esrpoise

Jean-Baptiste Verstraete, Jonathan Yong, Mohammadali Foroozandeh

Aug 01, 2023

Contents

1 Table of contents **2**

1.1 Installation . . . 2

1.1.1 Installing Python 3 . . . 2

1.1.2 Installing esrpoise . . . 3

1.1.3 Updating esrpoise . . . 4

1.1.4 Installing from source . . . 4

1.1.5 Installing without Internet . . . 4

1.2 Set up an optimisation . . . 4

1.2.1 with standard parameters . . . 5

1.2.2 with .def file parameters . . . 5

1.2.3 with user-defined parameters (advanced) . . . 6

1.2.4 with user-defined cost function (advanced) . . . 7

1.2.5 Setup Tips (advanced) . . . 8

1.3 Run an optimisation . . . 8

1.3.1 Procedure . . . 8

1.3.2 Running tips . . . 9

1.4 Built-in parameters . . . 9

1.4.1 Bridge - Receiver Unit . . . 10

1.4.2 Bridge - MPFU control . . . 10

1.4.3 FT EPR Parameters . . . 10

1.5 Modules . . . 10

1.5.1 xepr_link.py . . . 11

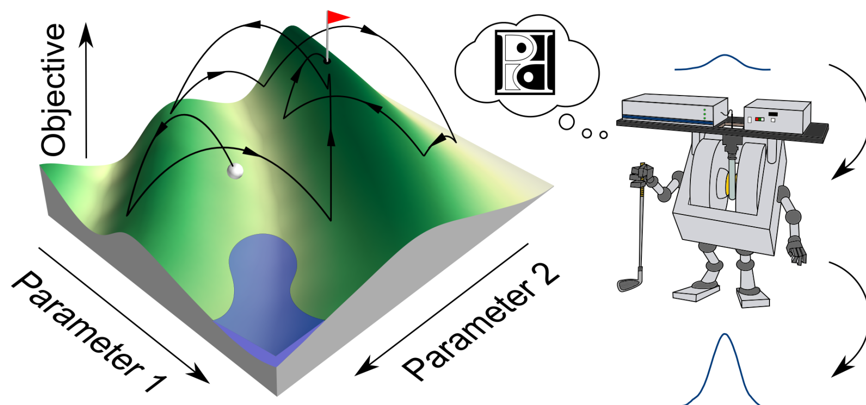
1.5.2 costfunctions.py . . . 12

1.6 Troubleshoot . . . 13

1.6.1 Compilation . . . 13

1.6.2 fast .def file modification . . . 14

1.6.3 Shape loading . . . 15



ESR-POISE (*Electron Spin Resonance Parameter Optimisation by Iterative Spectral Evaluation*) is a Python package for on-the-fly optimisation of ESR parameters in Bruker BioSpin Xepr software.

In here you will find guides on setting ESR-POISE up and using it to optimise ESR applications. Depending on your level of interaction with the software, you may not need to read all of it. For example, if somebody else has already installed and set up some esrpoise scripts for you, you can probably skip to *Run an optimisation*. You should also skip advanced subsections for your first steps with esrpoise.

For more insight, consult the source code. In particular, have a look at the examples scripts present in the package directory (`esrpoise/examples/`).

Note: The documentation you are currently reading is for version v1.0.1 of POISE. To check your current version of POISE, type `pip show esrpoise` in terminal.

Chapter 1

Table of contents

1.1 Installation

The requirements are:

- **Xepr**. We have tested Xepr versions 2.8b.5.
- **Python 3**. ESRPOISE requires a minimum version of **Python 3.6.** (We have tested up to Python 3.8, Python 3.9 installation is supported).

1.1.1 Installing Python 3

This is most easily done by downloading an installer from the official CPython website (https://www.python.org/downloads/), or via your package manager (`apt`, `yum` or similar).

On CentOS 7, you would typically need:

```
yum -y install python3
```

You might also need to tkinter with:

```
yum -y install python3-tkinter
```

Note that this creates a `python3` executable, which is independent from the previously installed Python 2 (which can still be invoked with `python`).

A particular issue we faced when setting up POISE was installing Python on CentOS 7 without administrator rights. If you are in this situation, you will have to compile the Python source code, using the following steps:

1. Download the source code for Python 3.7. We used 3.7.11, but anything should work.
2. Download the source code for Tcl/Tk (at least for now, Python needs to be compiled with tkinter for POISE to run successfully).
3. Compile Tcl/Tk, making sure to install it to a user-writable directory (e.g. your home directory).
4. Compile Python, again installing it to your home directory. This step is pretty difficult as there are certain (undocumented) compilation flags which must be passed correctly.

The following series of shell commands (give or take) worked for us, so you could simply run the following:

```
# Download source code
curl -LO https://www.python.org/ftp/python/3.7.11/Python-3.7.11.tgz
curl -LO https://prdownloads.sourceforge.net/tcl/tcl8.6.12-src.tar.gz
curl -LO https://prdownloads.sourceforge.net/tcl/tk8.6.12-src.tar.gz
tar xf Python-3.7.11.tgz
tar xf tcl8.6.12-src.tar.gz
tar xf tk8.6.12-src.tar.gz
# Install tcl/tk
cd tcl8.6.12/unix
./configure --prefix=$HOME/.local/tcltk
make
make install
cd ../../tk8.6.12/unix
./configure --prefix=$HOME/.local/tcltk
make
make install
# Install Python
cd ../../Python-3.7.11/
CPPFLAGS="-I$HOME/.local/tcltk/include" LDFLAGS="-L$HOME/.local/tcltk/lib -Wl,-rpath,$HOME/.
↪local/tcltk/lib -ltcl8.6 -ltk8.6" ./configure --prefix=$HOME/.local/python3 --with-tcltk-
↪includes="-I$HOME/.local/tcltk/include" --with-tcltk-libs="-L$HOME/.local/tcltk/lib"
make
make install
```

One of the tested spectrometer computer also required the installation of `libffi` and `libffi-devel` before being able to `make install`.

You should then have a working installation of Python 3.7 in `~/.local/python3/bin/python`. Note that you should always use this version of Python whenever installing packages: so, it's safer to always use `/path/to/python -m pip install X` rather than just `pip install`.

Naturally, you can place this Python executable first in your `$PATH` in order to avoid having to type out the full path every time, for example by placing the following line:

```
export PATH=~/.local/python3/bin:$PATH
```

inside your `~/.bashrc` or `~/.bash_profile`.

If there are any issues, please get in touch via GitHub or email.

1.1.2 Installing esrpoise

Once Python 3 is installed, you can install POISE using `pip`:

```
python -m pip install esrpoise
```

(replace `python` with `python3` if necessary)

The following Python packages are required, they should get automatically installed with esrpoise if you do not already have them:

- **numpy**
- **XeprAPI**
- **pybobyqa**

1.1.3 Updating esrpoise

Simply use:

```
python -m pip install --upgrade esrpoise
```

(again replacing `python` with `python3` if necessary). All other steps (including troubleshooting, if necessary) are the same.

1.1.4 Installing from source

If you obtained the source code (e.g. from `git clone` or a GitHub release) and want to install from there, simply `cd` into the top-level `esrpoise` directory and run:

```
python -m pip install .
```

Add `-e` to be able to edit the code. Equivalently you can run:

```
python -m setup.py install
```

Replace `install` with `develop` to edit the code.

(use `python3` if necessary)

1.1.5 Installing without Internet

If the computer you are using does not have an Internet connection, then you will need to:

1. Download the POISE source code from GitHub: `git clone https://github.com/foroozandehgroup/esrpoise` and copy it over to the target computer.
2. Install Python by downloading the installer from a different computer and copying it over.
3. On the target computer, install the POISE package locally by navigating to the `esrpoise` directory you copied over and doing `python -m pip install .` (note the full stop at the end).

1.2 Set up an optimisation

To set up an optimisation, you can to create a small python script which:

- loads an instance Xepr of the XeprAPI,
- calls the function optimise from esrpoise.

1.2.1 with standard parameters

When calling `optimise()`, you should to at least pass:

- the Xepr instance,
- your parameters names, initial values, lower bounds and upper bounds as lists,
- a cost function, which can be imported from the pre-existing ones (cf. *costfunctions.py*),
- the maximum number of function evaluations (not technically mandatory but set to `0` by default).

Only a handful of Xepr parameters can be accessed through a simple optimisation (cf. *Built-in parameters*).

Example of standard parameter optimisation script:

```
from esrpoise import xepr_link
from esrpoise import optimise
from esrpoise.costfunctions import maxabsint_echo

# load Xepr instance
xepr = xepr_link.load_xepr()

# fine adjustment of centre field for bisnitroxide sample at X-band
xbest, fbest, message = optimise(xepr,
                                 pars=['CenterField'],
                                 init=[3450],
                                 lb=[3445],
                                 ub=[3455],
                                 tol=[0.1],
                                 cost_function=maxabsint_echo,
                                 maxfev=20)
```

Note that `xepr`, `pars`, `init`, `lb`, `ub`, `tol` need to be input in this order, all other parameters are keywords arguments whose order does not matter.

> **Warning:** Make sure to chose an appropriate tolerance for your instrumentation. For example, delays and pulse duration need to be multiple of 2ns on our instrument. We therefore must choose a tolerance which is a multiple of 2ns when optimising those parameters.

1.2.2 with .def file parameters

Optimise parameters from your .def file by indicating their names. In addition to the standard parameter requirements, you should precise:

- the path of your .exp file,
- the path of your .def file.

Example (also note the use of several parameters in lists and the optimiser explicit choice):

```
# .exp and .def files location
location = '/home/xuser/xeprFiles/Data/'
exp_f = location + '2pflip.exp'
def_f = location + '2pflip.def'
```

(continues on next page)

(continued from previous page)

```
# optimisation of pulse length and amplitude
xbest, fbest, message = optimise(xepr,
                                 pars=["p0", "Attenuation"],
                                 init=[8, 5],
                                 lb=[2, 0],
                                 ub=[36, 10],
                                 tol=[2, 0.5],
                                 cost_function=maxabsint_echo,
                                 exp_file=exp_f,
                                 def_file=def_f,
                                 optimiser="bobyqa",
                                 maxfev=20)
```

1.2.3 with user-defined parameters (advanced)

To optimise your own parameters, you should define a callback function, used to set up your parameters. The simplest is to define it in your optimisation script (after the imports and before your script code):

```
def my_callback(my_callback_pars_dict, *mycallback_args):
    """
    Parameters
    ----------
    pars_dict: dictionary
        dictionary with the name of the parameters to optimise as key and
        their value as value
    *mycallback_args
        other possible arguments for callback function
    """
    # user operations and parameters modifications
```

Indicate that your parameter is user-defined by including '&' at the start of its name:

```
optimise(Xepr, pars=[... , '&_', ...], ...,
               callback=my_callback, callback_args=my_callback_args)
```

You can pass arguments (as a tuple) to `callback` by using `callback_pars`.

In the following example, we modify a shape parameter with the module mrpypulse. (bandwidth `bw` of an HS1 pulse)

```
import os
from esrpoise import xepr_link
from esrpoise import optimise
from esrpoise.costfunctions import maxabsint_echo
from mrpypulse import pulse


def shape_bw(callback_pars_dict, shp_nb):

    # getting  bw value from the callback parameters to be optimised
    bw = callback_pars_dict["&bw"]
```

(continues on next page)

(continued from previous page)

```
    # create hyperbolic sechant shape bw value
    p = pulse.Parametrized(bw=bw, tp=80e-9, Q=5, tres=0.625e-9,
                           delta_f=-65e6, AM="tanh", FM="sech")

    p.xepr_file(shp_nb)    # create shape file

    # shape path
    path = os.path.join(os.getcwd(), str(shp_nb) + '.shp')

    xepr_link.load_shp(xepr, path)  # send shape to Xepr

    return None


xepr = xepr_link.load_xepr()

#  HS pulse bandwidth optimisation
xbest, fbest, message = optimise(xepr,
                                 pars=['&bw'],
                                 init=[80e6],
                                 lb=[30e6],
                                 ub=[120e6],
                                 tol=[1e6],
                                 cost_function=maxabsint_echo,
                                 maxfev=120,
                                 nfactor=5,
                                 callback=shape_bw,
                                 callback_args=(7770,))

# NB: '(7770,)' is equivalent to 'tuple([7770])'
```

Note that we first have to import the modules and then define the callback function before finally writing the actual optimisation script.

1.2.4 with user-defined cost function (advanced)

You can define your own cost function and pass it to the function optimise. Your cost function should treat the data from one of your experiment run to return a single number which will be minimised by the optimiser.

We can for example conduct an optmisation on the spectrum with a zero-filling operation (data is here a simple time-domain FID):

```
import numpy
from esrpoise import xepr_link
from esrpoise import optimise

def maxabsint(data):
    """
    Maximises the absolute (magnitude-mode) intensity of the spectrum.
    """
    zero_filling = 4*length(data.O.real)
```

(continues on next page)

(continued from previous page)

```
    spectrum = np.fft.fft(data.O.real + 1j * data.O.imag, n=zero_filling)

    return -np.sum(np.abs(spectrum(data)))


# load Xepr instance
xepr = xepr_link.load_xepr()

# fine adjustment of center field for bisnitroxide sample at X-band
xbest, fbest, message = optimise(xepr,
                                 pars=['CenterField'],
                                 init=[3450],
                                 lb=[3445],
                                 ub=[3455],
                                 tol=[0.1],
                                 cost_function=maxabsint,
                                 maxfev=20)
```

1.2.5 Setup Tips (advanced)

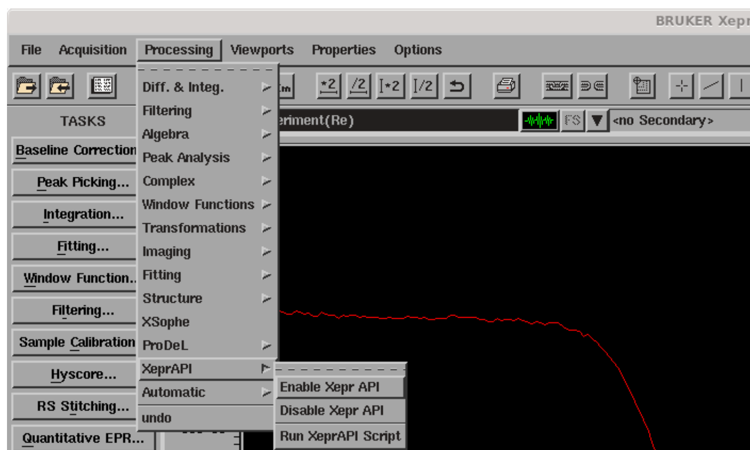
- Put several optimisations in one script.
- Automate your actions by using XeprAPI commands, the functions from *xepr_link.py* and `param_set` from main.py.
- Reuse the best parameter value(s) from the optimiser `xbest`.
- Use `callbak` to add user-specific operation at each iteration. You do not need to indicate user-defined parameters, `callback_pars_dict` is sent back empty if no user-defined parameters are found.
- Use `acquire_esr.calls` in your callback function to access the current number of iteration(s).
- Use the parameter `nfactor` of `optimise()` to expand the distance between the first steps of the optimisers, in particular if you have a low tolerance.
- Accelerate you optimisation routine if your .shp, .def and .exp file compile fast enough with `xepr_link.COMPILATION_TIME` (cf. *Compilation*).
- When using a single script with functions, be aware of your variables scope.

1.3 Run an optimisation

1.3.1 Procedure

In Xepr:

- Set up your experiment as you would normally do.
- **Enable XeprAPI (Processing → XeprAPI → enable XeprAPI).**



- Run your experiment once by clicking on play to load it into XeprAPI (can also be done before enabling XeprAPI).

Run your optimisation script in a terminal with

```
python yourscript.py
```

(replace `python` with `python3` if necessary)

At each iteration, esrpoise:

- adjusts the parameters to be optimised,
- runs the experiment,
- computes the cost function.

Once `optimise()` is done, you get a set of optimised parameters. These are set up in Xepr but your experiment is not run. You need to run it if you want to have the optimised result loaded in Xepr.

1.3.2 Running tips

- You can interrupt the optimisation with `ctrl+C`. It is recommend to do so while a measurement is running.
- When using .exp and .def file, save them beforehand to avoid getting a warning. Otherwise this warning pauses the optimisation and you need to manually get rid of it at each iteration.
- When running your experiment once to load it in XeprAPI, you can interupt it by clicking on play again. This is enough to set it up, sparing the time of a full run.

1.4 Built-in parameters

Built-in parameters can have different constraints depending on your spectrometer. Values documented here are just indications. All parameters currently supported are indicated below (in addition to any variable placed in the .def file as seen in *Set up an optimisation*).

1.4.1 Bridge - Receiver Unit

- `"VideoGain"` (dB), video gain, indicative minimum tolerances: 3dB, 6dB
- `"Attenuation"` (dB), high power attenuation, indicative minimum tolerance: 0.01dB
- `"SignalPhase"` (~0.129deg), signal phase, indicative minimum tolerance: 1
- `"TMLevel"` (%), transmitter level, 0 to 100

1.4.2 Bridge - MPFU control

- `"BrXPhase"`, +<x> Phase
- `"BrXAmp"`, +<x> Amplitude
- `"BrYPhase"`, +<y> Phase
- `"BrYAmp"`, +<y> Amplitude
- `"BrMinXPhase"`, -<x> Phase
- `"BrMinXAmp"`, -<x> Amplitude
- `"BrMinYPhase"`, -<y> Phase
- `"BrMinYAmp"`, -<y> Amplitude

These are rounded in Xepr to closest 0.049% (approximately, not linear).

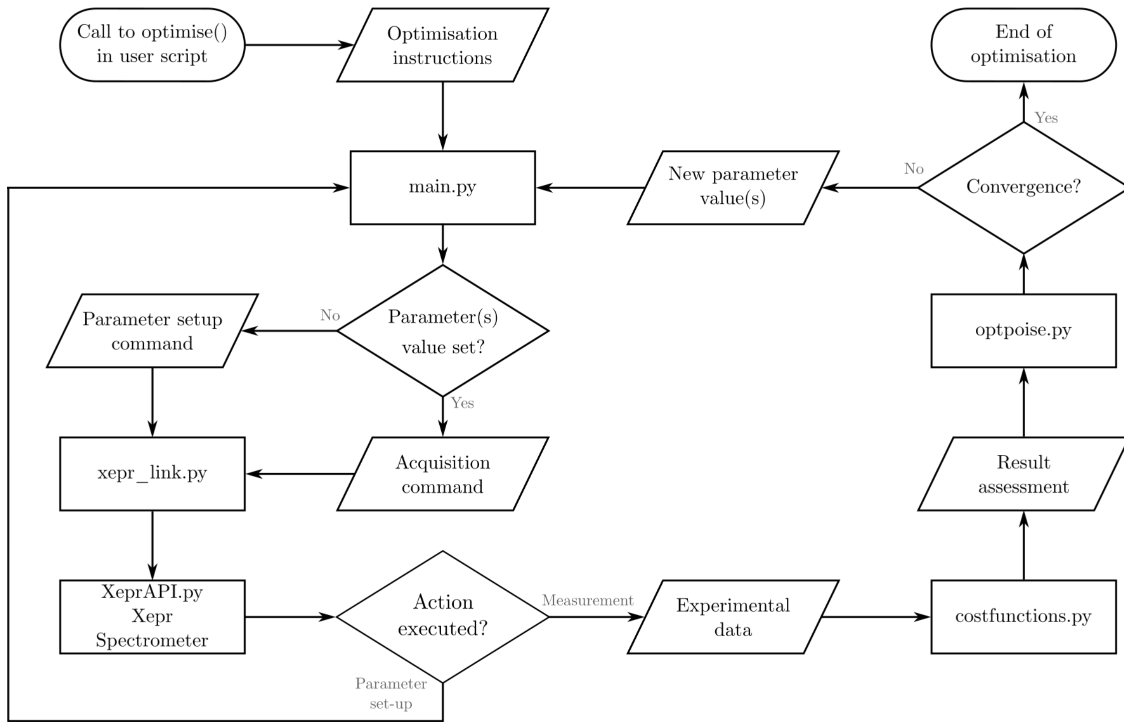
1.4.3 FT EPR Parameters

- `"CenterField"`, field position (G), indicative minimum tolerance: 0.05G

1.5 Modules

The esrpoise code is organized into 4 modules:

- `main.py` for the optimisation to take place (set parameters, run the experiment, report results...),
- `xepr_link.py` to handle communication with Xepr,
- `costfunctions.py` which contains standard cost functions,
- `optpoise.py` which contains the necessary for the optimisers (cf. source code for more details).



1.5.1 xepr_link.py

1.5.2 costfunctions.py

Spectrum

Echo

Other

1.6 Troubleshoot

1.6.1 Compilation

Bugs can be created if `xepr_link` does not allow for enough time to let Xepr compile.

Increase the compilation time of the .exp, .def and .shp files (1s by default) with the global variable `COMPILATION_TIME` from *xepr_link.py*:

```
xepr_link.COMPILATION_TIME = 2  # (s)

xepr = xepr_link.load_xepr()

# .exp and .def files location
location = '/home/xuser/xeprFiles/Data/'
exp_f = location + '2pflip.exp'
def_f = location + '2pflip.def'

# optimisation of pulse length and amplitude
xbest, fbest, message = optimise(xepr,
                                  pars=["p0", "Attenuation"],
```

(continues on next page)

(continued from previous page)

```
                                        init=[8, 5],
                                        lb=[2, 0],
                                        ub=[36, 10],
                                        tol=[2, 0.5],
                                        cost_function=maxabsint_echo,
                                        exp_file=exp_f,
                                        def_file=def_f,
                                        optimiser="bobyqa",
                                        maxfev=20)

# run experiment with optimal parameters
xepr_link.run2getdata_exp(xepr, "Signal", exp_f)
```

If the files are compiling fast enough, decrease `COMPILATION_TIME` to accelerate the optimisation routine.

1.6.2 fast .def file modification

ESR-POISE does not use the function `modif_def_PlsSPELGlbTxt()` from *xepr_link.py* despite its ability to modify the .def file variables without referring to the .def file location.

While this function would save a couple of seconds per iteration, it can cause a freeze of the .def file modification, forcing the user to manually interrupt its script:

```
Traceback (most recent call last):
  File "crash_def.py", line 6, in <module>
    xepr_link.modif_def(xepr, ['p0'], ['8'])
  File "/home/xuser/xeprFiles/Data/Organic/JB/220511/esrpoise-dev/esrpoise/xepr_link.py",␣
↪line 190, in modif_def
    currentExp["ftEPR.PlsSPELSetVar"].value = cmdStr
  File "/home/xuser/.local/lib/python3.6/site-packages/XeprAPI/main.py", line 1195, in __
↪getitem__
    return Parameter(self, name)
  File "/home/xuser/.local/lib/python3.6/site-packages/XeprAPI/main.py", line 1243, in __
↪init__
    self._name = self._parent.findParam(name, findall=True)
  File "/home/xuser/.local/lib/python3.6/site-packages/XeprAPI/main.py", line 1139, in␣
↪findParam
    for par in self.getFuParList(fu):
  File "/home/xuser/.local/lib/python3.6/site-packages/XeprAPI/main.py", line 1096, in␣
↪getFuParList
    self.aqGetExpFuParList(fu, buf, 10000)
  File "<string>", line 1, in <lambda>
  File "<string>", line 1, in <lambda>
  File "/home/xuser/.local/lib/python3.6/site-packages/XeprAPI/main.py", line 510, in _
↪callXeprfunc
    if self._API.XeprCallFunction(funcidx) != 0:
```

This bug was observed and reproduced after a few hundred to a few thousand calls to `modif_def_PlsSPELGlbTxt()`.

1.6.3 Shape loading

If a bug with shape loading is encountered after a certain number of iterations (typically 114 on older versions of Xepr), it should be solved by reseting Xepr to avoid AWG overloading. Use the following lines in your callback function (requires to import `acquire_esr` from `esrpoise`):

```
if acquire_esr.calls % 114 == 0 and acquire_esr.calls != 0:
    print('reset required')
    xepr_link.reset_exp(Xepr)
```