ISMRM 2019

Community Software Tools Demos

KS FOUNDATION

The KS Foundation Abstraction Layer for EPIC

Overview of the gradient echo tutorial sequence (ksgre_tutorial.e)

- **ksgre_tutorial.e**
  - Top-level file containing only the overall structure with
    - Mandatory EPIC sections and hooks in general
    - Order of execution of functions based on UI events etc

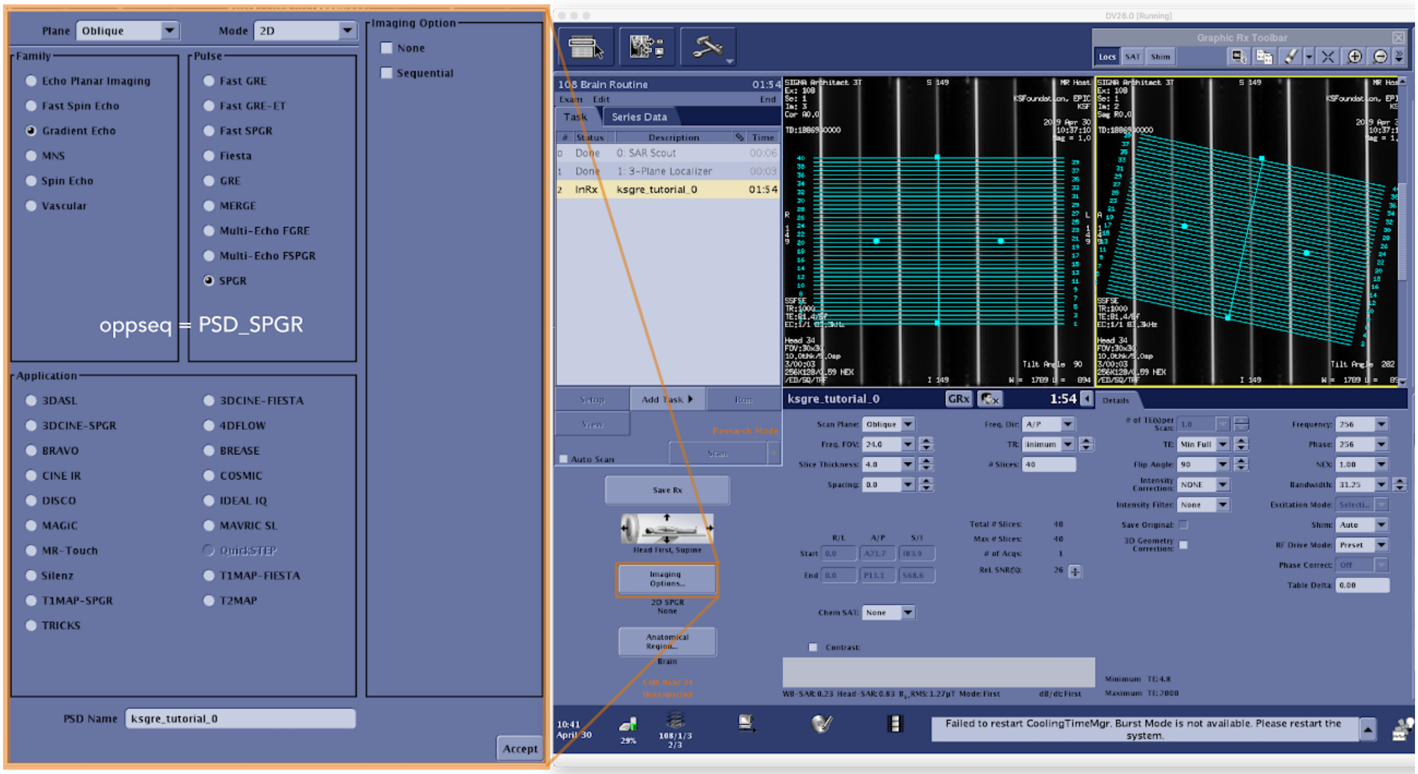
- **ksgre_tutorial_implementation.e**
  - Everytime a UI button is changed, **`my_cveval()`** is called. This contains a chain of events to setup (not play) the sequence

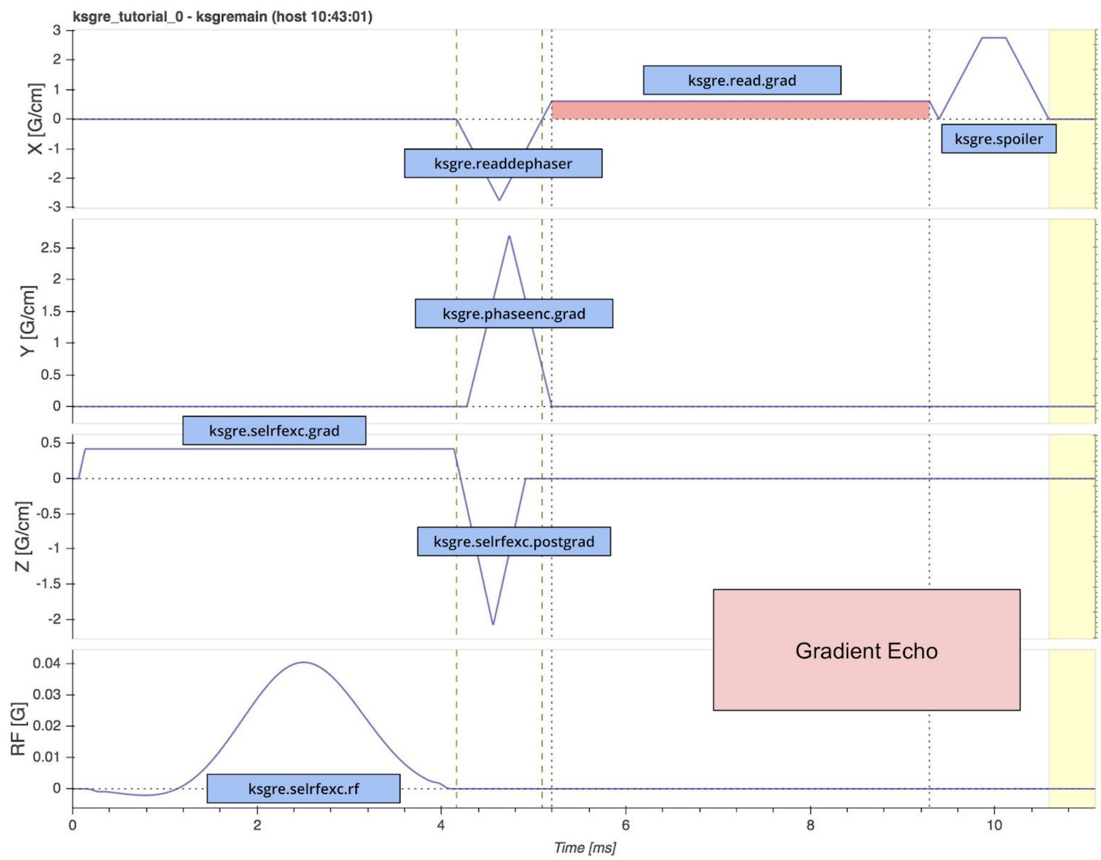
  - Designing gradients and RF pulses (a.k.a. sequence objects). See **`ksgre_eval_setupobjects()`**
    - ks_eval_*** (KS Foundation)
    - ksgre_eval_*** (sequence specific)

  - Placing out sequence objects on the sequence boards (X,Y,Z,RF, ...) in the sequence generating function. See **`ksgre_pg()`**

  - The scan loop has as a standardized structure in all KS Foundation psds. This makes scan execution scalable and modular:
  - ksgre_scan_scanloop()
    - Data for the entire scan, which in turn calls
  - ksgre_scan_acqloop()
    - All data for one set of slices that fit within one TR (one acquisition), which in turn calls
  - ksgre_scan_sliceloop()
    - One set of slices that fit within one TR played out for one shot, which in turn calls
  - ksgre_scan_coreslice()
    - One slice playout, with optional other sequence modules, which in turn calls
  - ksgre_scan_seqstate()
    - Updates to gradient amplitudes and RF freq/phase etc

Exercise 0 - Starting point of the tutorial sequence - *ksgre_tutorial.e*





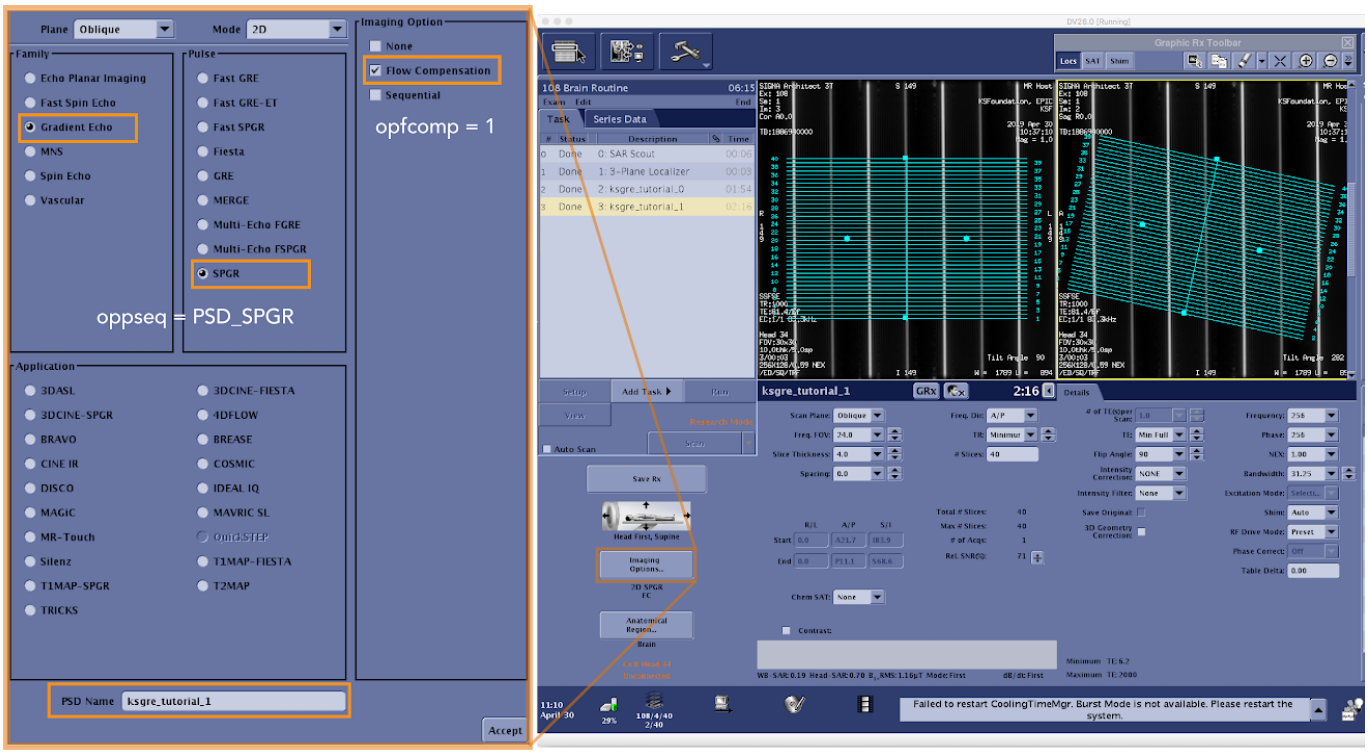
Exercise 0 - Starting point summary:

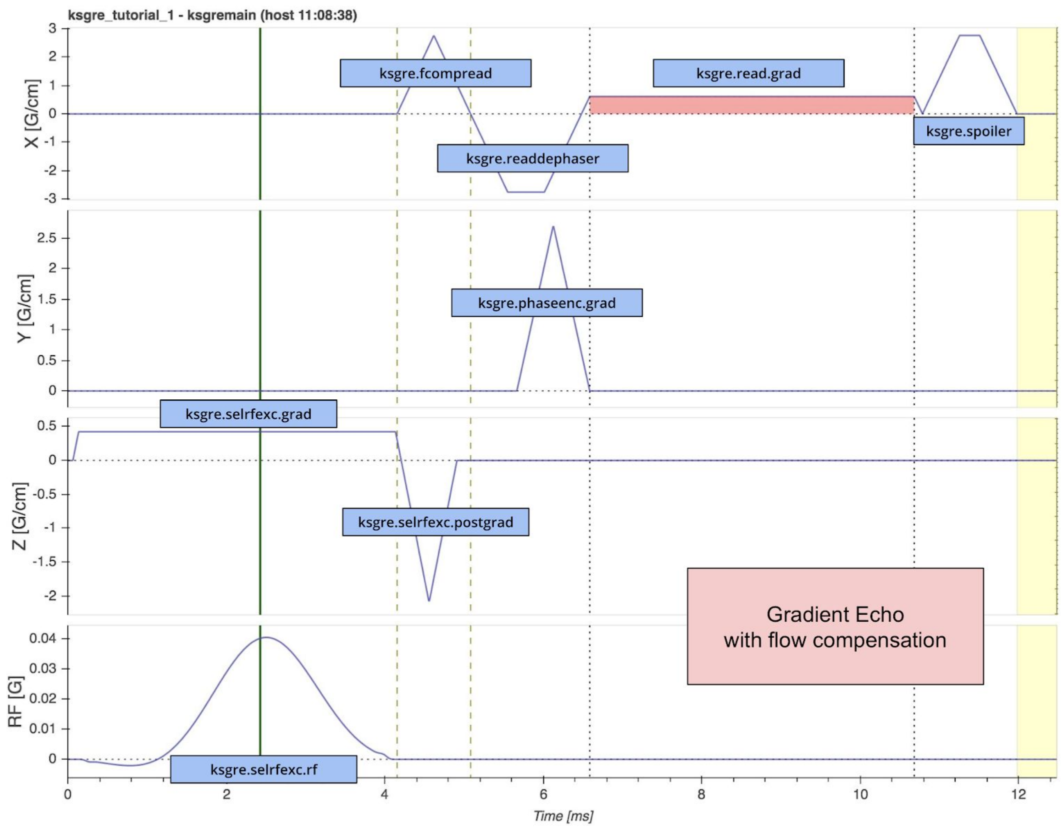
(CMD/CTRL click on links to open new tab)

- Code for ksgre_tutorial.e
- Code for ksgre_tutorial_implementation.e
- Interactive sequence plot (auto-generated using KS Foundation's plot functions)
- Sequence plot with annotations
- Slice-time plot for sequence (auto-generated using KS Foundation's plot functions)

Exercise 1 - Flow comp in the readout direction (XGRAD)

Below the sequence UI selections to be made are highlighted. On the XGRAD board, a new gradient is added.





Exercise 1 summary:

(CMD/CTRL click on links to open new tab)

- See all changes for exercise 1
- Interactive sequence plot (auto-generated using KS Foundation's plot functions)
- Sequence plot with annotations
- Slice-time plot for sequence (auto-generated using KS Foundation's plot functions)

Add flow comp in the X (read) direction to ksgre_tutorial.e if the flowcomp button is selected.

Normally, there is a static read dephaser before the readout trapezoid with a negative area corresponding to the area to k-space center on the readout trapezoid. To also rephase spins with constant velocity (linear flow), i.e. to do flow compensation, one more gradient needs to be added before this read dephaser. The area of the new gradient must be equal to the original read dephaser gradient, but positive. Also, the original read dephaser gradient (now between the new gradient and the readout), should double in size.

Note: Since readouts can sometimes be played out as partial Fourier, it is not certain that the center of k-space is reached at the center of the readout gradient. Therefore, use `ksgre.read.area2center` instead of `ksgre.read.area/2` to define the needed dephaser areas. The field `.area2center` is set by `ks_eval_readtrap()`.

- Step 1: The flowcomp button for the UI is present or not at the sequence selection page, based on the settings in the psd. To enable this button, add

```
PSD_IOPT_FLOW_COMP, /* opfcomp */
```

to the `int sequence_iopts[] = {...}` struct. If this button is pressed, `opfcomp = 1`.

```
/*******************************************************************************************************
 *******************************************************************************************************
 *
 *  ksgre_tutorial_implementation.e: CVINIT
 *
 *******************************************************************************************************
 ******************************************************************************************************/


int sequence_iopts[] = {
  PSD_IOPT_SEQUENTIAL, /* opirmode */
  PSD_IOPT_FLOW_COMP, /* opfcomp */ /* Exercise 1: Flow comp in the readout direction */
#ifdef UNDEF
  PSD_IOPT_ARC, /* oparc */
  PSD_IOPT_ASSET, /* opasset */
  PSD_IOPT_EDR, /* opptsize */
  PSD_IOPT_DYNPL, /* opdynaplan */
  PSD_IOPT_ZIP_512, /* opzip512 */
  PSD_IOPT_IR_PREP, /* opirprep */
  PSD_IOPT_MILDNOTE, /* opsilent */
  PSD_IOPT_ZIP_1024, /* opzip1024 */
  PSD_IOPT_SLZIP_X2, /* opzip2 */
  PSD_IOPT_MPH, /* opmph */
  PSD_IOPT_NAV, /* opnav */
#endif
};
```

- Step 2: Add a new KS_TRAP to KSGRE_SEQUENCE named fcompread. Also update KSGRE_INIT_SEQUENCE.

```
/** @brief #### `typedef struct` holding all all gradients and waveforms for the ksgre sequence
*/
typedef struct KSGRE_SEQUENCE {
  KS_SEQ_CONTROL seqctrl;
  KS_READTRAP read;
  KS_TRAP readdephaser;
  KS_TRAP fcompread; /* Exercise 1: Flow comp in the readout direction */
  KS_PHASER phaseenc;
  KS_TRAP spoiler;
  KS_SELRF selrfexc;
} KSGRE_SEQUENCE;
#define KSGRE_INIT_SEQUENCE {KS_INIT_SEQ_CONTROL, KS_INIT_READTRAP, KS_INIT_TRAP, KS_INIT_TRAP, \
                             KS_INIT_PHASER, KS_INIT_TRAP, KS_INIT_SELRF};
```

Set up the correct area for the new gradient in `ksgre_eval_setupobjects()`. Also, modify the `ksgre.readdephaser.area` accordingly. Do this in the "Readout dephaser" section of `ksgre_eval_setupobjects()`.

```
/*******************************************************************************************************
 *  Readout dephaser
 *******************************************************************************************************/

/* Exercise 1: Flow comp in the readout direction */
if (opfcomp == TRUE) {
  ksgre.readdephaser.area = -2.0 * ksgre.read.area2center;
  ksgre.fcompread.area = ksgre.read.area2center;

  status = ks_eval_trap(&ksgre.fcompread, "fcompread");
  if (status != SUCCESS) return status;
}
else {
  ksgre.readdephaser.area = -ksgre.read.area2center;
  /* Reset the KS_TRAP to KS_INIT_TRAP if opfcomp is toogled off again
     KS_INIT_TRAP (KSFoundation.h) sets all relevant fields to 0, including
     .duration, which is also an indicator for "don't use" by ks_pg_trap() */
  ks_init_trap(&ksgre.fcompread);
}

status = ks_eval_trap(&ksgre.readdephaser, "readdephaser");
if (status != SUCCESS) return status;
```

- Step 3: Add the new `fcompread` gradient in `ksgre_pg()`. This is the place where the gradient is positioned in time and on which gradient board. We use tmploc (KS_SEQLOC) to place all KS_** objects in a sequence.

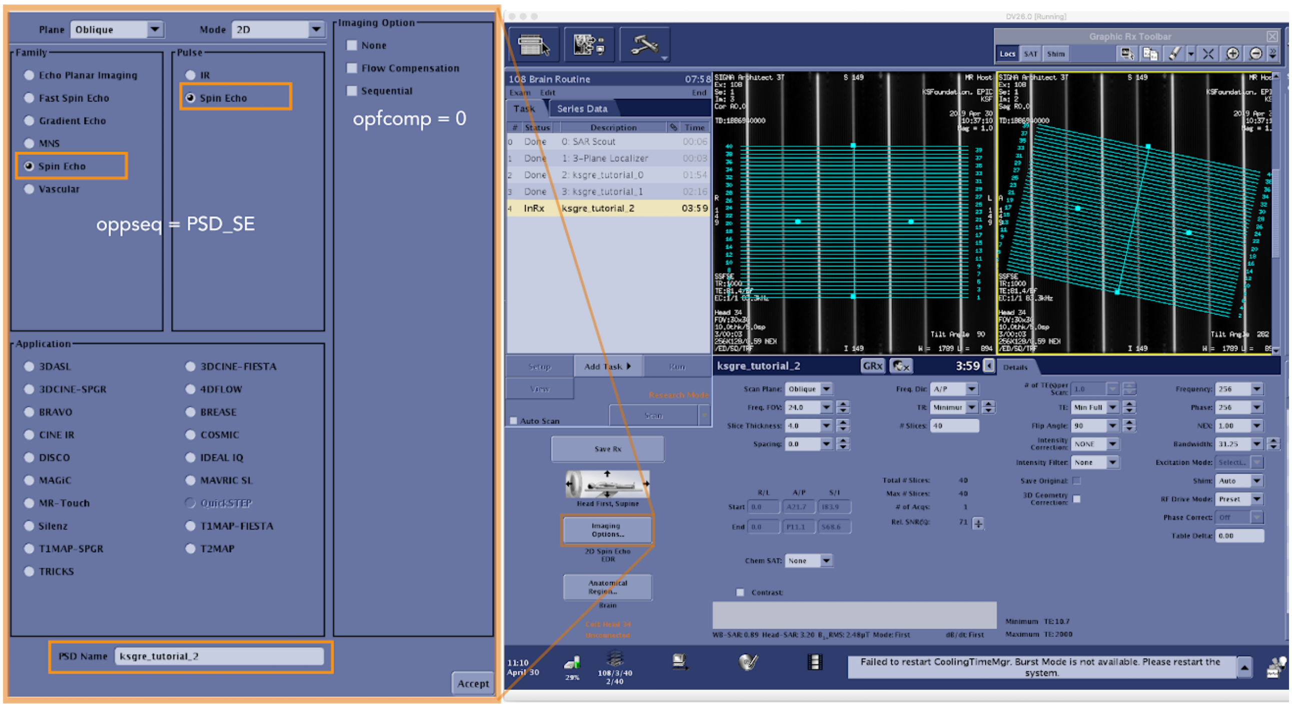
```
/*******************************************************************************************************
 *  Readout dephaser
 *******************************************************************************************************/

tmploc.pos = readstart_pos - ksgre.readdephaser.duration;
tmploc.board = XGRAD;

status = ks_pg_trap(&ksgre.readdephaser, tmploc, &ksgre.seqctrl);
if (status != SUCCESS) return status;

 /* Exercise 1: Flow comp in the readout direction.
    Note that we don't need if (opfcomp) {} here, since ks_pg_trap() will return
    quietly without errors doing nothing if ksgre.fcompread.duration = 0,
    i.e. not set up using ks_eval_trap() */
tmploc.pos = readstart_pos - ksgre.readdephaser.duration - ksgre.fcompread.duration;
status = ks_pg_trap(&ksgre.fcompread, tmploc, &ksgre.seqctrl);
if (status != SUCCESS) return status;
```

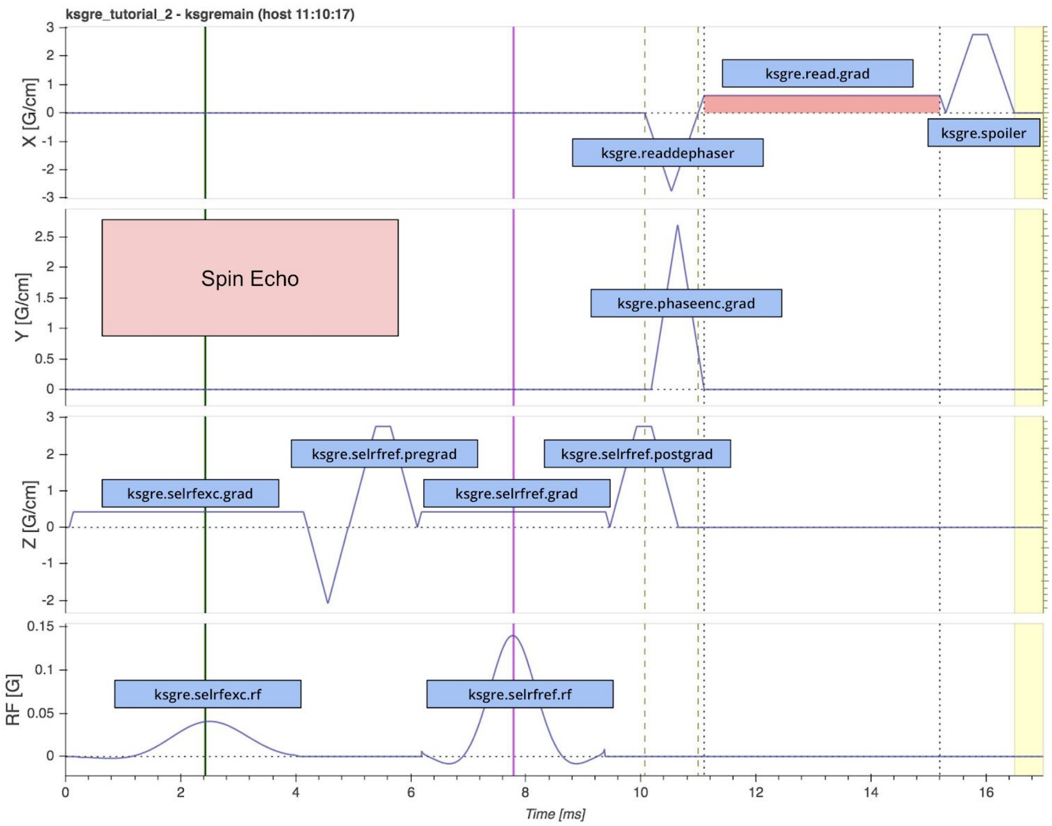
- Step 4: Make sure the TE cannot be so short that the first gradient on XGRAD is played out during the RF excitation. This is done by modifying `avminte` in `ksgre_eval_TErange()`. Make sure you add it correctly to avoid unnecessary gap in the sequence. Select TE = MinFull and check "TE: xx" at the bottom of the UI - with and without "Flowcomp" check box selected

```
STATUS ksgre_eval_TErange() {

  /* Minimum TE */
  avminte = ksgre.selrfexc.rf.iso2end;
  avminte += IMax(3, \
        /* Exercise 1: fcompread */ ksgre.fcompread.duration + ksgre.readdephaser.duration + ksgre.read.acqdelay, \
        ksgre.phaseenc.grad.duration, \
        ksgre.selrfexc.grad.ramptime + ksgre.selrfexc.postgrad.duration);
  avminte += ksgre.read.time2center - ksgre.read.acqdelay; /* from start of acq win to k-space center */
  avminte = RUP_FACTOR(avminte + 16us, 16); /* add 16us margin and round up to make time divisible by 16us */
```

Exercise 2 - Make a Spin-Echo sequence

Modify `ksgre_tutorial.e` to allow it to also be run as a Spin-Echo sequence.





Exercise 2 summary:

(CMD/CTRL click on links to open new tab)

- See all changes for exercise 2
- Interactive sequence plot (auto-generated using KS Foundation's plot functions)
- Sequence plot with annotations
- Slice-time plot for sequence (auto-generated using KS Foundation's plot functions)

Let the sequence run as GRE or SE based on the "Spin Echo" and "Gradient Echo" selection in the UI. If "Spin Echo" is selected (twice), `oppseq = PSD_SE`, otherwise `oppseq = PSD_SPGR` or `PSD_GRE` depending on the GRE sub type.

The following steps will be necessary:

- Step 1: Add an `KS_SELRF` pulse to `KSGRE_SEQUENCE`. Also update `KSGRE_INIT_SEQUENCE`. Call the new pulse `selrfref`. Set `acq_type = TYPSPIN` in `ksgre_init_UI()` if `oppseq == PSD_SE`.

```
/** @brief #### `typedef struct` holding all all gradients and waveforms for the ksgre sequence
*/
typedef struct KSGRE_SEQUENCE {
  KS_SEQ_CONTROL seqctrl;
  KS_READTRAP read;
  KS_TRAP readdephaser;
  KS_TRAP fcompread; /* Exercise 1: Flow comp in the readout direction */
  KS_PHASER phaseenc;
  KS_TRAP spoiler;
  KS_SELRF selrfexc;
  KS_SELRF selrfref; /* Exercise 2: Spin Echo */
} KSGRE_SEQUENCE;
#define KSGRE_INIT_SEQUENCE {KS_INIT_SEQ_CONTROL, KS_INIT_READTRAP, KS_INIT_TRAP, KS_INIT_TRAP, \
                             KS_INIT_PHASER, KS_INIT_TRAP, KS_INIT_SELRF, KS_INIT_SELRF};
```

```
STATUS ksgre_init_UI(void) {

  /* Menus and button content
     See epic.h or ksgre.e for more pi*** CVs to use to set up the user menus and buttons */

  /* Exercise 2: Spin Echo. Set GE's global variable 'acq_type' as some functions look at it */
  if (oppseq == PSD_SE) {
    /* Spin Echo Type of sequence */
    acq_type = TYPSPIN;
  } else {
    /* Gradient Echo Type of sequence */
    acq_type = TYPGRAD; /* loadrheader.e rheaderinit: sets eeff = 1 */
  }
```

- Step 2: Set up the new refocusing RF pulse in `ksgre_eval_setupobjects()` using the the `ref_fse1601` refocusing pulse in `KSFoundation_GERF.h`. Also change the excitation pulse to `exc_fse90` if Spin Echo is selected.

```
STATUS ksgre_eval_setupobjects() {
  STATUS status;

  /*******************************************************************************************************
   *  RF Excitation
   *******************************************************************************************************/

  /* RF pulse choice (KSFoundation_GERF.h) */
  ksgre.selrfexc.rf = exc_fse90;
  ksgre.selrfexc.rf.flip = opflip;
  ksgre.selrfexc.slthick = opslthick / ksgre_gscalerfexc;

  /* selective RF excitation */
  status = ks_eval_selrf(&ksgre.selrfexc, "rfexc");
  if (status != SUCCESS) return status;


  /*******************************************************************************************************
   *  Exercise 2: (Spin Echo) RF Refocusing
   *******************************************************************************************************/

  if (acq_type == TYPSPIN) {

    /* RF pulse choice (KSFoundation_GERF.h) */
    ksgre.selrfref.rf = ref_fse1601;

    ksgre.selrfref.rf.flip = 180;
    ksgre.selrfref.slthick = opslthick / ksgre_gscalerfexc;

    /* selective RF refocusing */
    status = ks_eval_selrf(&ksgre.selrfref, "rfref");
    if (status != SUCCESS) return status;

  } else {

    ks_init_selrf(&ksgre.selrfref); /* wipe this KS_SELRF if not Spin Echo */

  }

  /*******************************************************************************************************
   *  Readout
   *******************************************************************************************************/
```

- Step 3: Hide the flip angle menu and set opflip = 90 if Spin Echo in ksgre_init_UI().

```

  /* show flip angle menu only if GRE (Exercise 4: Spin Echo) */
  if (acq_type == TYPGRAD) {
    pifanub = 2;
    cvdef(opflip, 5); /* default low FA */
  } else {
    pifanub = 0;
    cvdef(opflip, 90); /* default 90 FA for Spin Echo */
  }
  opflip = _opflip.defval;

```

- Step 4: Add a `ks_pg_selrf()` call in `ksgre_pg()` at opte/2 from the isocenter of the excitation RF pulse. Note that half of the KS_SELRF total duration must be subtracted, i.e. the sum of:

- Half the refocusing pulse (`ksgre.selrfref.rf.start2iso` or `ksgre_selrfref.grad.plateautime/2`)
- The duration of the leading ramp time (`ksgre.selrfref.grad.ramptime`)
- The duration of the left crusher (`ksgre.selrfref.pregrad.duration`)

```
/*******************************************************************************************************
 *  RF Excitation
 *******************************************************************************************************/
tmploc.pos = RUP_GRD(KS_RFSSP_PRETIME);
tmploc.board = ZGRAD;

/* N.B.: ks_pg_selrf()->ks_pg_rf() detects that ksgre.selrfexc is an excitation pulse
   (ksgre.selrfexc.rf.role = KS_RF_ROLE_EXC) and will also set ksgre.seqctrl.momentstart
   to the absolute position in [us] of the isocenter of the RF excitation pulse */
status = ks_pg_selrf(&ksgre.selrfexc, tmploc, &ksgre.seqctrl);
if (status != SUCCESS) return status;


/*******************************************************************************************************
 *  RF Refocusing (Exercise 2)
 *******************************************************************************************************/
tmploc.pos = ksgre.seqctrl.momentstart + opte/2 -  \
             ksgre.selrfref.rf.start2iso - ksgre.selrfref.grad.ramptime - ksgre.selrfref.pregrad.duration;
tmploc.board = ZGRAD;

status = ks_pg_selrf(&ksgre.selrfref, tmploc, &ksgre.seqctrl);
if (status != SUCCESS) return status;

```

- Step 5: Find the proper value for `avminte` for Spin Echo. Modify `ksgre_eval_TErange()` to balance the refocusing pulse in between the excitation and k-space center and update avminte. Note that you need to a) sum all times between excitation center and center of refocusing pulse and b) sum all times between the refocusing pulse and the center of k-space. Then take the highest value of those sums and multiply by 2 to get minimum TE. Note that IMax(3, bla1, bla2, bla3) or IMax(2, bla1, bla2) is necessary to not sum up gradients that may overlap on different boards (XGRAD, YGRAD, ZGRAD).

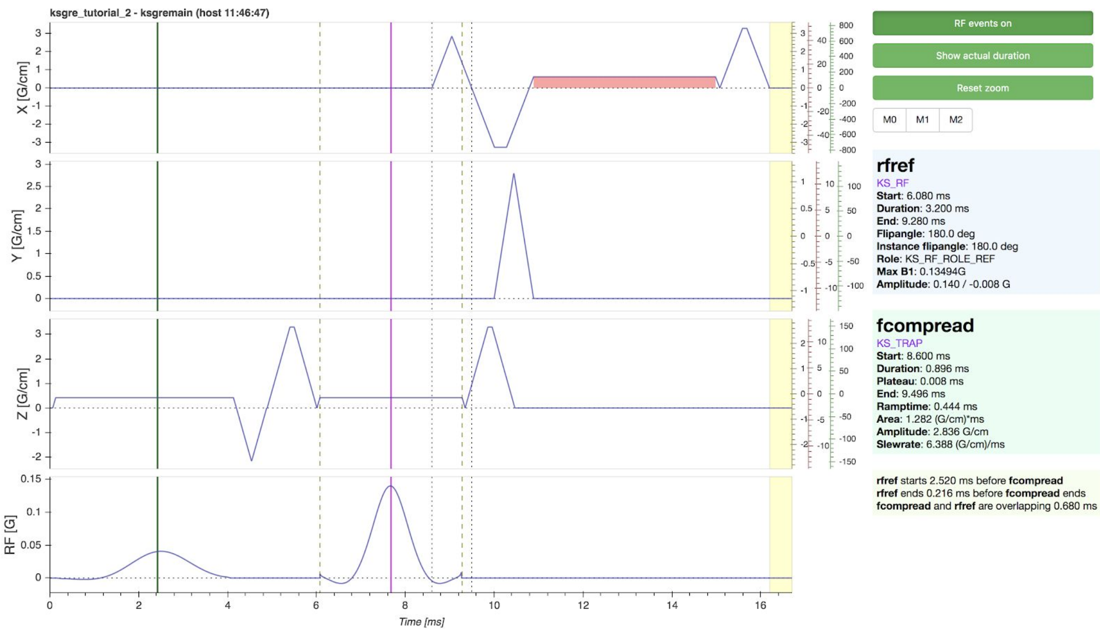
```
STATUS ksgre_eval_TErange() {

  /* Minimum TE */

  /* Exercise 2: Different minTE calcs for GRE and SE */
  if (acq_type == TYPGRAD) {

    avminte = ksgre.selrfexc.rf.iso2end;
    avminte += IMax(3, \
          /* Exercise 1: fcompread */ ksgre.fcompread.duration + ksgre.readdephaser.duration + ksgre.read.acqdelay, \
          ksgre.phaseenc.grad.duration, \
          ksgre.selrfexc.grad.ramptime + ksgre.selrfexc.postgrad.duration);
    avminte += ksgre.read.time2center - ksgre.read.acqdelay; /* from start of acq win to k-space center */
    /* Note: ksgre.read.acqdelay = ksgre.read.grad.ramptime normally, as long we don't use rampsampling */

  } else {
    /* Exercise 2 */
    int min90_180, min180_echo;

    /* min90_180: Minimum time needed between the iso point of the excitation pulse (TE = 0) and the iso point of the RF refocusing pulse (TE/2) */
    min90_180  = ksgre.selrfexc.rf.iso2end + ksgre.selrfexc.grad.ramptime + ksgre.selrfexc.postgrad.duration; /* excitation (Z) */
    min90_180 += ksgre.selrfref.pregrad.duration + ksgre.selrfref.grad.ramptime + ksgre.selrfref.rf.start2iso; /* refocusing incl left crusher (Z) */

    /* min180_echo: Minimum time needed between the iso point of the RF refocusing pulse (TE/2) and the k-space center (TE) at the center of the readout */
    min180_echo = ksgre.selrfref.rf.iso2end; /* center of RF refocusing pulse to its end */
    min180_echo += IMax(3,
                     ksgre.readdephaser.duration + ksgre.read.acqdelay /* X */,
                     ksgre.phaseenc.grad.duration /* Y */,
                     ksgre.selrfref.grad.ramptime + ksgre.selrfref.postgrad.duration /* Z */);
    min180_echo += ksgre.read.time2center - ksgre.read.acqdelay; /* time from start of plateau to k-space center */
    /* Note: ksgre.read.acqdelay = ksgre.read.grad.ramptime normally, as long we don't use rampsampling */

    avminte = IMax(2, min90_180, min180_echo) * 2;

  }
  avminte = RUP_FACTOR(avminte + 16us, 16); /* add 16us margin and round up to make time divisible by 16us */
```

- Step 6: Setting up slice location for the refocusing pulse in scan. For this task, do the necessary changes in `ksgre_scan_seqstate()`. Note that the RF phase needs to be 90 degrees between the excitation and refocusing RF pulses. Since the excitation and receiver phase are already in sync (rfphase = 0), just change the rf refocusing phase.

```
STATUS ksgre_scan_seqstate(SCAN_INFO slice_info, int kyview) {
  float rfphase = 0;

  /* rotate the slice plane */
  ks_scan_rotate(slice_info);

  /* RF frequency & phase */
  ks_scan_rf_on(&ksgre.selrfexc.rf, INSTRALL);
  ks_scan_selrf_setfreqphase(&ksgre.selrfexc, 0 /* instance */, slice_info, rfphase);

  /* Exercice 2 */
  ks_scan_selrf_setfreqphase(&ksgre.selrfref, 0 /* instance */, slice_info, rfphase + 90);


  /* FOV offsets (by changing freq/phase of ksgre.read) */
  ks_scan_offsetfov(&ksgre.read, INSTRALL, slice_info, kyview, opphasefov, rfphase);

  /* phase enc amp */
  ks_scan_phaser_toline(&ksgre.phaseenc,   0, kyview);

  return SUCCESS;

} /* ksgre_scan_seqstate() */
```

Sequence gradient overlap demo

Note that for Spin Echo mode and Flow Compensation, `ksgre_eval_TErange()` does not have code to handle minimum TE correctly. Plotting the Spin Echo sequence, with Flow Compensation on, looks then like this (note the gradient overlap). Good news is that these plots are generated before the error occurs on the scanner, which makes debugging easier:



See interactive larger version (hover over fcompread to see how it overlaps the refocusing RF pulse)