Simulation of R- and C-Sequences with SIMPSON-1.0.1

Jörn Schmedt auf der Günne

13.08.2001

1 Introduction

R- and C-Sequences are two pulse sequence classes which have been used in solid state NMR spectroscopy. You can find a number of publications on this subject on Malcolm Levitts Hompage [*] . Here You can find a short introduction on how to calculate theses sequences numerically using SIMPSON [*] . This includes a small library of routines that allows to setup phase-, timing- and amplitude lists in a convenient way.

2 Requirements

- SIMPSON. Get it here.
- R/C-Sequences package. Get it here [*] .
- A computer.

3 The R- and C-Library

R- and C-Sequences generate a pulse train which can be specified by the three symmetry numbers the basic R element (composite pulse defined as in Fig. 1) and the supercycling scheme which is used. An RN_n^ν sequence symmetry will be written as $\{N\,n\,\nu\}$ in the input files. For the composite pulse a

notation will be used, which is explained in following figure [*] .

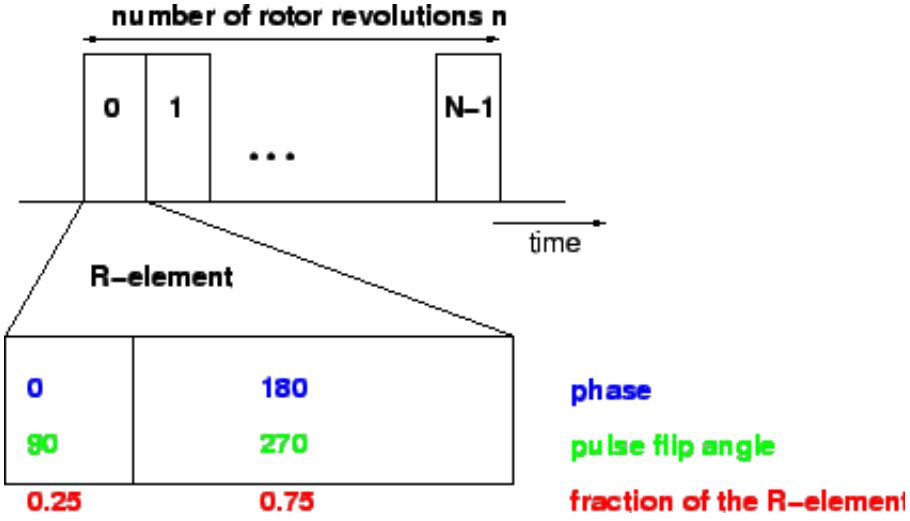

Figure: An R-Sequence with an R-element .

To give some examples for this notation:

- A C-element consisting out of 2 pi pulses with a 180 degree phase change, would be written as: $C = 360_0, 360_{180}$ or in terms of fraction-phase-flipangle

  $C = \{fraction\ phase\ flipangle\} = \{\{0.5\,0.5\}\ \{0\,180\}\ \{360\,360\}\}$

- A R-element consisting of a simple pi-pulse, would be written as: $R = 180_0$ or in terms of fraction-phase-flipangle

  $R = \{\{1.0\}\ \{0\}\ \{180\}\}$

Note that this notation also allows for windows in a pulsesequence. One simply has to set the flipangle to 0.

The simulation of R- and C-sequences is straightforward. The only error-prone step is the calculation of the numerical values of phases and amplitudes and timings. In fact for a general R- or C-sequence with a complicated composite pulse this step becomes very tedious. The idea is therefore to provide functions which calculate phase- amplitude- and timinglists which can be used like phase lists in a spectrometer pulse program. These functions take R- and C-symmetry, the composite and the supercycling as input.

In this chapter follows a simple inputfile for the calculation of a double-quantum excitation curve as a startup example, that doesn't make use of any of the special functions, then the same example with these functions and after that a description of all functions and the implemented options.

3.1 Simple Example

This is an input file which performs the calculation of a C7-double quantum curve. The C7-sequence used is the one from the original C7-paper.

$C7_2^1$ C-element: $C = 360_0, 360_{180}$

----snip----example1.in----------

```
#C7_2_1 with a 360_0 360_180 C-element
spinsys {
  channels 31P
  nuclei   31P 31P
  dipole   1 2 -2000 0 0 0
}

par {
  proton_frequency 400e6
  method          direct
  spin_rate       20000
  gamma_angles    1
  np              32
  crystal_file    bcr100
  start_operator  Inz
  detect_operator -Inz
```

```
  verbose         11111111111111
}

proc pulseq {} {
  global par spinsys
  matrix set 1 totalcoherence {2 -2}
  set length_c_ele [expr 2000000.0/$par(spin_rate)/7]
  set amplitude_c [expr $par(spin_rate)*7]
  set phase_incr_c [expr 360.0/7]
  maxdt [expr $length_c_ele/20.0]

# -- calculate propagator for C7 C=360_0 360_180 --
  reset
  set phase_c 0
  for {set i 0} {$i < 7} {incr i} {
    pulse [expr $length_c_ele/2.0] $amplitude_c [expr $phase_c]
    pulse [expr $length_c_ele/2.0] $amplitude_c [expr $phase_c+180.0]
    set phase_c [expr $phase_incr_c+$phase_c]
  }
  store 1

#-- calculate evolution and sample points--
  reset
  store 2
  for {set i 0} {$i < $par(np)} {incr i} {
    reset
    prop 2
    prop 1
    store 2
    filter 1
    prop 2
    acq
  }
  }

proc main {} {
  global par spinsys

# -- set sweep width --
set par(sw) [expr double($par(spin_rate))/2]

# -- start powder loop --
  set f [fsimpson]

# -- process and save data --
  fsave $f $par(name).fid
  funload $f
}
```

--------snip---------

3.2 Simple Example With R- and C-Library

Same as in [*] . This time making use of the R- and C-Library. This requires that the file RCpackage.tcl is in the same directory as example2.in. Changes are marked in red, explanations for the red code are marked blue.

$C7_2^1$ C-element: $C = 360_0, 360_{180}$

----snip----example2.in----------

```
# C7_2_1 with a 360_0 360_180 C-element

# load procedures from RC-library
source ./RCpackage.tcl

spinsys {
  channels 31P
  nuclei   31P 31P
  dipole   1 2 -2000 0 0 0
}

par {
  proton_frequency 400e6
  method         direct
  spin_rate      20000
  gamma_angles    1
  np             32
  crystal_file    bcr100
  start_operator  Inz
  detect_operator -Inz
  verbose        11111111111111
# Define C Symmetry and C element
  variable Csym   {7 2 1}
  variable composite {{0.5 0.5} \
                {0.0 180.0} \
                {360.0 360.0}}
}

proc pulseq {} {
# Make global variables readable
  global par spinsys element_phase_c element_amplitude_c element_length_c shortest_pulse
# Calculate maxdt from shortest pulse in the sequence
  maxdt [expr $shortest_pulse/10.0]
  matrix set 1 totalcoherence {2 -2}
  # -- calculate propagator for C-cycle --
  reset
# Longest list is usually the phase list. Use phase-, amplitude- and timinglists to calculate the pulse values.
```

```
  for {set i 0} {$i < [llength $element_phase_c]} {incr i} {
    pulse [lindex $element_length_c [expr $i%[llength $element_length_c]]] \
    [lindex $element_amplitude_c [expr $i%[llength $element_amplitude_c]]] \
    [lindex $element_phase_c $i]
  }
  store 1

  # -- calculate evolution and sample points --
  reset
  store 2
  for {set i 0} {$i < $par(np)} {incr i} {
    reset
    prop 2
    prop 1
    store 2
    filter 1
    prop 2
    acq
  }
  }

proc main {} {
# Make variables global, so they can be read in ‘‘proc pulseq’’
  global par spinsys element_phase_c element_amplitude_c element_length_c shortest_pulse

# -- set sweep width --
  set par(sw) [expr double($par(spin_rate))/[lindex $par(Csym) 1]]

# -- Create timing list --
  set element_length_c [generatePulselengthCList $par(Csym) $par(composite) $par(spin_rate)]

# -- Create phase list --
  set element_phase_c [C_phase $par(Csym) $par(composite)]

# -- determine shortest element -> maxdt --
  set shortest_pulse [smallestNumberInList $element_length_c]

# -- Create amplitude list --
  set element_amplitude_c [generateAmplitudeCList $par(Csym) $par(composite) $par(spin_rate)]

# -- start powder loop --
  set f [fsimpson]

# -- process and save data --
  fsave $f $par(name).fid
  funload $f
}

--------snip---------
```

What are the advantages compared to example1.in?

- Pulse sequence symmetry and composite pulse become parameters
- It is possible to change composite pulse, symmetry and spinning frequency without doing any other changes to the pulse program.

3.3 Procedures and Options

Non-essential options are written in brackets.

phase_list **R_phase** *Rsymmetry Relement* [*Supercyclenu SupercycleNstep Addphase*]
Generates a phase list according to the R-element, R-symmetry (notation see above [*]), the supercycling scheme and an additive phase. There are two supercycling schemes available. Both may be switched of by either not specifying the last three parameters or by setting *Supercyclenu* and *SupercycleNstep* to 1. Setting *Supercyclenu* to 2 introduces a supercycle $Cycle Cycle'$ such that all phases in $Cycle'$ are the negative values of $Cycle$. Setting *SupercycleNstep* to an integer value introduces a supercycle $Cycle_0 Cycle_1 \ldots Cycle_{Supercycle Nstep-1}$ such that all phases of C_n may be calculated by adding $360°/SupercycleNstep$ to the phases of C_{n-1} .

timing_list **generatePulselengthRList** *Rsymmetry Relement spin_rate*
Generates a timing list according to the R-element, R-symmetry (notation see above [*]) and spin rate

rf_amplitude_list **generateAmplitudeRList** *Rsymmetry Relement spin_rate*
Generates a timing list according to the R-element, R-symmetry (notation see above [*]) and spin rate

phase_list **C_phase** *Csymmetry Celement* [*Supercyclenu SupercycleNstep Addphase*]
Generates a phase list according to the C-element, C-symmetry (notation see above [*]), and the supercycling scheme. There are two supercycling schemes available. Both may be switched of by either not specifying the last three parameters or by setting *Supercyclenu* and *SupercycleNstep* to 1. Setting *Supercyclenu* to 2 introduces a supercycle $Cycle Cycle'$ such that all phases in $Cycle'$ are the negative values of $Cycle$. Setting *SupercycleNstep* to an integer value introduces a supercycle $Cycle_0 Cycle_1 \ldots Cycle_{Supercycle Nstep-1}$ such that all phases of C_n may be calculated by adding $360°/SupercycleNstep$ to the phases of C_{n-1} .

timing_list **generatePulselengthCList** *Csymmetry Celement spin_rate*
Generates a timing list according to the C-element, C-symmetry (notation see above [*]) and spin rate

rf_amplitude_list **generateAmplitudeCList** *Csymmetry Celement spin_rate*
Generates a timing list according to the C-element, C-symmetry (notation see above [*]) and spin rate

number **smallestNumberInList** *list*
Sorts *list* and returns the smallest *number*

4 Download Section

Use it at your own risk. The functions described work and are slowly going to be improved to give a better error feedback.

- Download simpson_RC_lib-0.1.tar.gz

5 Feedback and References

Before sending any emails please check that the simple examples, that come with SIMPSON, are working fine. In case you find mistakes please let me know. email:gunnej@tom.fos.su.se

[1] For information on R- and C-Sequences check Malcolm H. Levitts page. He has written a review on these sequences for the ‘‘Encyclopedia in NMR’’.

[2] SIMPSON by Niels C. Nielsen and co-workers you can find here.

- -
2001-08-16