opti4Abq, a python toolbox to run abaqus in an optimisation loop

A tentative tutorial/documentation

Marlène Mengoni

m.mengoni@leeds.ac.uk

May 31, 2017

Concept: FE model calibration

One use of FE models is to calibrate some unknown variable(s) - e.g. material parameters - to match known experimental data.
To do so, one need:

- 1 or more FE models that are parametrised with the unknown variable(s) to calibrate;
- a way to run FE models with parameters automatically;
- a way to process FE models so that it outputs the value(s) of interest;
- the corresponding experimental data;
- a process to vary the parameters for the FE to match the experimental data.

Concept: FE model calibration

One use of FE models is to calibrate some unknown variable(s) - e.g. material parameters - to match known experimental data.
To do so, one need:

- 1 or more FE models that are parametrised with the unknown variable(s) to calibrate;
- a way to run FE models with parameters automatically;
- a way to process FE models so that it outputs the value(s) of interest;
- the corresponding experimental data;
- a process to vary the parameters for the FE to match the experimental data.

Abaqus scripting interface; opti4Abq toolbox framework;
e.g. postPro4Abq toolbox

opti4Abq

Data Preparation

- 1 or more FE models = 1 or more Python files defining an Abaqus job and post-processing function
- the corresponding experimental data = 1 or more text files containing the experimental data formatted as the post-processing of the FE models formats its output

NOTES:

all Python files must be stored in 1 folder [`pyPath`]

all experimental files must be stored in 1 folder [`expPath`]

(can be same folder!)

`pyPath` must contain an empty file called `__init__.py`

Data Preparation - python file

The file name of the python file is the model identifier. The file must contain:

1. an abaqus job creation (function of parameters):

```
if __name__ == '__main__':
    import sys
    nbParam = 2
    paramToOpti = list()
    for arg in range(nbParam):
       paramToOpti.insert(0,float(sys.argv[-1-arg]))
    job = jobCreation(paramToOpti)
    job.submit()
    job.waitForCompletion()
```

2. A function (called `postPro`) that reads the odb and writes a file called `output.dat` with output of interest

Data Preparation - data file

The experimental data relative to each model must be stored in a text file called `identifier.dat`

The type of data supported is:

1. a function y = y(x) in 2 column x y — e.g. displacement/load data
   ```
   0.0 0.0
   0.1 0.1
   0.2 0.5
   0.9 2.3
   ```
2. a list of 1D values — e.g. σ_{VM} at known locations
   ```
   0.0
   0.1
   0.2
   0.9
   ```
3. a scalar — e.g. structure stiffness
   ```
   10.0
   ```

opti4Abq object

- preparing the optimisation

```
import opti4AbqTools.Opti4AbqClass as optiTools
myOpti = optiTools.Opti4Abq(p0, expPath, pyPath)
```

  p0 is a Python list with the initial guess of the parameter values
- Running the optimisation

  if type of data is scalar:

```
p,fVal,info = myOpti.runScalar()
```

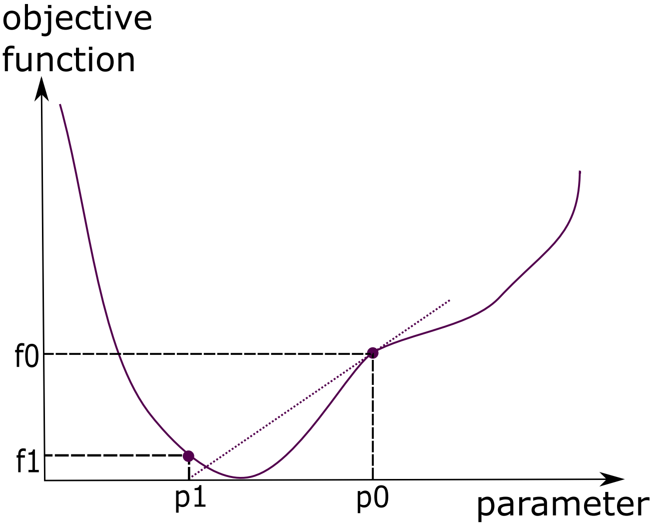
  else:

```
p,fVal,info = myOpti.run()
```

opti4Abq methods

- `myOpti.setBounds(low = boundsL,high = boundsH)`
  where `boundsL` and `boundsH` are the lowest and highest authorised values of the parameters
- `myOpti.setResidualsAsAbsolute(True)`
  will try to minimise the absolute difference between the experimental and computational data (default is set at False: relative difference is used)
- `myOpti.setVerbose(True)`
  writes plenty of things and save values at each iterations (for some opti algorithms)

Gradient-based optimisation



opti4Abq options

`myOpti.setOptions(options):`

- control end of process

`options['maxIter']= 10`	max number of iterations
`options['tol']= 1e-4`	tolerance on the parameter variation (also referred to as `xtol`)
`options['ftol']= options['tol']*1e-4`	tolerance on the function
`options['gtol']= options['tol']*1e-4`	tolerance on the gradient

- control step to evaluate gradient

`options['eps']= 1e-4`	step for the jacobian

Objective function

scalar data

difference or error between data and FE

1D data

RMS difference or error between data and FE

(x,y) data

RMS difference or error between data and FE

BUT unlikely to have same sampling rate in x

$\longrightarrow$ first resample to same (smallest) sampling rate in x

If Nb models > 1, RMS error/difference over all models of previous value.

Optimisation methods (interfaced from scipy)

scalar data & 1 parameter

Brent method	whether parameter bounded or not

other data & 1 parameter

L-BFGS-B method	bounded parameter
Conjugate gradient method	non-bounded

any data & >1 parameters

Trust Region Reflective method	bounded parameters
Levenberg-Marquadt method (MINPACK)	non-bounded

Outputs

`p,fVal,info = myOptiProcess.run()`

- `p`: output parameters
- `fVal`: value of the objective function
- `info`: a dictionary of information:
  - `info['funcalls']`: number of function evaluation that were required
  - `info['task']`: an output message by the optimisation process
  - `info['grad']`: the value of the jacobian

\+ all Abaqus files (and output of interest) of the last run (in `workspace`)

\+ for some of the algorithms, intermediate `p` and `fVal` values into text files (in `results`) [if `verbose==True`]

Limitations

- all parameters need to have values of the same order
- all python files need to be stored in same folder (hence need second copy in other directory if running on subset of models) and only python files that are models can be in that folder (in particular no tools module)
- all python files defining abaqus jobs need to be launchable with `abaqus cae nogui=myPythonFile.py`
  ? may be an issue with user subroutines
- all models need to have the same type of data
- not easily portable on HPC environments with SGE queues!
- currently only limited gradient-based opti algorithms interfaced
- . . . [probably plenty of others!]

Examples

1. `scalar1Param` directory: scalar function, 1 parameter, bounded
2. `1D2Param` directory: 1D function, 2 parameters, bounded
3. `xy2Param` directory: (x,y) function, 2 parameters, bounded

Note

All example files are set with absolute paths to search for external modules/files that need setting up and all use the postPro4Abq toolbox to post-process the data! They won't run without a bit of changes from the user!

Requirements

The toolbox is built for Python 2.x and Abaqus $>$ 6.13 (not been tested on anterior versions).
Requires scipy 0.18 or above, with numpy 1.11 or above.

I personally use the anaconda distribution of Python (conda v.4.3.11, Python v.2.7.13)

The toolbox has been tested on Windows platform only with no guarantee to work on any other OS

Acknowledging the toolbox

The toolbox is available in github with latest stable/documented release on zenodo

To reference opti4Abq in publications, please cite both of the following:

1. Mengoni M., Luxmoore B.J., Jones A.C., Wijayathunga V.N., Broom N.D. & Wilcox R.K. (2015) "Derivation of inter-lamellar behaviour of the intervertebral disc annulus." Journal of the Mechanical Behavior of Biomedical Materials, v 48, 164–172
2. Mengoni M. (2017) "opti4Abq (v 2.0), a generic python code to run Abaqus in an optimisation loop". `http://dx.doi.org/10.5281/zenodo.580475`

e.g. *The opti4Abq toolbox*[1,2] *using the L-BFGS-B algorithm implemented in SciPy (Python 2.7, www.python.org) was used in this work.*

Thanks!

This work was funded through WELMEC, a Centre of Excellence in Medical Engineering funded by the Wellcome Trust and EPSRC, under Grant number 088908/Z/09/Z and through EPSRC Grant EP/K020757/1 and ERC Grant StG-2012-306615

EPSRC
Engineering and Physical Sciences
Research Council

erc
European Research Council
Established by the European Commission

wellcome trust