Pulse Programming

With Python

Contents

1. Combining Python Programs and Pulse Programs

TopSpin provides a way to combine pulse programs and Python programs in a single Python program file. This feature allows pulse program designers to perform calculations using Python statements to generate delays, pulses, phases, amplitudes, frequencies and powers for one or more pulse programs contained (or referenced) in the same Python file.

When executing such a Python program, there are two possible results:

– A new pulse program which has the acquisition parameters set as computed by the Python statements according to the intentions of the program designer. The resulting pulse program, whose name can be defined in the Python program, can be executed repeatedly with the normal ***zg*** acquisition command. In this case the Python program is in fact a pulse program generator,

– NMR raw or processed data. In this case it first acts a a pulse program generator as above. In addition, it contains control statements to start acquisition or processing immediately. The Python program thus represents a complete experiment (or even several): Starting the Python program means executing the experiment(s).

Computing pulse program parameters with Python means that computation takes place before the pulse program is executed in the spectrometer. Python does not provide a way of performing “real time“ calculations while the experiment is in progress. For this purpose, the corresponding pulse program statements must be employed (e.g. `"p13=p13 + (p1*3.5 + d2/5.7)*td"`, embedded in a pulse program loop).

2. Python Functions To Compute Pulse Program Parameters

In order to manipulate pulse programs in Python, TopSpin provides a number of Python functions, outlined in the following table. We do not explain the pulse program syntax here, please refer to the Pulse Programming Manual for this purpose. You can open it from TopSpin's *Help-->Manuals* menu. In the next chapter you will find a number of complete Python program examples containing real pulse programs.

All pulse programs contained in a Python program must have the following line included before of the actual pulse program statements (typically before 'ze'):

`;$EXTERN.`

This line is treated as a comment by the pulse program compiler. It is recognized by the functions described below as a tag where to insert parameter definition statements. It indicates that the pulse program may need external parameters to be calucated by Python statements.

Python Function	***Description***

DEF_PULSPROG(ppText)
 	
Example:
`ppText = """`
`1 ze`
`2 d1`
`  p1`
`  go=2`
`exit`
`"""`
`ppObject = DEF_PULSPROG(ppText)`
	Returns a pulse program object from a text representing a pulse program in Bruker's pulse program language. This object provides the functionalities described below: setting pulse lengths, delays, etc.

GET_PULSPROG_TEXT(ppName)	Returns the pulse program text for the specified pulse

Python Function	***Description***

Example 1:
`ppText = GET_PULSPROG_TEXT('zgpr')`
`MSG(ppText) # print the text`
 	
Example 2:
`ppText = GET_PULSPROG_TEXT('zgpr')`
`ppObject = DEF_PULSPROG(ppText)`
	program name. This function can be used in conjunction with DEF_PULSPROG(ppText) so as to construct a pulse program object if one does not want to keep the pulse program text entirely in the pulse program.

SAVE_SHAPE(name, type, amplitude, phase) *Example:* `SAVE_SHAPE('sin.512', 'Excitation', am, ph)` *(assuming the lists am and ph have been filled with amplitude and phase values)*	Given a list of amplitudes and phases (both as an a array of floating numbers), this function stores the values in a TopSpin shape file which is ready to be used for data acquisition, or for display in TopSpin's Shapetool. „name“ is the shape file name, and „type“ defines the shape type (excitation, inversion)

SAVE_GRADIENT(name, amplitude)
 	
Example:
`SAVE_Gradient('grad.256', am)`
 	
(assuming the list am has been filled with amplitudes)
	Given a list of amplitudes (as an a array of floating numbers), this function stores the values in a TopSpin gradient file which is ready to be used for data acquisition, or for display in TopSpin's Shapetool.
	„name“ is the shape file name, and „type“ defines the shape type (excitation, inversion)

The following functions must be called for a pulse program object pp as defined with pp = `DEF_PULSPROG(ppText)` in the form pp.<function name>(<arguments>)

Python Function	***Description***

DEF_PULSE(String name, double value)
 	
Example:
`value = 1.5`
`pp.DEF_PULSE('myp90', value)`
 	
This will insert:
define pulse myp90
`"myp90=1.5"`
	Inserts a new pulse into the pulse program. The pulse name will be <name>, its value will be <value> microsec.
	The pulse will be inserted before the ;$EXTERN tag, which must be present in the pulse program.
	
	As a name you may use any character String allowed by the pulse program compiler. Particularly you may use standard pulses (“p1“ etc.), which will override the values set by 'eda'.

DEF_DELAY(String name, double value)
 	
Example:
`pp.DEF_DELAY('d3', 0.1)`
 	
This will insert:
`"d3=0.1"`
	Inserts a new delay into the pulse program. The delay name will be <name>, its value will be <value> sec.
	The delay will be inserted before the ;$EXTERN tag.
	
	As a name you may use any character String allowed by the pulse program compiler. Particularly you may use standard delays (“d1“ etc.), which will override the values set by 'eda'.

DEF_LOOP_COUNTER(String name, int value) *Example:* `pp.DEF_LOOP_COUNTER('counter1', 5)` *This will insert:*	Inserts a new loop counter into the pulse program. The loop counter name will be <name>, its value will be <value>. The loop counter will be inserted before the ;$EXTERN tag.

Python Function	***Description***

`define loopcounter counter1`
`"counter1=5"`	

DEF_PULSE_LIST(String name, double[] values)
 	
Example:
`values = [0.5, 1.0, 1.5]`
`pp.DEF_PULSE_LIST('plist1', values)`
 	
`This will insert:`
`define list<pulse> plist1 =`
`  {0.5 1.0 1.5}`
	Inserts a pulse list into the pulse program. The pulse list name will be <name>, its values will be <values> in microsec.
	The pulse list will be inserted before the ;$EXTERN tag.

DEF_DELAY_LIST(String name, double[] values)
 	
Example:
`values = [0.5, 1.0, 1.5]`
`pp.DEF_DELAY_LIST('dlist1', values)`
 	
`This will insert:`
`define list<delay> dlist1 =`
`  {0.5 1.0 1.5}`
	Inserts a delay list into the pulse program. The delay list name will be <name>, its values will be <values> in sec.
	The delay list will be inserted before the ;$EXTERN tag.

DEF_POWER_LIST(String name, double[] values)
 	
Example:
`values = [-6.0, -3.0, 0]`
`pp.DEF_POWER_LIST('powlist1', values)`
 	
`This will insert:`
`define list<power> powlist1 =`
`  {-6.0 -3.0 0.0}`
	Inserts a power list into the pulse program. The pulse list name will be <name>, its values will be <values> in dB.
	The power list will be inserted before the ;$EXTERN tag.

DEF_AMPLITUDE_LIST(String name, double[] values)
 	
Example:
`values = [60, 70, 80]`
`pp.DEF_AMPLITUDE_LIST('amlist1', values)`
 	
`This will insert:`
`define list<amplitude> amlist1 =`
`  {60.0 70.0 80.0}`
	Inserts an amplitude list into the pulse program. The amplitude list name will be <name>, its values will be <values> in per cent of the pulse power.
	The amplitude list will be inserted before the ;$EXTERN tag.

DEF_FREQ_LIST(String name, double[] values, String unit, String bias)
 	
Example:
`values = [100, 150, 200]`
`pp.DEF_FREQ_LIST('frlist1', values, 'hz', 'sfo')`
 	
`This will insert:`
	Inserts a frequency list into the pulse program. The frequency list name will be <name>, its values will be <values> in units defined by <unit> (must be "hz" or "ppm"), and by <bias> (must be "sfo" or "bf"). The frequency list will be inserted before the ;$EXTERN tag.

Python Function	*Description*
`define list<frequency> frlist1 = {hz sfo, 100, 150, 200}`	
DEF_PHASE_LIST(int num, double[] values, double incr, int `fractionDigits`) *Example:* `pp.DEF_PHASE_LIST(10, ph10, 45, 2)`	Appends a phase list as a phase program to the pulse program. The phase program name will be 'ph<num>', its values will be <values> in degrees. <incr> is the phase increment in degrees when the pulse program encounters an 'ip' statement. The phases are appended with the specified number of digits after the decimal point. -1 means full accuracy.

SAVE_AS(String name)
 	
Example:
`pp.SAVE_AS('my_new_pp')`
	Saves the pulse program under the specified name. From now on it is available ***edpul*** , ***gs, zg***, etc.

GET_TEXT()
 	
Example:
`MSG(pp.GET_TEXT())`
 	
`Prints the pulse program text in a window.`
	Returns the current text of the pulse program.

3. Examples

3.1 How to execute the examples

In order to execute the examples of the next sections, mark the desired complete Python program text with the mouse (assuming you have opened this manual e.g. with Acrobat Reader) and copy it to the clipboard with CTRL C. In TopSpin, type *edpy example1* to open the Python editor, and paste the clipboard into the text editor (CTRL V in Windows). Click on the *Execute* button of the editor to execute the program, or type *example1* into the TopSpin command line.

3.2 Computing phase lists

The Python program of this section consists of 2 parts: Part 1 is the actual pulse program in Bruker syntax, part 2 consists of a number of Python staments. We will now explain the Python program in detail.

1. The pulse program text is defined by `PPTEXT = """<text>"""`, where `"""` (3 quotes) is the Python way of declaring a multi-line text.
1. The statement `pp = DEF_PULSPROG(PPTEXT)` creates a pulse program object from the pulse program text. This enables us to apply the functions defined in the table of the previous chapter to the pulse program using `pp` as a reference.
2. The statements
   ```
   phlist = [0,0,4,4]*2 + [2,2,6,6]*2 + [4,4,0,0]*2 + [6,6,2,2]*2
   phinc = 360/8
   for i in range(32):
       phlist[i] = float(phlist[i])*phinc
   ```
   compute a phase list with 32 phase values using Python syntax. In Python , `[0,0,4,4]*2` is

equivalent to `[0,0,4,4,0,0,4,4]`, and a '+' sign concatenates two lists. The `for` loop iterates over the list to convert it to degrees.

3. The statement `pp.DEF_PHASE_LIST(1, phlist, phinc, 1)` applies the function `DEF_PHASE_LIST` defined in the table of the last chapter to the pulse program. It creates a phase program *ph1* (because the first argument is a '1'), and appends it to the pulse program. The increment value `phinc` is also passed on to the phase program and can be used inside the pulse program with the *ip* command.
4. The purpose of the statement `VIEWTEXT("", "", pp.GET_TEXT())` is to display what `pp.DEF_PHASE_LIST` did with the pulse program: It opens a window shows the modified pulse program text. When you run the Python program, you will see that the following was appended:

   ```
   ph1 = (float,45.0) 0.0 0.0 180.0 180.0 0.0 0.0 180.0 180.0
    90.0 90.0 270.0 270.0 90.0 90.0 270.0 270.0
    180.0 180.0 0.0 0.0 180.0 180.0 0.0 0.0
    270.0 270.0 90.0 90.0 270.0 270.0 90.0 90.0
   ```
   You may comment out `VIEWTEXT` by preceding it with a '#' character.
5. The statement `ppName = "inadph.ppy"` defines a name for the modified pulse program, and `pp.SAVE_AS(ppName)` stores it under this name in the pulse program data base. It is now available as any other pulse program for TopSpin commands such as *edpul* and *zg*.

   At this point the Python program could be complete: A new pulse program is available that has the computed phases included. When you execute the Python program, it is not required that an NMR data set is currently displayed in TopSpin, because we did not make use of any data set properties so far.
6. The last two statements in the Python program `#PUTPAR("PULPROG", "inadph.ppy")` and `#ZG()` are commented out and have no effect. If you remove the comment character the following will happen when executing the program:
   - It is now required that an NMR data set is on the TopSpin screen. `PUTPAR` will set the acquisition parameter `PULPROG` to the name `inadph.ppy`.
   - Acquisition will be started with `ZG()` using this pulse program.

   The Python program could even be further extended by appending processing functions such as `EFP()`.

Python pulse program. Computes the phase list ph1. Optionally starts acquisition.

```
PPTEXT = """
;inadph
;2D INADEQUATE phase sensitive
#include <Avance.incl>

"p2=p1*2"
"d4=1s/(cnst3*4)"
"d11=30m"
"d0=in0/2-p1*4/3.1416"

1 ze
  d11 pl12:f2
  d11 cpd2:f2
2 d1
3 p1 ph1
```

Python pulse program. Computes the phase list ph1. Optionally starts acquisition.

```
  d4
  p2 ph2
  d4
  p1 ph1
  d0
  p1 ph3
  go=2 ph31
  d1 mc #0 to 2 F1PH(ip1 & ip2, id0)
  d4 do:f2
exit

ph2=(8)  4 0 0 4 4 0 0 4 6 2 2 6 6 2 2 6
         0 4 4 0 0 4 4 0 2 6 6 2 2 6 6 2
ph3=     0 1 2 3 1 0 3 2 2 3 0 1 3 2 1 0
ph31=    0 3 2 1 3 0 1 2
"""
pp = DEF_PULSPROG(PPTEXT)

phlist = [0,0,4,4]*2 + [2,2,6,6]*2 + [4,4,0,0]*2 + [6,6,2,2]*2
phinc = 360/8
for i in range(32):
            phlist[i] = float(phlist[i])*phinc
pp.DEF_PHASE_LIST(1, phlist, phinc, 2)

VIEWTEXT("", "", pp.GET_TEXT()) # for debugging

ppName = "inadph.ppy"
pp.SAVE_AS(ppName)
#PUTPAR("PULPROG", "inadph.ppy")
#ZG()
```

3.3 Computing delays and shaped pulses

The Python program of this section consists of 2 parts: Part 1 is the actual pulse program in Bruker syntax, part 2 consists of a number of Python staments. We will now explain the Python program in detail.

1. The pulse program text is defined by `PPTEXT = """<text>"""`, where `"""` (3 quotes) is the Python way of declaring a multi-line text. It has the `;$EXTERN` tag included as an indicator where th computed delay must be inserted by the Python statements.

1. The statement `pp = DEF_PULSPROG(PPTEXT)` creates a pulse program object from the pulse program text. This enables us to apply the functions defined in the table of the previous chapter to the pulse program using `pp` as a reference.

2. The statement
   ```
     jxh = INPUT_DIALOG("ineptrdsp", "Please enter XH coupling:", ["J(XH)
   (Hz) = "])
   ```
   opens a dialog with the specified title and header inviting the user to enter a J coupling value. The result is returned in the first element of the array `jhx` (in text string representation).

   The statements
   ```
   d3 = 1/(6*float(jxh[0]))
   d4 = 1/(4*float(jxh[0]))
   pp.DEF_DELAY("d3", d3)
   ```

```
  pp.DEF_DELAY("d4", d4)
```
convert the entered text to floating numbers and calculate the delays `d3` and `d4`. The respective delay definitions are inserted into the pulse program before the `;$EXTERN` tag using the `DEF_DELAY` function.

3. The statement `pp.DEF_PHASE_LIST(1, phlist, phinc, 2)` applies the function `DEF_PHASE_LIST` defined in the table of the last chapter to the pulse program. It creates a phase program *ph1* (because the first argument is a '1'), and appends it to the pulse program. The increment value `phinc` is also passed on to the phase program and can be used inside the pulse program with the *ip* command. The last argument defines the number of digits after the decimal point. This number is applied when writing the list into the pulse program. -1 mean full precision.
4. The purpose of the statement `VIEWTEXT("", "", pp.GET_TEXT())` is to display what `pp.DEF_PHASE_LIST` did with the pulse program: It opens a window shows the modified pulse program text. When you run the Python program and enter “5“ for the JXH coupling, you will see that the following was inserted before `;$EXTERN`:

```
 "d3=0.03333333333333333"
 "d4=0.05"
```
You may comment out `VIEWTEXT` by preceding it with a '#' character.

5. The next section of the Python program computes the amplitudes and phases of a shaped pulse and needs Python's mathematical library. `import math` loads it to be ready for use.
6. `amplitudes = []` defines an empty array to hold the shape's amplitude values.
```
for i in range(512):
 amplitudes.append(100*math.sin((3.1415*i)/512))
```
fills this array with 512 values, normalized to 100 (as required by the acquisition software), using the sin() library function.

```
phases = [] # in degrees
for i in range(512):
      if i<256:
            phases.append(360*math.pow(float(i)/256, 3))
      else:
            phases.append(360*math.pow(float(512-i)/256, 3))
```

defines and computes the shape's phase values in a similar way.
The next statements define a file name for the shape, and store it as an "Excitation" shape into TopSpin's standard place. You can view the computed result by opening this shape with TopSpin's S*hapeTool*.

```
shapeName = "SinExam.512"
SAVE_SHAPE(shapeName, "Excitation", amplitudes, phases)
```

7. The last four statements in the Python program
```
  #PUTPAR("SPNAM2", shapeName)
  #PUTPAR("SPOFFS 2", "5.0")
  #PUTPAR("PULPROG", "inadph.ppy")
  #ZG()
```
are commented out and have no effect. If you remove the comment character the following will happen when executing the program:
- It is now required that an NMR data set is on the TopSpin screen. `PUTPAR` will set the acquisition parameters `SPNAM2, SPOFFS 2, PULPROG` to the specified values.
- Acquisition will be started with `ZG()` using this pulse program.

The Python program could even be further extended by appending processing functions such as

```
EFP().
```

Python pulse program. Computes the delays d3/d4 and a shaped pulse. Optionally starts acquisition

```
PPTEXT = """
;ineptrdsp
;INEPT for non-selective polarization transfer
;with decoupling during acquisition
;using shaped pulses for 180degree pulses on f1 - channel

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

;$EXTERN
"p4=p3*2"
"d12=20u"
"DELTA1=d4-cnst17*p12/2"
"DELTA2=d3-cnst17*p12/2"

1 ze
2 30m do:f2
  d1
  d12 pl2:f2
  (p3 ph1):f2
  4u
  DELTA1 pl0:f1
  (center (p4 ph2):f2 (p12:sp2 ph4) )
  4u
  DELTA1 pl1:f1
  (p3 ph3):f2 (p1 ph5)
  4u
  DELTA2 pl0:f1
  (center (p4 ph2):f2 (p12:sp2 ph6) )
  4u
  DELTA2 pl12:f2
  go=2 ph31 cpd2:f2
  30m do:f2 mc #0 to 2 F0(zd)
exit

ph1=0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2
ph2=0 2
ph3=1 1 3 3
ph4=0 2
ph5=0 0 0 0 1 1 1 1 2 2 2 2 3 3 3 3
ph6=0 2 0 2 1 3 1 3
ph31=0 0 2 2 1 1 3 3
"""

import math
pp = DEF_PULSPROG(PPTEXT) # define a variable "pp" representing
the pulse program
jxh = INPUT_DIALOG("ineptrdsp", "Please enter XH coupling:",
   ["J(XH) (Hz) = "])
d3 = 1/(6*float(jxh[0]))
d4 = 1/(4*float(jxh[0]))
pp.DEF_DELAY("d3", d3)
pp.DEF_DELAY("d4", d4)
VIEWTEXT("", "", pp.GET_TEXT()) # for debugging
```

Python pulse program. Computes the delays d3/d4 and a shaped pulse. Optionally starts acquisition

```
# calc. a sine shape (0..PI)
amplitudes = [] #  normalized to 0...100
for i in range(512):
      amplitudes.append(100*math.sin((3.1415*i)/512))

phases = [] # in degrees
for i in range(512):
      if i<256:
            phases.append(360*math.pow(float(i)/256, 3))
      else:
            phases.append(360*math.pow(float(512-i)/256, 3))

shapeName = "SinExam.512"
SAVE_SHAPE(shapeName, "Excitation", amplitudes, phases)
ppName = "ineptrdsp.ppy"
pp.SAVE_AS(ppName)
#PUTPAR("SPNAM2", shapeName)
#PUTPAR("SPOFFS 2", "5.0")
#PUTPAR("PULPROG", ppName)
#ZG()
```