grad_ctp_amp2n =? -grad_ctp_amp2;
      grad_ctp_amp3n =? -grad_ctp_amp3;
      grad_ctp_amp4n =? -grad_ctp_amp4;

comment_18              =? "*** GEMSTONE: Active Spin Refocusing 180 Pulse ***";
    obs_asr180_shape         => "RSNOB_180", std_shape_names, help "selective pulse
shape";
    soft_shape_input         =  obs_asr180_shape;

    --include "soft_pulse_calc";
    soft_bw_input                => 100[Hz], help "bandwidth in [ppm] or [Hz] of
obs_sel_180";
    xfreq                    =  _get_freq(x_domain);
    soft_angle              =? _get_s_mparam(soft_shape_input, "m_angle", 90[deg]);
    int_soft_pc              =  _get_s_mparam(soft_shape_input, "m_integ", 100[%]);
    int_r_soft_pc             =  int_soft_pc / _get_s_mparam("gauss_90", "m_integ",
100[%]);
    unit_soft_pc             =  if soft_bw_input * 0 = 0[ppm]
                                then 1
                                else if soft_bw_input * 0 = 0[Hz]
                                then 2
                                else 3;
    soft_bw_hz               =  if unit_soft_pc = 1
                                then soft_bw_input * xfreq / 1[Mppm]
                                else if unit_soft_pc = 2
                                then soft_bw_input
                                else 100[Hz];
    type_soft_pc             =  _get_s_mparam(soft_shape_input, "m_type", "std");
    is_std_pc                =  if type_soft_pc = "std" and unit_soft_pc < 3
                                then TRUE
                                else FALSE;
    soft_pw_calc             =? if is_std_pc
                                then  _get_s_mparam(soft_shape_input,  "bw",  1)  /
soft_bw_hz
                                else 1[us];
    soft_atn_calc            =? if is_std_pc
                                then xatn_soft + 20[dB] * log (soft_pw_calc / x90_soft
* 90[deg] / soft_angle * int_r_soft_pc)
                                else 79[dB];

    obs_asr180               => soft_pw_calc, help "selective 180 pulse duration";
    obs_asr180_atn            => soft_atn_calc, help "attenuator of selective pulse,
obs_asr_180";

comment_8               =? "*** Pulse Field Gradient ***";
    gradient_max              =? z_gradient_max, help "Maximum amplitude for a given
probe as defined in the probe file";
    grad_1                   => 1[ms], help "CTP gradient pulse duration";
    grad_1_amp               => 160[mT/m], help "CTP gradient pulse amplitude";
       grad_1_ampn =? -grad_1_amp;
    grad_shape                => "S_RECTANGLE", fg_shape_names, help "shape type of
CTP gradient pulses";
    grad_recover             => 1.0[ms], help "gradient recovery time";

    grad_spoil               => 1[ms], help "duration of grad_1 during mixing";
    grad_spoil_amp              => 133[mT/m], help "Enter amplitude in Tesla/meter
units";
    grad_recover_zqs         => 3.0[ms], help "gradient recovery time";
```

```
comment_12               =? "*** TOCSY mixing time ***";
    x_spinlock_pulse        => x90_spin, help "90deg pulse width of isotropic mixing
sequence(clean-DIPSI2)";
    x_spinlock_atn           => xatn_spin, help "attenuator of x_spinlock_pulse";
    tau_interval                => 10[us], help "tau interval of isotropic mixing
sequence(clean-DIPSI2) about 10-20use";
    mix_time                 => 60[ms], help "mixing time of TOCSY(clean-dipsi2)";
    mix_time_loop             =? round (mix_time / (x_spinlock_pulse * 10240 / 90 +
36 * tau_interval));
    total_mix_time           =? (x_spinlock_pulse * 10240 / 90 + 36 * tau_interval)
* mix_time_loop;

comment_41               =? "*** Zero-Quantum Dephasing ***";
    obs_chp_shape            =? "CHIRP", ad_shape_names, help "shape pulse";
    obs_chp1_m_pulse           => 50[ms], (30[ms], 40[ms], 50[ms]), help "ZQfilter
pulse width";
    obs_chp1_m_fsweep        =? if obs_chp1_m_pulse = 30[ms]
                                then x_sweep * 2
                                else if obs_chp1_m_pulse = 40[ms]
                                then x_sweep * 2
                                else x_sweep * 2;
    obs_chp1_sweep_ppm       =? round (obs_chp1_m_fsweep * 1[Mppm] / obsfreq), help
"obs_chp1_ad_sweep_ppm";
    obs_chp1_chirp_smooth    =  10[%];
    obs_chp1_m_q0              =  11, help "q >= 5 On resonance adiabatic Quality
factor";
    obs_chp1_m_ph_rev        =  1, (1, -1), help "phase reverse";
    obs_chp1_b1_calc         =  sqrt (obs_chp1_m_q0 * obs_chp1_m_fsweep / (6.28319 *
obs_chp1_m_pulse));
    obs_chp1_m_b1max         =? obs_chp1_b1_calc;
    obs_chp1_pw90            =  1 / (4 * obs_chp1_m_b1max);
    obs_chp1_atn_calc       =  if obs_atn + 20[dB] * log (obs_chp1_pw90 / obs_pulse)
< obs_atn + 6[dB]
                                then obs_atn + 6[dB]
                                else  obs_atn  +  20[dB]  *  log  (obs_chp1_pw90  /
obs_pulse);
    obs_chp1_atn             =? obs_chp1_atn_calc;
    obs_chp2_m_pulse           => 30[ms], (30[ms], 40[ms], 50[ms]), help "ZQfilter
pulse width";
    obs_chp2_m_fsweep        =? if obs_chp2_m_pulse = 30[ms]
                                then x_sweep * 2
                                else if obs_chp2_m_pulse = 40[ms]
                                then x_sweep * 2
                                else x_sweep * 2;
    obs_chp2_sweep_ppm       =? round (obs_chp2_m_fsweep * 1[Mppm] / obsfreq), help
"obs_chirp2_ad_sweep_ppm";
    obs_chp2_chirp_smooth    =  10[%];
    obs_chp2_m_q0              =  11, help "q >= 5 On resonance adiabatic Quality
factor";
    obs_chp2_m_ph_rev        =  1, (1, -1), help "phase reverse";
    obs_chp2_b1_calc         =  sqrt (obs_chp2_m_q0 * obs_chp2_m_fsweep / (6.28319 *
obs_chp2_m_pulse));
    obs_chp2_m_b1max         =? obs_chp2_b1_calc;
    obs_chp2_pw90            =  1 / (4 * obs_chp2_m_b1max);
    obs_chp2_atn_calc       =  if obs_atn + 20[dB] * log (obs_chp2_pw90 / obs_pulse)
< obs_atn + 6[dB]
                                then obs_atn + 6[dB]
                                else  obs_atn  +  20[dB]  *  log  (obs_chp2_pw90  /
obs_pulse);
    obs_chp2_atn             =? obs_chp2_atn_calc;
    g_factor                 =  1, help "obs_gamma/1H_gamma";
    coil_factor              =  1, help "coil_length/20mm";
    f_factor_zqf1                  =   obs_chp1_m_fsweep  /  40[kHz],  help  "ref
10ppm*8/500MHz=40kHz";
    grad_zqf1_amp_calc        =  47[mT/m] * f_factor_zqf1 / coil_factor / g_factor,
help "gradient amplitude of grad_sl_ps";
    grad_zqf1_amp                =? grad_zqf1_amp_calc, help "Enter amplitude in
Tesla/meter units";
```

```
    f_factor_zqf2                    =    obs_chp2_m_fsweep  /  40[kHz],  help  "ref
10ppm*8/500MHz=40kHz";
    grad_zqf2_amp_calc       =  47[mT/m] * f_factor_zqf2 / coil_factor / g_factor,
help "gradient amplitude of grad_sl_ps";
    grad_zqf2_amp                =? grad_zqf2_amp_calc,  help  "Enter  amplitude  in
Tesla/meter units";


comment_7 =? "*** Pulse Delay ***";
    initial_wait     =  1[s];
    include "relaxation_delay_1d_calc";
    relaxation_delay => 2[s]; -- relaxation_delay_default, help "relaxation delay";
    repetition_time          =?     relaxation_delay     +     x_acq_time,     help
"relaxation_delay+x_acq_time";

comment_900              =? "*** lock hold ***";
    lock_hold                => TRUE, help "select TRUE or FALSE for lock hold";

    include "pulse";

    phase_90  = {0};
    phase_asr = {0, 90};
    phase_ad1 = {0};
    phase_ad2 = {0}; -- {16(0),16(90)};
    phase_ad3 = {0};
    phase_ad4 = {0}; -- {4(0), 4(90)};

    phase_ad1r= {0};
    phase_ad2r= {0}; -- {8(0),8(90)};
    phase_ad3r= {0};
    phase_ad4r= {0}; -- {2(0), 2(90)};

    phase_spinlock= {0, 90};
    --scans        1    2       4                8                                  16
32
    --phase_acq    =    {0,180,180,0,    180,0,0,180,    180,0,0,180,    0,180,180,0,
180,0,0,180,0,180,180,0,0,180,180,0,180,0,0,180   };
       phase_acq = {0,180};

    --phase sequencing options
    phase_mlev4  = {0,0,180,180};
    phase_mlev8  = {0,0,180,180, 0,180,180,0};
    phase_mlev16 = {0,0,180,180, 0,180,180,0, 180,180,0,0, 180,0,0,180};
    phase_t5 = {0,150,60,150,0};
    phase_t7 = {0,105,300,255,300,105,0};
    phase_t9 = {0,15,180,165,270,165,180,15,0};

module_config = "";

begin
    initial_wait;
    relaxation_delay;

    when lock_hold do
        on LOCKHOLD;
    end when;
    1[us];

    x_pulse, (obs.gate, obs.phs.phase_90, obs.atn.x_atn);

    loop gs_loop times
        parallel
            justify center
                obs_ad180_m_pulse,(obs.gate,obs.phs.phase_ad1,obs.atn.obs_ad180_atn,
obs.shape.{obs_ad180_shape,"obs_ad180"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slPos_amp,fgz.shape.grad_shape_sl);
```

```
        end parallel;
        grad_recover_sl;

        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp1, fgz.shape.grad_shape);
            grad_recover;
        end when;
        parallel
            justify center
                obs_ad180_m_pulse,(obs.gate,obs.phs.phase_ad2,obs.atn.obs_ad180_atn,
obs.shape.{obs_ad180_shape,"obs_ad180"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slNeg_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp1, fgz.shape.grad_shape);
            grad_recover;
        end when;

        parallel
            justify center
                obs_ad180_m_pulse,(obs.gate,obs.phs.phase_ad3,obs.atn.obs_ad180_atn,
obs.shape.{obs_ad180_shape,"obs_ad180"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slPos_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;

        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp2, fgz.shape.grad_shape);
            grad_recover;
        end when;
        parallel
            justify center
                obs_ad180_m_pulse,(obs.gate,obs.phs.phase_ad4,obs.atn.obs_ad180_atn,
obs.shape.{obs_ad180_shape,"obs_ad180"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slNeg_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp2, fgz.shape.grad_shape);
            grad_recover;
        end when;
    end loop;

    grad_1, (fgz.gate, fgz.amp.grad_1_amp, fgz.shape.grad_shape);
    grad_recover;
    obs_asr180,      (obs.gate,      obs.phs.phase_asr,      obs.atn.obs_asr180_atn,
obs.shape.obs_asr180_shape);
    grad_1, (fgz.gate, fgz.amp.grad_1_amp, fgz.shape.grad_shape);
    grad_recover;

    loop gs_loop times
        parallel
            justify center
                obs_ad180r_m_pulse,          (obs.gate,          obs.phs.phase_ad1r,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180r"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slPos_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;
```

```
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp3, fgz.shape.grad_shape);
            grad_recover;
        end when;
        parallel
            justify center
                obs_ad180r_m_pulse,          (obs.gate,          obs.phs.phase_ad2r,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180r"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slNeg_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp3, fgz.shape.grad_shape);
            grad_recover;
        end when;

        parallel
            justify center
                obs_ad180r_m_pulse,          (obs.gate,          obs.phs.phase_ad3r,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180r"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slPos_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;

        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp4, fgz.shape.grad_shape);
            grad_recover;
        end when;
        parallel
            justify center
                obs_ad180r_m_pulse,          (obs.gate,          obs.phs.phase_ad4r,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180r"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slNeg_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp4, fgz.shape.grad_shape);
            grad_recover;
        end when;
    end loop;

    x_pulse, (obs.gate, obs.phs.phase_90, obs.atn.x_atn);
    10[us];
    on (FGZ.GATE, FGZ.AMP.grad_zqf1_amp);
    obs_chp1_m_pulse,     (OBS.GATE,     OBS.PHS.phase_90,     OBS.ATN.obs_chp1_atn,
OBS.SHAPE.{obs_chp_shape, "obs_chp1"});
    off (FGZ.GATE, FGZ.AMP.grad_zqf1_amp);
    grad_recover;
    loop mix_time_loop times
        dipsi2_rc(    x_spinlock_pulse,    obs.phs.phase_spinlock,    x_spinlock_atn,
tau_interval );
    end loop;
    10[us];
    grad_spoil, (fgz.gate, fgz.amp.grad_spoil_amp, fgz.shape.grad_shape);
    grad_recover;
    on (FGZ.GATE, FGZ.AMP.grad_zqf2_amp);
    obs_chp2_m_pulse,     (OBS.GATE,     OBS.PHS.phase_90,     OBS.ATN.obs_chp2_atn,
OBS.SHAPE.{obs_chp_shape, "obs_chp2"});
    off (FGZ.GATE, FGZ.AMP.grad_zqf2_amp);
    grad_recover;
    grad_spoil, (fgz.gate, fgz.amp.grad_spoil_amp*0.77, fgz.shape.grad_shape);
    grad_recover;
```

```
    grad_recover_zqs;
    x_pulse, (obs.gate, obs.phs.phase_90, obs.atn.x_atn);

    acq( dead_time, delay, phase_acq );
    10[us];
    when lock_hold do
        off LOCKHOLD;
    end when;
    10[us];

       relaxation_delay;

    when lock_hold do
        on LOCKHOLD;
    end when;
    1[us];

    x_pulse, (obs.gate, obs.phs.phase_90, obs.atn.x_atn);

    loop gs_loop times
        parallel
            justify center
                obs_ad180_m_pulse,(obs.gate,obs.phs.phase_ad1,obs.atn.obs_ad180_atn,
obs.shape.{obs_ad180_shape,"obs_ad180"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slNeg_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;

        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp1n, fgz.shape.grad_shape);
            grad_recover;
        end when;
        parallel
            justify center
                obs_ad180_m_pulse,(obs.gate,obs.phs.phase_ad2,obs.atn.obs_ad180_atn,
obs.shape.{obs_ad180_shape,"obs_ad180"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slPos_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp1n, fgz.shape.grad_shape);
            grad_recover;
        end when;

        parallel
            justify center
                obs_ad180_m_pulse,(obs.gate,obs.phs.phase_ad3,obs.atn.obs_ad180_atn,
obs.shape.{obs_ad180_shape,"obs_ad180"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slNeg_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;

        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp2n, fgz.shape.grad_shape);
            grad_recover;
        end when;
        parallel
            justify center
                obs_ad180_m_pulse,(obs.gate,obs.phs.phase_ad4,obs.atn.obs_ad180_atn,
obs.shape.{obs_ad180_shape,"obs_ad180"});
            justify center
```

```
                grad_sl, (fgz.gate, fgz.amp.grad_slPos_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp2n, fgz.shape.grad_shape);
            grad_recover;
        end when;
    end loop;

    grad_1, (fgz.gate, fgz.amp.grad_1_amp, fgz.shape.grad_shape);
    grad_recover;
    obs_asr180,      (obs.gate,      obs.phs.phase_asr,      obs.atn.obs_asr180_atn,
obs.shape.obs_asr180_shape);
    grad_1, (fgz.gate, fgz.amp.grad_1_amp, fgz.shape.grad_shape);
    grad_recover;

    loop gs_loop times
        parallel
            justify center
                obs_ad180r_m_pulse,          (obs.gate,          obs.phs.phase_ad1r,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180r"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slNeg_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;

        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp3n, fgz.shape.grad_shape);
            grad_recover;
        end when;
        parallel
            justify center
                obs_ad180r_m_pulse,          (obs.gate,          obs.phs.phase_ad2r,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180r"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slPos_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp3n, fgz.shape.grad_shape);
            grad_recover;
        end when;

        parallel
            justify center
                obs_ad180r_m_pulse,          (obs.gate,          obs.phs.phase_ad3r,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180r"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slNeg_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;

        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp4n, fgz.shape.grad_shape);
            grad_recover;
        end when;
        parallel
            justify center
                obs_ad180r_m_pulse,          (obs.gate,          obs.phs.phase_ad4r,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180r"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slPos_amp,fgz.shape.grad_shape_sl);

        end parallel;
```

```
        grad_recover_sl;
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp4n, fgz.shape.grad_shape);
            grad_recover;
        end when;
    end loop;

    x_pulse, (obs.gate, obs.phs.phase_90, obs.atn.x_atn);
    10[us];
    on (FGZ.GATE, FGZ.AMP.grad_zqf1_amp);
    obs_chp1_m_pulse,     (OBS.GATE,     OBS.PHS.phase_90,     OBS.ATN.obs_chp1_atn,
OBS.SHAPE.{obs_chp_shape, "obs_chp1"});
    off (FGZ.GATE, FGZ.AMP.grad_zqf1_amp);
    grad_recover;
    loop mix_time_loop times
        dipsi2_rc(    x_spinlock_pulse,    obs.phs.phase_spinlock,    x_spinlock_atn,
tau_interval );
    end loop;
    10[us];
    grad_spoil, (fgz.gate, fgz.amp.grad_spoil_amp, fgz.shape.grad_shape);
    grad_recover;
    on (FGZ.GATE, FGZ.AMP.grad_zqf2_amp);
    obs_chp2_m_pulse,     (OBS.GATE,     OBS.PHS.phase_90,     OBS.ATN.obs_chp2_atn,
OBS.SHAPE.{obs_chp_shape, "obs_chp2"});
    off (FGZ.GATE, FGZ.AMP.grad_zqf2_amp);
    grad_recover;
    grad_spoil, (fgz.gate, fgz.amp.grad_spoil_amp*0.77, fgz.shape.grad_shape);
    grad_recover;

    grad_recover_zqs;
    x_pulse, (obs.gate, obs.phs.phase_90, obs.atn.x_atn);

    acq( dead_time, delay, phase_acq );
    10[us];
    when lock_hold do
        off LOCKHOLD;
    end when;
    10[us];

end pulse;
```

5.5 JND-GEMSTONE NOESY for JEOL spectrometers

```
-------------------------------------------------------------------------------
--                                                                           --
--                          Experiment Source Code                           --
--              Delta NMR Experiment & Machine Control Interface             --
--                                                                           --
--                     Copyright (c) 2021 JEOL Ltd                           --
--                            All Rights Reserved                            --
--                                                                           --
-------------------------------------------------------------------------------
-- HELP.eng: GEMSTONE
-- Category: 1D, liquid_advanced, gemstone
-- File name : fsGEMSTONE_NOESY
--
-- Sequence name : kp_fsGEMSTONE_NOESY_16ph2
--
-- Reference :
--  P. Kiraly et al., Angew. Chem. Int. Ed. 2021, 60, 666-669. DOI:
10.1002/anie.202011642
-- P. Kiraly et al., Chem. Commun. , 2021, 57, 2368-2371. DOI: 10.1039/D0CC08033K
--  M. Bazzoni et al., Angew. Chem. Int. Ed. 2023, 62, e202314598. DOI:
10.1002/anie.202314598
--
```

```
-- Parameters
-- x_pulse          : 90[deg] pulse width
-- x_atn            : attenuator of x_pulse
--
-- obs_sel_180      : 180[deg] selective pulse
-- obs_sel_offset   : offset for obs_sel_180
-- obs_sel_atn      : attenuator of obs_sel_180
-- obs_sel_shape    : pulse shape of obs_sel_180
--
--  mix_time        : mixting time of NOESY
--
-- relaxation_delay : inter-pulse delay
-- repetition_time  : pulse repetition_time (= relaxation_delay+x_acq_time)
--
-- Note :
-- scans = 1*n
--
-- fs-GEMSTONE sequence
--  x90-(ad180/+Ge  ad180/-Ge  ad180/+Ge  ad180/-Ge)loop  G1-sel180-G1  (ad180r/+Ge
ad180r/-Ge ad180r/+Ge ad180r/-Ge)loop- x90-mix_time(noesy)-x90-acq
--
--
--

header
    filename        => "fsGEMSTONE_NOESY";
    sample_id       => "";
    comment         => "fs-GEMSTONE_NOESY";
    process         =  "kp_basic1D.list";
    include "header";
end header;

instrument
    include "instrument";
    recvr_gain          => 40;
    spin_state          => "SPIN OFF";
end instrument;

acquisition
    x_domain         => "Proton";
    x_offset         => 1460.0[Hz];
    x_sweep          => 10[kHz];
    include "x_points_default_1d_calc";
    x_points        => 8192; --x_points_default;

    scans            => 8, help "scans = 8*n";
    x_prescans       => 8;
    mod_return       => 8;
    include "acquisition";
end acquisition;

pulse
    collect COMPLEX,OBS;

    macro comp180( pulse_90, phase ,atn1 )is
        pulse_90, (obs.gate,obs.phs.phase,obs.atn.atn1);
        pulse_90*240/90, (obs.gate, obs.phs.phase+90,obs.atn.atn1);
        pulse_90, (obs.gate, obs.phs.phase,obs.atn.atn1);
    end comp180;

    experiment              =  "kp_fsGEMSTONE_NOESY_16ph2.jxp";
    obs_domain              =  x_domain;
    obs_offset              =  x_offset;
    obsfreq                 =  _get_freq(obs_domain);

comment_1               =? "*** Pulse ***";
    x_pulse                 => x90, help "observe 90[deg] pulse";
    x_atn                   =? xatn, help "attenuator for x_pulse";
```

```
    obs_pulse                =  x_pulse;
    obs_atn                  =  x_atn;


comment_16               =? "*** GEMSTONE: adiabatic 180 pulses ***";
    gemstone_selectivity      => 10[Hz], 0.5[Hz]->50[Hz]:0.001[Hz], help "bandwidth
of sinc excitation profile";
    gemstone_encoding        => 1/gemstone_selectivity, 20[ms] -> 2000[ms] : 1[ms],
help "total chemical shift encoding time";
    gs_target_pw              => 20[ms], (10[ms], 20[ms], 30[ms], 40[ms], 50[ms]),
help "adiabatic pulse duration desired, typically 20ms";

    gs_calc1                 = gemstone_encoding / (8*gs_target_pw);
    gs_calc2                 = lower(gs_calc1 + 0.001);
    gs_calc4                  = if gs_calc2=0 then gs_target_pw else gs_target_pw *
gs_calc1 / gs_calc2;

    gs_calc1m                = gemstone_encoding / (8 * 10[ms]);
    gs_calc2m                = lower(gs_calc1m + 0.001);
    gs_calc4m                = if gs_calc2m=0 then 10[ms] else 10[ms] * gs_calc1m /
gs_calc2m;

    gs_loop_number           =? if gs_calc1>=0.999 then gs_calc2 else gs_calc2m;
    gs_pulse_duration        =? if gs_calc1>=0.999 then
                                   if  gs_calc4  >=  gs_target_pw  and  gs_calc4  <=
2*gs_target_pw then gs_calc4 else gs_target_pw
                               else
                                   if  gs_calc4m  >=  9.999[ms]  and  gs_calc4m  <=
20.001[ms] then gs_calc4m else 10[ms];
    gs_encodingTotal         =? 8 * gs_pulse_duration * gs_loop_number;


    gs_loop                 => gs_loop_number, 0 -> 32 : 1, help "number of cycles";


    obs_ad180_shape           => "CHIRP", ad_shape_names, help "smoothed CHIRP 180
pulse shape";
    obs_ad180_m_pulse         => gs_pulse_duration, 10[ms] -> 100[ms] : 1[ms], help
"adiabatic pulse duration = gs_encodingTotal / (8*gs_loop_number)";
    obs_ad180_m_fsweep       => 5[kHz], help "adiabatic pulse sweep width";
    obs_ad180_sweep_ppm      =  obs_ad180_m_fsweep / obsfreq * 1[Mppm];
    obs_ad180_chirp_smooth   => 10[%], 0[%] -> 50[%] : 1[%], help "smoothing";
    obs_ad180_m_q0            => 11, 3 -> 20 : 1, help "adiabaticity factor typically
11";
    obs_ad180_m_ph_rev         =  1;
    obs_ad180_b1_calc        =  sqrt (obs_ad180_m_q0 * obs_ad180_m_fsweep / (6.28319
* obs_ad180_m_pulse));
    obs_ad180_m_b1max          => obs_ad180_b1_calc;
    obs_ad180_pw90           =  1 / (4 * obs_ad180_m_b1max);
    obs_ad180_atn_calc       =  obs_atn + 20[dB] * log (obs_ad180_pw90 / obs_pulse);
    obs_ad180_atn            => obs_ad180_atn_calc;

    obs_ad180r_m_fsweep        =  obs_ad180_m_fsweep;
    obs_ad180r_m_pulse         =  obs_ad180_m_pulse;

    obs_ad180r_chirp_smooth  =  obs_ad180_chirp_smooth;
    obs_ad180r_m_q0            =  obs_ad180_m_q0;
    obs_ad180r_m_ph_rev        = -1, (1, -1), help "phase reverse";
    obs_ad180r_atn           =  obs_ad180_atn;


comment_17               =? "*** GEMSTONE: gradients ***";
    grad_slPos_amp             => -6.0[mT/m], help "Amplitude of spatial encoding
gradient during 1st chirp";
    grad_slTweak                => 1.0, 0.0 -> 2.0 : 0.001, help "tweak gradient
amplitude";
    grad_slNeg_amp           =? -1 * grad_slPos_amp * grad_slTweak, help "Amplitude
of spatial encoding gradient during 2nd chirp";
```

```
    grad_recover_sl          => 1[ms], help "gradient recovery time in loop";
    grad_sl                  =? obs_ad180_m_pulse;
    grad_shape_sl           =? "SQUARE", help "Gradient shape for spatial encoding";

    grad_ctp                 => TRUE, help "Use optional CTP gradient pulses";
    grad_ctp_sl             => 1.0[ms], help "optional CTP gradient pulse durations";
    grad_ctp_amp1            => 94[mT/m], help "CTP gradient pulse amplitude in the
loop";
    grad_ctp_amp2            => 59[mT/m], help "CTP gradient pulse amplitude in the
loop";
    grad_ctp_amp3           => 112[mT/m], help "CTP gradient pulse amplitude in the
loop";
    grad_ctp_amp4            => 79[mT/m], help "CTP gradient pulse amplitude in the
loop";
       grad_ctp_amp1n =? -grad_ctp_amp1;
       grad_ctp_amp2n =? -grad_ctp_amp2;
       grad_ctp_amp3n =? -grad_ctp_amp3;
       grad_ctp_amp4n =? -grad_ctp_amp4;

comment_18              =? "*** GEMSTONE: Active Spin Refocusing 180 Pulse ***";
    obs_asr180_shape         => "RSNOB_180", std_shape_names, help "selective pulse
shape";
    soft_shape_input         =  obs_asr180_shape;

    --include "soft_pulse_calc";
    soft_bw_input                => 100[Hz], help "bandwidth in [ppm] or [Hz] of
obs_sel_180";
    xfreq                    =  _get_freq(x_domain);
    soft_angle              =? _get_s_mparam(soft_shape_input, "m_angle", 90[deg]);
    int_soft_pc              =  _get_s_mparam(soft_shape_input, "m_integ", 100[%]);
    int_r_soft_pc             =  int_soft_pc / _get_s_mparam("gauss_90", "m_integ",
100[%]);
    unit_soft_pc             =  if soft_bw_input * 0 = 0[ppm]
                                then 1
                                else if soft_bw_input * 0 = 0[Hz]
                                then 2
                                else 3;
    soft_bw_hz               =  if unit_soft_pc = 1
                                then soft_bw_input * xfreq / 1[Mppm]
                                else if unit_soft_pc = 2
                                then soft_bw_input
                                else 100[Hz];
    type_soft_pc             =  _get_s_mparam(soft_shape_input, "m_type", "std");
    is_std_pc                =  if type_soft_pc = "std" and unit_soft_pc < 3
                                then TRUE
                                else FALSE;
    soft_pw_calc             =? if is_std_pc
                                then  _get_s_mparam(soft_shape_input,  "bw",  1)  /
soft_bw_hz
                                else 1[us];
    soft_atn_calc            =? if is_std_pc
                                then xatn_soft + 20[dB] * log (soft_pw_calc / x90_soft
* 90[deg] / soft_angle * int_r_soft_pc)
                                else 79[dB];

    obs_asr180               => soft_pw_calc, help "selective 180 pulse duration";
    obs_asr180_atn           => soft_atn_calc, help "attenuator of selective pulse,
obs_asr_180";

comment_8               =? "*** Pulse Field Gradient ***";
    gradient_max             =? z_gradient_max, help "Maximum amplitude for a given
probe as defined in the probe file";
    grad_1                   => 1[ms], help "CTP gradient pulse duration";
    grad_1_amp               => 160[mT/m], help "CTP gradient pulse amplitude";
       grad_1_ampn =? -grad_1_amp;
    grad_shape               => "S_RECTANGLE", fg_shape_names, help "shape type of
CTP gradient pulses";
```

```
    grad_recover            => 1.0[ms], help "gradient recovery time";

    grad_spoil              => 1[ms], help "duration of grad_1 during mixing";
    grad_spoil_amp             => 133[mT/m], help "Enter amplitude in Tesla/meter
units";
       grad_spoiln_amp =? -grad_spoil_amp;
    grad_recover_zqs        => 3.0[ms], help "gradient recovery time";

comment_11 =? "*** NOESY mixing time ***";
    mix_time        => 500[ms], help "mixing time of NOESY (= about T1)";

comment_41              =? "*** Zero-Quantum Dephasing ***";
    obs_chp_shape           =? "CHIRP", ad_shape_names, help "shape pulse";
    obs_chp1_m_pulse          => 50[ms], (30[ms], 40[ms], 50[ms]), help "ZQfilter
pulse width";
    obs_chp1_m_fsweep       =? if obs_chp1_m_pulse = 30[ms]
                               then x_sweep * 2
                               else if obs_chp1_m_pulse = 40[ms]
                               then x_sweep * 2
                               else x_sweep * 2;
    obs_chp1_sweep_ppm      =? round (obs_chp1_m_fsweep * 1[Mppm] / obsfreq), help
"obs_chp1_ad_sweep_ppm";
    obs_chp1_chirp_smooth   =  10[%];
    obs_chp1_m_q0             =  11, help "q >= 5 On resonance adiabatic Quality
factor";
    obs_chp1_m_ph_rev       =  1, (1, -1), help "phase reverse";
    obs_chp1_b1_calc        =  sqrt (obs_chp1_m_q0 * obs_chp1_m_fsweep / (6.28319 *
obs_chp1_m_pulse));
    obs_chp1_m_b1max        =? obs_chp1_b1_calc;
    obs_chp1_pw90           =  1 / (4 * obs_chp1_m_b1max);
    obs_chp1_atn_calc       =  if obs_atn + 20[dB] * log (obs_chp1_pw90 / obs_pulse)
< obs_atn + 6[dB]
                               then obs_atn + 6[dB]
                               else  obs_atn  +  20[dB]  *  log  (obs_chp1_pw90  /
obs_pulse);
    obs_chp1_atn            =? obs_chp1_atn_calc;
    obs_chp2_m_pulse          => 30[ms], (30[ms], 40[ms], 50[ms]), help "ZQfilter
pulse width";
    obs_chp2_m_fsweep       =? if obs_chp2_m_pulse = 30[ms]
                               then x_sweep * 2
                               else if obs_chp2_m_pulse = 40[ms]
                               then x_sweep * 2
                               else x_sweep * 2;
    obs_chp2_sweep_ppm      =? round (obs_chp2_m_fsweep * 1[Mppm] / obsfreq), help
"obs_chirp2_ad_sweep_ppm";
    obs_chp2_chirp_smooth   =  10[%];
    obs_chp2_m_q0             =  11, help "q >= 5 On resonance adiabatic Quality
factor";
    obs_chp2_m_ph_rev       =  1, (1, -1), help "phase reverse";
    obs_chp2_b1_calc        =  sqrt (obs_chp2_m_q0 * obs_chp2_m_fsweep / (6.28319 *
obs_chp2_m_pulse));
    obs_chp2_m_b1max        =? obs_chp2_b1_calc;
    obs_chp2_pw90           =  1 / (4 * obs_chp2_m_b1max);
    obs_chp2_atn_calc       =  if obs_atn + 20[dB] * log (obs_chp2_pw90 / obs_pulse)
< obs_atn + 6[dB]
                               then obs_atn + 6[dB]
                               else  obs_atn  +  20[dB]  *  log  (obs_chp2_pw90  /
obs_pulse);
    obs_chp2_atn            =? obs_chp2_atn_calc;
    g_factor                =  1, help "obs_gamma/1H_gamma";
    coil_factor             =  1, help "coil_length/20mm";
    f_factor_zqf1                 =   obs_chp1_m_fsweep  /  40[kHz],  help  "ref
10ppm*8/500MHz=40kHz";
    grad_zqf1_amp_calc       =  47[mT/m] * f_factor_zqf1 / coil_factor / g_factor,
help "gradient amplitude of grad_sl_ps";
    grad_zqf1_amp              =? grad_zqf1_amp_calc, help "Enter amplitude in
Tesla/meter units";
```

```
    f_factor_zqf2                    =    obs_chp2_m_fsweep  /  40[kHz],  help  "ref
10ppm*8/500MHz=40kHz";
    grad_zqf2_amp_calc       =  47[mT/m] * f_factor_zqf2 / coil_factor / g_factor,
help "gradient amplitude of grad_sl_ps";
    grad_zqf2_amp                =? grad_zqf2_amp_calc, help "Enter amplitude in
Tesla/meter units";
       grad_zqf1n_amp =? -grad_zqf1_amp;
    grad_zqf2n_amp =? -grad_zqf2_amp;

comment_7 =? "*** Pulse Delay ***";
    initial_wait     =  1[s];
    include "relaxation_delay_1d_calc";
    relaxation_delay => 2[s]; -- relaxation_delay_default, help "relaxation delay";
    repetition_time          =?     relaxation_delay     +     x_acq_time,     help
"relaxation_delay+x_acq_time";

    mix_time2 = mix_time - obs_chp1_m_pulse - 10[us] - grad_spoil - grad_recover -
grad_recover;

comment_900              =? "*** lock hold ***";
    lock_hold                => TRUE, help "select TRUE or FALSE for lock hold";

    include "pulse";

    phase_x   = {0};
    phase_90  = {0,180};
    phase_asr = {2(0),2(90),2(180),2(270)};
    phase_zero = {0};
    --scans      1   2     4            8
    phase_acq = {180,0,0,180};


    phase_ad1 = {0};
    phase_ad2 = {0}; -- {16(0),16(90)};
    phase_ad3 = {0};
    phase_ad4 = {0}; -- {4(0), 4(90)};

    phase_ad1r= {0};
    phase_ad2r= {0}; -- {8(0),8(90)};
    phase_ad3r= {0};
    phase_ad4r= {0}; -- {2(0), 2(90)};

    --phase sequencing options
    phase_mlev4  = {0,0,180,180};
    phase_mlev8  = {0,0,180,180, 0,180,180,0};
    phase_mlev16 = {0,0,180,180, 0,180,180,0, 180,180,0,0, 180,0,0,180};
    phase_t5 = {0,150,60,150,0};
    phase_t7 = {0,105,300,255,300,105,0};
    phase_t9 = {0,15,180,165,270,165,180,15,0};

module_config = "";

begin
    initial_wait;
    relaxation_delay;

    when lock_hold do
        on LOCKHOLD;
    end when;
    1[us];

    x_pulse, (obs.gate, obs.phs.phase_90, obs.atn.x_atn);

    loop gs_loop times
        parallel
            justify center
                obs_ad180_m_pulse,(obs.gate,obs.phs.phase_ad1,obs.atn.obs_ad180_atn,
obs.shape.{obs_ad180_shape,"obs_ad180"});
```

```
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slPos_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;

        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp1, fgz.shape.grad_shape);
            grad_recover;
        end when;
        parallel
            justify center
                obs_ad180_m_pulse,(obs.gate,obs.phs.phase_ad2,obs.atn.obs_ad180_atn,
obs.shape.{obs_ad180_shape,"obs_ad180"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slNeg_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp1, fgz.shape.grad_shape);
            grad_recover;
        end when;

        parallel
            justify center
                obs_ad180_m_pulse,(obs.gate,obs.phs.phase_ad3,obs.atn.obs_ad180_atn,
obs.shape.{obs_ad180_shape,"obs_ad180"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slPos_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;

        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp2, fgz.shape.grad_shape);
            grad_recover;
        end when;
        parallel
            justify center
                obs_ad180_m_pulse,(obs.gate,obs.phs.phase_ad4,obs.atn.obs_ad180_atn,
obs.shape.{obs_ad180_shape,"obs_ad180"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slNeg_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp2, fgz.shape.grad_shape);
            grad_recover;
        end when;
    end loop;

    grad_1, (fgz.gate, fgz.amp.grad_1_amp, fgz.shape.grad_shape);
    grad_recover;
    obs_asr180,      (obs.gate,      obs.phs.phase_asr,      obs.atn.obs_asr180_atn,
obs.shape.obs_asr180_shape);
    grad_1, (fgz.gate, fgz.amp.grad_1_amp, fgz.shape.grad_shape);
    grad_recover;

    loop gs_loop times
        parallel
            justify center
                obs_ad180r_m_pulse,          (obs.gate,          obs.phs.phase_ad1r,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180r"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slPos_amp,fgz.shape.grad_shape_sl);
```

```
        end parallel;
        grad_recover_sl;

        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp3, fgz.shape.grad_shape);
            grad_recover;
        end when;
        parallel
            justify center
                obs_ad180r_m_pulse,          (obs.gate,          obs.phs.phase_ad2r,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180r"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slNeg_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp3, fgz.shape.grad_shape);
            grad_recover;
        end when;

        parallel
            justify center
                obs_ad180r_m_pulse,          (obs.gate,          obs.phs.phase_ad3r,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180r"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slPos_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;

        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp4, fgz.shape.grad_shape);
            grad_recover;
        end when;
        parallel
            justify center
                obs_ad180r_m_pulse,          (obs.gate,          obs.phs.phase_ad4r,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180r"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slNeg_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp4, fgz.shape.grad_shape);
            grad_recover;
        end when;
    end loop;

    x_pulse, (obs.gate, obs.phs.phase_zero, obs.atn.x_atn);
    10[us];
    on (FGZ.GATE, FGZ.AMP.grad_zqf1_amp);
    obs_chp1_m_pulse,    (OBS.GATE,    OBS.PHS.phase_zero,    OBS.ATN.obs_chp1_atn,
OBS.SHAPE.{obs_chp_shape, "obs_chp1"});
    off (FGZ.GATE, FGZ.AMP.grad_zqf1_amp);
    grad_recover;
    grad_spoil, (fgz.gate, fgz.amp.grad_spoil_amp, fgz.shape.grad_shape);
    grad_recover;

    mix_time2*0.31-1[ms], (obs.phs.phase_x);
    1[ms], (fgz.gate, fgz.amp.0.015[T/m]);
    comp180(x_pulse, phase_x, x_atn) ;
    1[ms], (fgz.gate, fgz.amp.-0.015[T/m]);
    mix_time2*0.49-2[ms], (obs.phs.phase_x);
    1[ms], (fgz.gate, fgz.amp.0.021[T/m]);
    comp180(x_pulse, phase_x, x_atn) ;
    1[ms], (fgz.gate, fgz.amp.-0.021[T/m]);
```

```
    mix_time2*0.20-1[ms], (obs.phs.phase_x);

    x_pulse, (obs.gate, obs.phs.phase_zero, obs.atn.x_atn);

    acq( dead_time, delay, phase_acq );
    10[us];
    when lock_hold do
        off LOCKHOLD;
    end when;
    10[us];

       relaxation_delay;

    when lock_hold do
        on LOCKHOLD;
    end when;
    1[us];

    x_pulse, (obs.gate, obs.phs.phase_90, obs.atn.x_atn);

    loop gs_loop times
        parallel
            justify center
                obs_ad180_m_pulse,(obs.gate,obs.phs.phase_ad1,obs.atn.obs_ad180_atn,
obs.shape.{obs_ad180_shape,"obs_ad180"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slNeg_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;

        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp1n, fgz.shape.grad_shape);
            grad_recover;
        end when;
        parallel
            justify center
                obs_ad180_m_pulse,(obs.gate,obs.phs.phase_ad2,obs.atn.obs_ad180_atn,
obs.shape.{obs_ad180_shape,"obs_ad180"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slPos_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp1n, fgz.shape.grad_shape);
            grad_recover;
        end when;

        parallel
            justify center
                obs_ad180_m_pulse,(obs.gate,obs.phs.phase_ad3,obs.atn.obs_ad180_atn,
obs.shape.{obs_ad180_shape,"obs_ad180"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slNeg_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;

        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp2n, fgz.shape.grad_shape);
            grad_recover;
        end when;
        parallel
            justify center
                obs_ad180_m_pulse,(obs.gate,obs.phs.phase_ad4,obs.atn.obs_ad180_atn,
obs.shape.{obs_ad180_shape,"obs_ad180"});
            justify center
```

```
                grad_sl, (fgz.gate, fgz.amp.grad_slPos_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp2n, fgz.shape.grad_shape);
            grad_recover;
        end when;
    end loop;

    grad_1, (fgz.gate, fgz.amp.grad_1_amp, fgz.shape.grad_shape);
    grad_recover;
    obs_asr180,      (obs.gate,      obs.phs.phase_asr,      obs.atn.obs_asr180_atn,
obs.shape.obs_asr180_shape);
    grad_1, (fgz.gate, fgz.amp.grad_1_amp, fgz.shape.grad_shape);
    grad_recover;

    loop gs_loop times
        parallel
            justify center
                obs_ad180r_m_pulse,          (obs.gate,          obs.phs.phase_ad1r,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180r"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slNeg_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;

        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp3n, fgz.shape.grad_shape);
            grad_recover;
        end when;
        parallel
            justify center
                obs_ad180r_m_pulse,          (obs.gate,          obs.phs.phase_ad2r,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180r"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slPos_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp3n, fgz.shape.grad_shape);
            grad_recover;
        end when;

        parallel
            justify center
                obs_ad180r_m_pulse,          (obs.gate,          obs.phs.phase_ad3r,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180r"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slNeg_amp,fgz.shape.grad_shape_sl);

        end parallel;
        grad_recover_sl;

        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp4n, fgz.shape.grad_shape);
            grad_recover;
        end when;
        parallel
            justify center
                obs_ad180r_m_pulse,          (obs.gate,          obs.phs.phase_ad4r,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180r"});
            justify center
                grad_sl, (fgz.gate, fgz.amp.grad_slPos_amp,fgz.shape.grad_shape_sl);

        end parallel;
```

```
        grad_recover_sl;
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp4n, fgz.shape.grad_shape);
            grad_recover;
        end when;
    end loop;

    x_pulse, (obs.gate, obs.phs.phase_zero, obs.atn.x_atn);
    10[us];
    on (FGZ.GATE, FGZ.AMP.grad_zqf1n_amp);
    obs_chp1_m_pulse,    (OBS.GATE,    OBS.PHS.phase_zero,    OBS.ATN.obs_chp1_atn,
OBS.SHAPE.{obs_chp_shape, "obs_chp1"});
    off (FGZ.GATE, FGZ.AMP.grad_zqf1n_amp);
    grad_recover;
    grad_spoil, (fgz.gate, fgz.amp.grad_spoiln_amp, fgz.shape.grad_shape);
    grad_recover;

    mix_time2*0.31-1[ms], (obs.phs.phase_x);
    1[ms], (fgz.gate, fgz.amp.-0.015[T/m]);
    comp180(x_pulse, phase_x, x_atn) ;
    1[ms], (fgz.gate, fgz.amp.0.015[T/m]);
    mix_time2*0.49-2[ms], (obs.phs.phase_x);
    1[ms], (fgz.gate, fgz.amp.-0.021[T/m]);
    comp180(x_pulse, phase_x, x_atn) ;
    1[ms], (fgz.gate, fgz.amp.0.021[T/m]);
    mix_time2*0.20-1[ms], (obs.phs.phase_x);

    x_pulse, (obs.gate, obs.phs.phase_zero, obs.atn.x_atn);

    acq( dead_time, delay, phase_acq );
    10[us];
    when lock_hold do
        off LOCKHOLD;
    end when;
    10[us];

end pulse;
```

5.6 JND-GEMSTONE easyROESY for JEOL spectrometers

```
--------------------------------------------------------------------------------
--                                                                            --
--                          Experiment Source Code                            --
--              Delta NMR Experiment & Machine Control Interface              --
--                                                                            --
--                     Copyright (c) 2021 JEOL Ltd                            --
--                            All Rights Reserved                             --
--                                                                            --
--------------------------------------------------------------------------------
-- HELP.eng: GEMSTONE
-- Category: 1D, liquid_advanced, gemstone
-- File name : fsGEMSTONE_eROESY
--
-- Sequence name : kp_fsGEMSTONE_eROESY_16ph2
--
-- Reference :
-- P. Kiraly et al., Angew. Chem. Int. Ed. 2021, 60, 666-669. DOI:
10.1002/anie.202011642
-- P. Kiraly et al., Chem. Commun. , 2021, 57, 2368-2371. DOI: 10.1039/D0CC08033K
-- M. Bazzoni et al., Angew. Chem. Int. Ed. 2023, 62, e202314598. DOI:
10.1002/anie.202314598
-- D. Taylor et al., Chem. Commun., 2023, 59, 6734. DOI: 10.1039/d3cc01333b
-- E.L. Gates et al Chem. Commun. 2023, 59, 5854-5857; DOI: 10.1039/D3CC00550J
--
-- Parameters
```

```
-- x_pulse          : 90[deg] pulse width
-- x_atn            : attenuator of x_pulse
--
-- obs_sel_180      : 180[deg] selective pulse
-- obs_sel_offset   : offset for obs_sel_180
-- obs_sel_atn      : attenuator of obs_sel_180
-- obs_sel_shape    : pulse shape of obs_sel_180
--
-- relaxation_delay : inter-pulse delay
-- repetition_time  : pulse repetition_time (= relaxation_delay+x_acq_time)
--
-- Note :
-- scans = 4*n
--
-- fs-GEMSTONE sequence with easy ROESY transfer
-- x90-(ad180/+Ge ad180/-Ge ad180/+Ge ad180/-Ge)loop G1-sel180-G1 (ad180r/+Ge
ad180r/-Ge ad180r/+Ge ad180r/-Ge)loop- x90-mix_time(easyROESY)-x90-acq
--
--
--

header
    filename        => "fsGEMSTONE_eROESY";
    sample_id       => "";
    comment         => "fs-GEMSTONE_eROESY";
    process         =  "kp_basic1D.list";
    include "header";
end header;

instrument
    include "instrument";
    recvr_gain          => 40;
    spin_state          => "SPIN OFF";
end instrument;

acquisition
    x_domain         => "Proton";
    x_offset         => 5.0[ppm];
    x_sweep          => 10[kHz];
    include "x_points_default_1d_calc";
    x_points        => 8192; --x_points_default;

    scans            => 8, help "scans = 8*n";
    x_prescans       => 8;
    mod_return       => 8;
    include "acquisition";
end acquisition;

pulse
    collect COMPLEX,OBS;

    experiment              =  "kp_fsGEMSTONE_eROESY_16ph2.jxp";
    obs_domain              =  x_domain;
    obs_offset              =  x_offset;
    obsfreq                 =  _get_freq(obs_domain);

comment_1               =? "*** Pulse ***";
    x_pulse                 => x90, help "observe 90[deg] pulse";
    x_atn                   =? xatn, help "attenuator for x_pulse";
    obs_pulse               =  x_pulse;
    obs_atn                 =  x_atn;


comment_16              =? "*** GEMSTONE: adiabatic 180 pulses ***";
    gemstone_selectivity    => 10[Hz], 0.5[Hz]->50[Hz]:0.001[Hz], help "bandwidth
of sinc excitation profile";
    gemstone_encoding       => 1/gemstone_selectivity, 20[ms] -> 2000[ms] : 1[ms],
help "total chemical shift encoding time";
```

```
    gs_target_pw            => 20[ms], (10[ms], 20[ms], 30[ms], 40[ms], 50[ms]),
help "adiabatic pulse duration desired, typically 20ms";

    gs_calc1                = gemstone_encoding / (8*gs_target_pw);
    gs_calc2                = lower(gs_calc1 + 0.001);
    gs_calc4                = if gs_calc2=0 then gs_target_pw else gs_target_pw *
gs_calc1 / gs_calc2;

    gs_calc1m               = gemstone_encoding / (8 * 10[ms]);
    gs_calc2m               = lower(gs_calc1m + 0.001);
    gs_calc4m               = if gs_calc2m=0 then 10[ms] else 10[ms] * gs_calc1m /
gs_calc2m;

    gs_loop_number          =? if gs_calc1>=0.999 then gs_calc2 else gs_calc2m;
    gs_pulse_duration       =? if gs_calc1>=0.999 then
                                  if gs_calc4 >= gs_target_pw and gs_calc4 <=
2*gs_target_pw then gs_calc4 else gs_target_pw
                               else
                                  if gs_calc4m >= 9.999[ms] and gs_calc4m <=
20.001[ms] then gs_calc4m else 10[ms];
    gs_encodingTotal        =? 8 * gs_pulse_duration * gs_loop_number;


    gs_loop                 => gs_loop_number, 0 -> 32 : 1, help "number of
cycles";


    obs_ad180_shape         => "CHIRP", ad_shape_names, help "smoothed CHIRP 180
pulse shape";
    obs_ad180_m_pulse       => gs_pulse_duration, 10[ms] -> 100[ms] : 1[ms], help
"adiabatic pulse duration = gs_encodingTotal / (8*gs_loop_number)";
    obs_ad180_m_fsweep      => 5[kHz], help "adiabatic pulse sweep width";
    obs_ad180_sweep_ppm     =  obs_ad180_m_fsweep / obsfreq * 1[Mppm];
    obs_ad180_chirp_smooth  => 10[%], 0[%] -> 50[%] : 1[%], help "smoothing";
    obs_ad180_m_q0            => 11, 3 -> 20 : 1, help "adiabaticity factor
typically 11";
    obs_ad180_m_ph_rev        =  1;
    obs_ad180_b1_calc       =  sqrt (obs_ad180_m_q0 * obs_ad180_m_fsweep /
(6.28319 * obs_ad180_m_pulse));
    obs_ad180_m_b1max         => obs_ad180_b1_calc;
    obs_ad180_pw90          =  1 / (4 * obs_ad180_m_b1max);
    obs_ad180_atn_calc      =  obs_atn + 20[dB] * log (obs_ad180_pw90 /
obs_pulse);
    obs_ad180_atn           => obs_ad180_atn_calc;

    obs_ad180r_m_fsweep       =  obs_ad180_m_fsweep;
    obs_ad180r_m_pulse        =  obs_ad180_m_pulse;

    obs_ad180r_chirp_smooth  =  obs_ad180_chirp_smooth;
    obs_ad180r_m_q0           =  obs_ad180_m_q0;
    obs_ad180r_m_ph_rev       = -1, (1, -1), help "phase reverse";
    obs_ad180r_atn          =  obs_ad180_atn;


comment_17              =? "*** GEMSTONE: gradients ***";
    grad_slPos_amp          => -6.0[mT/m], help "Amplitude of spatial encoding
gradient during 1st chirp";
    grad_slTweak                => 1.0, 0.0 -> 2.0 : 0.001, help "tweak gradient
amplitude";
    grad_slNeg_amp          =? -1 * grad_slPos_amp * grad_slTweak, help "Amplitude
of spatial encoding gradient during 2nd chirp";

    grad_recover_sl         => 1[ms], help "gradient recovery time in loop";
    grad_sl                 =? obs_ad180_m_pulse;
    grad_shape_sl         =? "SQUARE", help "Gradient shape for spatial encoding";

    grad_ctp                => TRUE, help "Use optional CTP gradient pulses";
```

```
    grad_ctp_sl              => 1.0[ms], help "optional CTP gradient pulse
durations";
    grad_ctp_amp1            => 94[mT/m], help "CTP gradient pulse amplitude in the
loop";
    grad_ctp_amp2            => 59[mT/m], help "CTP gradient pulse amplitude in the
loop";
    grad_ctp_amp3            => 112[mT/m], help "CTP gradient pulse amplitude in
the loop";
    grad_ctp_amp4            => 79[mT/m], help "CTP gradient pulse amplitude in the
loop";
       grad_ctp_amp1n =? -grad_ctp_amp1;
       grad_ctp_amp2n =? -grad_ctp_amp2;
       grad_ctp_amp3n =? -grad_ctp_amp3;
       grad_ctp_amp4n =? -grad_ctp_amp4;

comment_18               =? "*** GEMSTONE: Active Spin Refocusing 180 Pulse ***";
    obs_asr180_shape         => "RSNOB_180", std_shape_names, help "selective pulse
shape";
    soft_shape_input         =  obs_asr180_shape;

    --include "soft_pulse_calc";
    soft_bw_input            => 100[Hz], help "bandwidth in [ppm] or [Hz] of
obs_sel_180";
    xfreq                    =  _get_freq(x_domain);
    soft_angle               =? _get_s_mparam(soft_shape_input, "m_angle",
90[deg]);
    int_soft_pc              =  _get_s_mparam(soft_shape_input, "m_integ", 100[%]);
    int_r_soft_pc            =  int_soft_pc / _get_s_mparam("gauss_90", "m_integ",
100[%]);
    unit_soft_pc             =  if soft_bw_input * 0 = 0[ppm]
                                then 1
                                else if soft_bw_input * 0 = 0[Hz]
                                then 2
                                else 3;
    soft_bw_hz               =  if unit_soft_pc = 1
                                then soft_bw_input * xfreq / 1[Mppm]
                                else if unit_soft_pc = 2
                                then soft_bw_input
                                else 100[Hz];
    type_soft_pc             =  _get_s_mparam(soft_shape_input, "m_type", "std");
    is_std_pc                =  if type_soft_pc = "std" and unit_soft_pc < 3
                                then TRUE
                                else FALSE;
    soft_pw_calc             =? if is_std_pc
                                then _get_s_mparam(soft_shape_input, "bw", 1) /
soft_bw_hz
                                else 1[us];
    soft_atn_calc            =? if is_std_pc
                                then xatn_soft + 20[dB] * log (soft_pw_calc /
x90_soft * 90[deg] / soft_angle * int_r_soft_pc)
                                else 79[dB];

    obs_asr180               => soft_pw_calc, help "selective 180 pulse duration";
    obs_asr180_atn           => soft_atn_calc, help "attenuator of selective pulse,
obs_asr_180";

comment_8                =? "*** Pulse Field Gradient ***";
    gradient_max             =? z_gradient_max, help "Maximum amplitude for a given
probe as defined in the probe file";

    grad_shape               => "S_RECTANGLE", fg_shape_names, help "shape type of
CTP gradient pulses";
    grad_recover             => 0.5[ms], help "gradient recovery time";

    grad_1             => 1[ms], help "duration of CTP encoding gradient";
    grad_1_amp         => 0.037[T/m], help "Amplitude of CTP encoding gradient G1";
    grad_1m_amp         =? -1.0*grad_1_amp, help "Amplitude of inverted CTP
encoding gradient -G1";
```

```
    grad_2             => 1[ms], help "duration of spoiler gradient G2";
    grad_2_amp         => 0.16[T/m], help "Amplitude of spoiler gradient G2";
    grad_2n_amp =? -grad_2_amp;
    grad_4             => 1[ms], help "duration of spoiler gradient G4";
    grad_4_amp         => 0.24[T/m], help "Amplitude of spoiler gradient G4";
    grad_4n_amp =? -grad_4_amp;
    grad_6             => 1[ms], help "duration of CTP refocusing gradient";
    grad_6_amp         =? 2.0*grad_1_amp, help "Amplitude of CTP refocusing
gradient G6";
    grad_6n_amp =? -grad_6_amp;

    delay6 =? grad_6 + grad_recover, help "echo time";

    --grad_spoil               => 1[ms], help "duration of grad_1 during mixing";
    --grad_spoil_amp           => 133[mT/m], help "Enter amplitude in Tesla/meter
units";
    grad_recover_zqs         => 3.0[ms], help "gradient recovery time";

comment_13 =? "*** ROESY mixing time and field strength ***";
   -- use about 7800 for 600Mhz system taken from paper. Scale down or up
accordingly to put offsets outside a likely 10ppm window.
    field = obsfreq/1000000;
       trim =? (x_sweep/10)/field;
       spinlock_strength => trim*((field*7800)/600) , help "Use about 7800Hz B1 for
10ppm in a 600. This expression sets";
    --originally .. spinlock_strength => 6.53[kHz] , help "spin lock field
2~8[kHz]";
    spinlock_pw       =  0.25*(1/spinlock_strength), help"spinlock pulse width";
    spinlock_power    =? xatn_spin+20[dB]*log(spinlock_pw/x90_spin), help
"attenuator of spinlock";
    mix_time          => 200[ms], help "mixing time of ROESY (about 100-300msec)";

    spinlock_pulse    =  mix_time/2;
  --  originally.. tilt_angle        => 60.0[deg], 1.0[deg] -> 179.0[deg] :
0.1[deg];
    tilt_angle          => 60.0[deg], (60[deg],45[deg],54.7[deg],70[deg]);
    tilt_factor1      =  if tilt_angle = 90.0[deg] then 572.95721
                         else tan(tilt_angle * pi`[rad] / 180[deg]);
    tilt_factor       =? tilt_factor1, help "roesy tilt between 1.8 & 1.2 with
1.73=60degrees";

    fieldshift => 1[ppm]*((spinlock_strength/tilt_factor)/field);

    rpos1 =? x_offset + fieldshift;
    rpos2 =? x_offset - fieldshift;

    hG_ramp_pulse => 10[ms];

comment_41               =? "*** Zero-Quantum Dephasing ***";
    obs_chp_shape            =? "CHIRP", ad_shape_names, help "shape pulse";
    obs_chp1_m_pulse         => 50[ms], (30[ms], 40[ms], 50[ms]), help "ZQfilter
pulse width";
    obs_chp1_m_fsweep        =? if obs_chp1_m_pulse = 30[ms]
                               then x_sweep * 2
                               else if obs_chp1_m_pulse = 40[ms]
                               then x_sweep * 2
                               else x_sweep * 2;
    obs_chp1_sweep_ppm       =? round (obs_chp1_m_fsweep * 1[Mppm] / obsfreq), help
"obs_chp1_ad_sweep_ppm";
    obs_chp1_chirp_smooth    =  10[%];
    obs_chp1_m_q0            =  11, help "q >= 5 On resonance adiabatic Quality
factor";
    obs_chp1_m_ph_rev        =  1, (1, -1), help "phase reverse";
    obs_chp1_b1_calc         =  sqrt (obs_chp1_m_q0 * obs_chp1_m_fsweep / (6.28319
* obs_chp1_m_pulse));
    obs_chp1_m_b1max         =? obs_chp1_b1_calc;
    obs_chp1_pw90            =  1 / (4 * obs_chp1_m_b1max);
```

```
    obs_chp1_atn_calc        =  if obs_atn + 20[dB] * log (obs_chp1_pw90 /
obs_pulse) < obs_atn + 6[dB]
                                then obs_atn + 6[dB]
                                else obs_atn + 20[dB] * log (obs_chp1_pw90 /
obs_pulse);
    obs_chp1_atn             =? obs_chp1_atn_calc;
    obs_chp2_m_pulse         => 30[ms], (30[ms], 40[ms], 50[ms]), help "ZQfilter
pulse width";
    obs_chp2_m_fsweep        =? if obs_chp2_m_pulse = 30[ms]
                                then x_sweep * 2
                                else if obs_chp2_m_pulse = 40[ms]
                                then x_sweep * 2
                                else x_sweep * 2;
    obs_chp2_sweep_ppm       =? round (obs_chp2_m_fsweep * 1[Mppm] / obsfreq), help
"obs_chirp2_ad_sweep_ppm";
    obs_chp2_chirp_smooth    =  10[%];
    obs_chp2_m_q0            =  11, help "q >= 5 On resonance adiabatic Quality
factor";
    obs_chp2_m_ph_rev        =  1, (1, -1), help "phase reverse";
    obs_chp2_b1_calc         =  sqrt (obs_chp2_m_q0 * obs_chp2_m_fsweep / (6.28319
* obs_chp2_m_pulse));
    obs_chp2_m_b1max         =? obs_chp2_b1_calc;
    obs_chp2_pw90            =  1 / (4 * obs_chp2_m_b1max);
    obs_chp2_atn_calc        =  if obs_atn + 20[dB] * log (obs_chp2_pw90 /
obs_pulse) < obs_atn + 6[dB]
                                then obs_atn + 6[dB]
                                else obs_atn + 20[dB] * log (obs_chp2_pw90 /
obs_pulse);
    obs_chp2_atn             =? obs_chp2_atn_calc;
    g_factor                 =  1, help "obs_gamma/1H_gamma";
    coil_factor              =  1, help "coil_length/20mm";
    f_factor_zqf1            =  obs_chp1_m_fsweep / 40[kHz], help "ref
10ppm*8/500MHz=40kHz";
    grad_zqf1_amp_calc       =  47[mT/m] * f_factor_zqf1 / coil_factor / g_factor,
help "gradient amplitude of grad_sl_ps";
    grad_zqf1_amp            =? grad_zqf1_amp_calc, help "Enter amplitude in
Tesla/meter units";
    f_factor_zqf2            =  obs_chp2_m_fsweep / 40[kHz], help "ref
10ppm*8/500MHz=40kHz";
    grad_zqf2_amp_calc       =  47[mT/m] * f_factor_zqf2 / coil_factor / g_factor,
help "gradient amplitude of grad_sl_ps";
    grad_zqf2_amp            =? grad_zqf2_amp_calc, help "Enter amplitude in
Tesla/meter units";
    grad_zqf1n_amp =? -grad_zqf1_amp;
    grad_zqf2n_amp =? -grad_zqf2_amp;

comment_7 =? "*** Pulse Delay ***";
    initial_wait     =  1[s];
    include "relaxation_delay_1d_calc";
    relaxation_delay => 2[s]; -- relaxation_delay_default, help "relaxation delay";
    repetition_time  =? relaxation_delay + x_acq_time, help
"relaxation_delay+x_acq_time";

comment_900              =? "*** lock hold ***";
    lock_hold                => TRUE, help "select TRUE or FALSE for lock hold";

    include "pulse";

    -- not optimized, taken from ref
    phase_90  = {16(0), 16(180)};
    phase_asr = {4(0), 4(90), 4(180), 4(270)};
    phase_3 = {2(0), 2(180)};
    phase_4 = {0, 180};
    phase_zero = {0};
    --scans      1   2
    phase_acq = {2(0, 180, 180, 0, 180, 0, 0, 180), 2(180, 0, 0, 180, 0, 180, 180,
0)};
```

```
    phase_ad1 = {0};
    phase_ad2 = {0}; -- {16(0),16(90)};
    phase_ad3 = {0};
    phase_ad4 = {0}; -- {4(0), 4(90)};

    phase_ad1r= {0};
    phase_ad2r= {0}; -- {8(0),8(90)};
    phase_ad3r= {0};
    phase_ad4r= {0}; -- {2(0), 2(90)};

module_config = "";

begin
    initial_wait;
    relaxation_delay;

    when lock_hold do
        on LOCKHOLD;
    end when;
    1[us];

    x_pulse, (obs.gate, obs.phs.phase_90, obs.atn.x_atn);

    loop gs_loop times
        parallel
            justify center
                obs_ad180_m_pulse, (obs.gate, obs.phs.phase_ad1,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180"});
            justify center
                grad_sl, (fgz.gate,
fgz.amp.grad_slPos_amp,fgz.shape.grad_shape_sl);
        end parallel;
        grad_recover_sl;

        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp1, fgz.shape.grad_shape);
            grad_recover;
        end when;
        parallel
            justify center
                obs_ad180_m_pulse, (obs.gate, obs.phs.phase_ad2,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180"});
            justify center
                grad_sl, (fgz.gate,
fgz.amp.grad_slNeg_amp,fgz.shape.grad_shape_sl);
        end parallel;
        grad_recover_sl;
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp1, fgz.shape.grad_shape);
            grad_recover;
        end when;

        parallel
            justify center
                obs_ad180_m_pulse, (obs.gate, obs.phs.phase_ad3,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180"});
            justify center
                grad_sl, (fgz.gate,
fgz.amp.grad_slPos_amp,fgz.shape.grad_shape_sl);
        end parallel;
        grad_recover_sl;

        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp2, fgz.shape.grad_shape);
            grad_recover;
        end when;
        parallel
            justify center
```

```
                obs_ad180_m_pulse, (obs.gate, obs.phs.phase_ad4,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180"});
            justify center
                grad_sl, (fgz.gate,
fgz.amp.grad_slNeg_amp,fgz.shape.grad_shape_sl);
        end parallel;
        grad_recover_sl;
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp2, fgz.shape.grad_shape);
            grad_recover;
        end when;
    end loop;

    grad_1, (fgz.gate, fgz.amp.grad_1_amp, fgz.shape.grad_shape);
    grad_recover;
    obs_asr180, (obs.gate, obs.phs.phase_asr, obs.atn.obs_asr180_atn,
obs.shape.obs_asr180_shape);
    grad_1, (fgz.gate, fgz.amp.grad_1m_amp, fgz.shape.grad_shape);
    grad_recover;

    loop gs_loop times
        parallel
            justify center
                obs_ad180r_m_pulse, (obs.gate, obs.phs.phase_ad1r,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180r"});
            justify center
                grad_sl, (fgz.gate,
fgz.amp.grad_slPos_amp,fgz.shape.grad_shape_sl);
        end parallel;
        grad_recover_sl;

        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp3, fgz.shape.grad_shape);
            grad_recover;
        end when;
        parallel
            justify center
                obs_ad180r_m_pulse, (obs.gate, obs.phs.phase_ad2r,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180r"});
            justify center
                grad_sl, (fgz.gate,
fgz.amp.grad_slNeg_amp,fgz.shape.grad_shape_sl);
        end parallel;
        grad_recover_sl;
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp3, fgz.shape.grad_shape);
            grad_recover;
        end when;

        parallel
            justify center
                obs_ad180r_m_pulse, (obs.gate, obs.phs.phase_ad3r,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180r"});
            justify center
                grad_sl, (fgz.gate,
fgz.amp.grad_slPos_amp,fgz.shape.grad_shape_sl);
        end parallel;
        grad_recover_sl;

        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp4, fgz.shape.grad_shape);
            grad_recover;
        end when;
        parallel
            justify center
                obs_ad180r_m_pulse, (obs.gate, obs.phs.phase_ad4r,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180r"});
            justify center
```

```
                grad_sl, (fgz.gate,
fgz.amp.grad_slNeg_amp,fgz.shape.grad_shape_sl);
        end parallel;
        grad_recover_sl;
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp4, fgz.shape.grad_shape);
            grad_recover;
        end when;
    end loop;

    x_pulse, (obs.gate, obs.phs.phase_3, obs.atn.x_atn);
    grad_2, (fgz.gate, fgz.amp.grad_2_amp, fgz.shape.grad_shape);
    grad_recover;

    10[us];
    on (FGZ.GATE, FGZ.AMP.grad_zqf1_amp);
    obs_chp1_m_pulse, (OBS.GATE, OBS.PHS.phase_zero, OBS.ATN.obs_chp1_atn,
OBS.SHAPE.{obs_chp_shape, "obs_chp1"});
    off (FGZ.GATE, FGZ.AMP.grad_zqf1_amp);
    grad_recover;

    hG_ramp_pulse,( obs.gate, obs.phs.phase_zero, obs.atn.spinlock_power,
obs.shape."gauss_lhalf_90", obs.offset.rpos2);
    spinlock_pulse,( obs.gate, obs.phs.phase_zero, obs.atn.spinlock_power,
obs.offset.rpos2);
    hG_ramp_pulse,( obs.gate, obs.phs.phase_zero, obs.atn.spinlock_power,
obs.shape."gauss_rhalf_90", obs.offset.rpos2);
    4[us];
    hG_ramp_pulse,( obs.gate, obs.phs.phase_zero, obs.atn.spinlock_power,
obs.shape."gauss_lhalf_90", obs.offset.rpos1);
    spinlock_pulse,( obs.gate, obs.phs.phase_zero, obs.atn.spinlock_power,
obs.offset.rpos1);
    hG_ramp_pulse,( obs.gate, obs.phs.phase_zero, obs.atn.spinlock_power,
obs.shape."gauss_rhalf_90", obs.offset.rpos1);

    grad_4, (fgz.gate, fgz.amp.grad_4_amp, fgz.shape.grad_shape);
    grad_recover;
    on (FGZ.GATE, FGZ.AMP.grad_zqf2_amp);
    obs_chp2_m_pulse, (OBS.GATE, OBS.PHS.phase_zero, OBS.ATN.obs_chp2_atn,
OBS.SHAPE.{obs_chp_shape, "obs_chp2"});
    off (FGZ.GATE, FGZ.AMP.grad_zqf2_amp);
    grad_recover_zqs;

    x_pulse, (obs.gate, obs.phs.phase_4, obs.atn.x_atn);
    delay6;
    x_pulse*2, (obs.gate, obs.phs.phase_zero, obs.atn.x_atn);
    grad_6, (fgz.gate, fgz.amp.grad_6_amp, fgz.shape.grad_shape);
    grad_recover;


    acq( dead_time, delay, phase_acq );
    10[us];
    when lock_hold do
        off LOCKHOLD;
    end when;
    10[us];

       relaxation_delay;

    when lock_hold do
        on LOCKHOLD;
    end when;
    1[us];

    x_pulse, (obs.gate, obs.phs.phase_90, obs.atn.x_atn);

    loop gs_loop times
        parallel
```

```
            justify center
                obs_ad180_m_pulse, (obs.gate, obs.phs.phase_ad1,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180"});
            justify center
                grad_sl, (fgz.gate,
fgz.amp.grad_slNeg_amp,fgz.shape.grad_shape_sl);
        end parallel;
        grad_recover_sl;

        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp1n, fgz.shape.grad_shape);
            grad_recover;
        end when;
        parallel
            justify center
                obs_ad180_m_pulse, (obs.gate, obs.phs.phase_ad2,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180"});
            justify center
                grad_sl, (fgz.gate,
fgz.amp.grad_slPos_amp,fgz.shape.grad_shape_sl);
        end parallel;
        grad_recover_sl;
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp1n, fgz.shape.grad_shape);
            grad_recover;
        end when;

        parallel
            justify center
                obs_ad180_m_pulse, (obs.gate, obs.phs.phase_ad3,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180"});
            justify center
                grad_sl, (fgz.gate,
fgz.amp.grad_slNeg_amp,fgz.shape.grad_shape_sl);
        end parallel;
        grad_recover_sl;

        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp2n, fgz.shape.grad_shape);
            grad_recover;
        end when;
        parallel
            justify center
                obs_ad180_m_pulse, (obs.gate, obs.phs.phase_ad4,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180"});
            justify center
                grad_sl, (fgz.gate,
fgz.amp.grad_slPos_amp,fgz.shape.grad_shape_sl);
        end parallel;
        grad_recover_sl;
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp2n, fgz.shape.grad_shape);
            grad_recover;
        end when;
    end loop;

    grad_1, (fgz.gate, fgz.amp.grad_1m_amp, fgz.shape.grad_shape);
    grad_recover;
    obs_asr180, (obs.gate, obs.phs.phase_asr, obs.atn.obs_asr180_atn,
obs.shape.obs_asr180_shape);
    grad_1, (fgz.gate, fgz.amp.grad_1_amp, fgz.shape.grad_shape);
    grad_recover;

    loop gs_loop times
        parallel
            justify center
                obs_ad180r_m_pulse, (obs.gate, obs.phs.phase_ad1r,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180r"});
```

```
            justify center
                grad_sl, (fgz.gate,
fgz.amp.grad_slNeg_amp,fgz.shape.grad_shape_sl);
        end parallel;
        grad_recover_sl;

        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp3n, fgz.shape.grad_shape);
            grad_recover;
        end when;
        parallel
            justify center
                obs_ad180r_m_pulse, (obs.gate, obs.phs.phase_ad2r,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180r"});
            justify center
                grad_sl, (fgz.gate,
fgz.amp.grad_slPos_amp,fgz.shape.grad_shape_sl);
        end parallel;
        grad_recover_sl;
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp3n, fgz.shape.grad_shape);
            grad_recover;
        end when;

        parallel
            justify center
                obs_ad180r_m_pulse, (obs.gate, obs.phs.phase_ad3r,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180r"});
            justify center
                grad_sl, (fgz.gate,
fgz.amp.grad_slNeg_amp,fgz.shape.grad_shape_sl);
        end parallel;
        grad_recover_sl;

        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp4n, fgz.shape.grad_shape);
            grad_recover;
        end when;
        parallel
            justify center
                obs_ad180r_m_pulse, (obs.gate, obs.phs.phase_ad4r,
obs.atn.obs_ad180_atn, obs.shape.{obs_ad180_shape,"obs_ad180r"});
            justify center
                grad_sl, (fgz.gate,
fgz.amp.grad_slPos_amp,fgz.shape.grad_shape_sl);
        end parallel;
        grad_recover_sl;
        when grad_ctp do
            grad_ctp_sl, (fgz.gate, fgz.amp.grad_ctp_amp4n, fgz.shape.grad_shape);
            grad_recover;
        end when;
    end loop;

    x_pulse, (obs.gate, obs.phs.phase_3, obs.atn.x_atn);
    grad_2, (fgz.gate, fgz.amp.grad_2n_amp, fgz.shape.grad_shape);
    grad_recover;

    10[us];
    on (FGZ.GATE, FGZ.AMP.grad_zqf1n_amp);
    obs_chp1_m_pulse, (OBS.GATE, OBS.PHS.phase_zero, OBS.ATN.obs_chp1_atn,
OBS.SHAPE.{obs_chp_shape, "obs_chp1"});
    off (FGZ.GATE, FGZ.AMP.grad_zqf1n_amp);
    grad_recover;

    hG_ramp_pulse,( obs.gate, obs.phs.phase_zero, obs.atn.spinlock_power,
obs.shape."gauss_lhalf_90", obs.offset.rpos2);
    spinlock_pulse,( obs.gate, obs.phs.phase_zero, obs.atn.spinlock_power,
obs.offset.rpos2);
```

```
    hG_ramp_pulse,( obs.gate, obs.phs.phase_zero, obs.atn.spinlock_power,
obs.shape."gauss_rhalf_90", obs.offset.rpos2);
    4[us];
    hG_ramp_pulse,( obs.gate, obs.phs.phase_zero, obs.atn.spinlock_power,
obs.shape."gauss_lhalf_90", obs.offset.rpos1);
    spinlock_pulse,( obs.gate, obs.phs.phase_zero, obs.atn.spinlock_power,
obs.offset.rpos1);
    hG_ramp_pulse,( obs.gate, obs.phs.phase_zero, obs.atn.spinlock_power,
obs.shape."gauss_rhalf_90", obs.offset.rpos1);

    grad_4, (fgz.gate, fgz.amp.grad_4n_amp, fgz.shape.grad_shape);
    grad_recover;
    on (FGZ.GATE, FGZ.AMP.grad_zqf2n_amp);
    obs_chp2_m_pulse, (OBS.GATE, OBS.PHS.phase_zero, OBS.ATN.obs_chp2_atn,
OBS.SHAPE.{obs_chp_shape, "obs_chp2"});
    off (FGZ.GATE, FGZ.AMP.grad_zqf2n_amp);
    grad_recover_zqs;

    x_pulse, (obs.gate, obs.phs.phase_4, obs.atn.x_atn);
    delay6;
    x_pulse*2, (obs.gate, obs.phs.phase_zero, obs.atn.x_atn);
    grad_6, (fgz.gate, fgz.amp.grad_6n_amp, fgz.shape.grad_shape);
    grad_recover;

    acq( dead_time, delay, phase_acq );
    10[us];
    when lock_hold do
        off LOCKHOLD;
    end when;
    10[us];

end pulse;
```