Interface graphiques avec pyQT / PySide

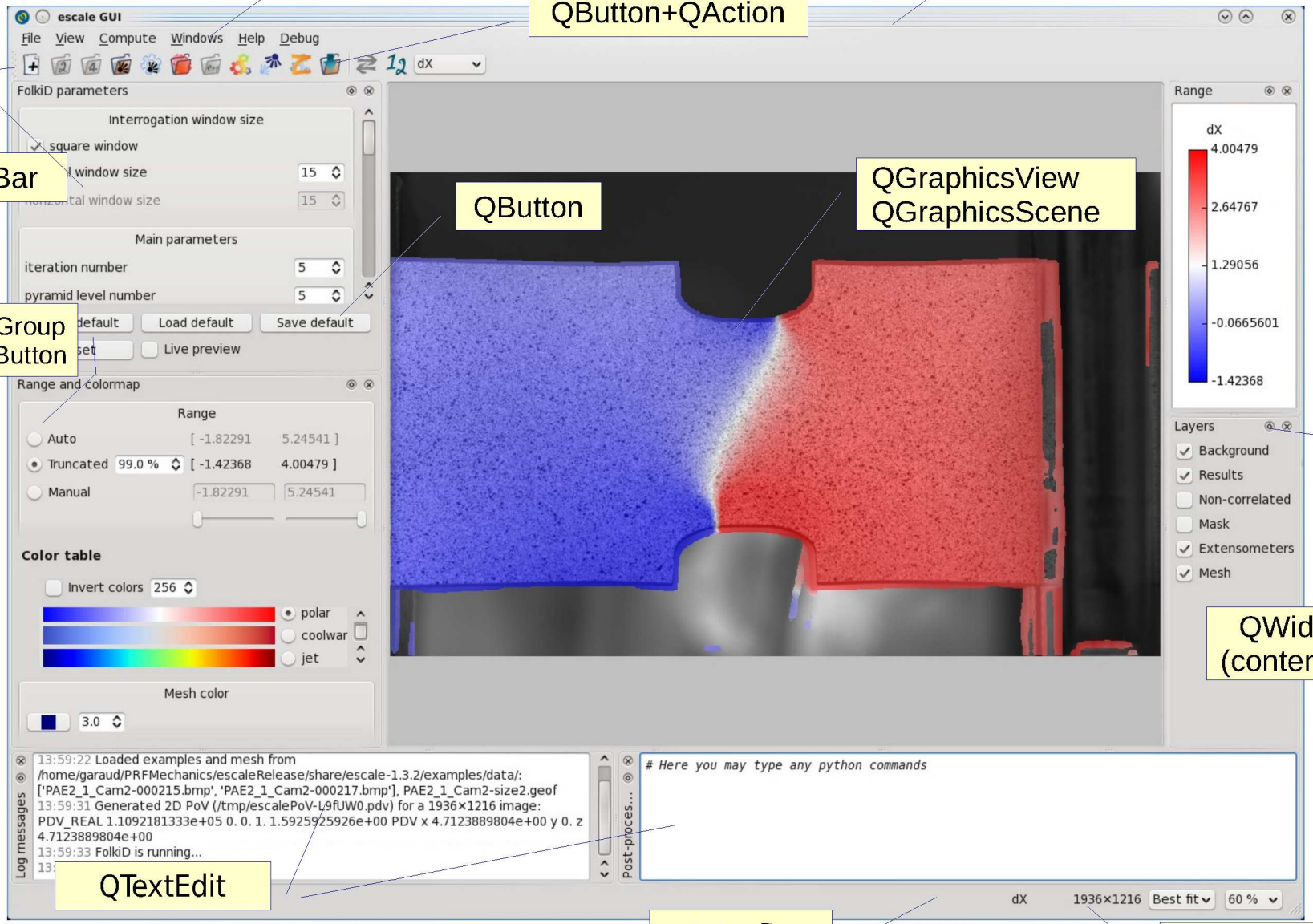
- Module python pour QT
  - toolkit très beau, portable
  - interface de programmation bien conçue
  - programmation événementielle
    - setup
    - event_loop
  - l'aspect "langage interprété & objet" est très bien adapté

- Concepts
  - tout ce qui se voit est *Widget*
    - fenêtre principale
    - popup
    - boutons
    - zone dessin
  - tout ce qui se passe est *event*
    - clic
    - drag-drop
    - tape au clavier

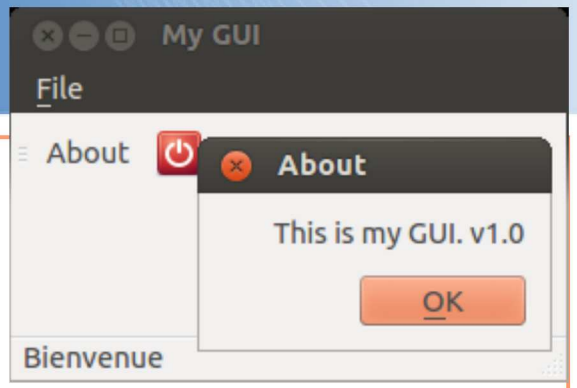
chronophage
mais ludique

http://srinikom.github.io/pyside-docs/index.html

Exemple


QMenuBar
QMainWindow
QButton+QAction
QToolBar
QButton
QGraphicsView
QGraphicsScene
QRadioGroup
QRadioButton
QWidget
(conteneur)
QTextEdit
statusBar
QLabel
escale GUI
File View Compute Windows Help Debug
FolkiD parameters
Interrogation window size
square window
Main parameters
iteration number
pyramid level number
Load default
Save default
Live preview
Range and colormap
Range
Auto
Truncated
Manual
Color table
Invert colors
polar
Mesh color
Range
dX
4.00479
2.64767
1.29056
-0.0665601
-1.42368
Layers
Background
Results
Non-correlated
Mask
Extensometers
Mesh
Log messages
Post-proces...
Here you may type any python commands
1936×1216
Best fit
60 %

GUI from scratch




```
from PySide import QtCore
from PySide import QtGui

class MyGUI(QtGui.QMainWindow):
    def __init__(self):
        QtGui.QMainWindow.__init__(self)
        self.setWindowTitle('My GUI')
        self.fileMenu = self.menuBar().addMenu("&File")
        self.toolBar = self.addToolBar("my toolbar")

        act = QtGui.QAction('About', self)
        act.triggered.connect(self.popup_hello)
        for x in (self.fileMenu, self.toolBar): x.addAction(act)

        act.setShortcuts(QKeySequence.Quit)
        act.triggered.connect(self.close)
        for x in (self.fileMenu, self.toolBar): x.addAction(act)

    def popup_hello(self):
        self.statusBar().showMessage('Bienvenue')
        QtGui.QMessageBox.about(self,"About", "This is my GUI. v1.0")

if __name__ == '__main__':
    qt_app = QtGui.QApplication(sys.argv)
    app = MyGUI()
    app.show()
    qt_app.exec_()
```

ed → event
(la grammaire est bien faite)

slot
(toute fonction est ok)

nombreux Widget
bien pratiques

TODO : pyQT & python3 ?

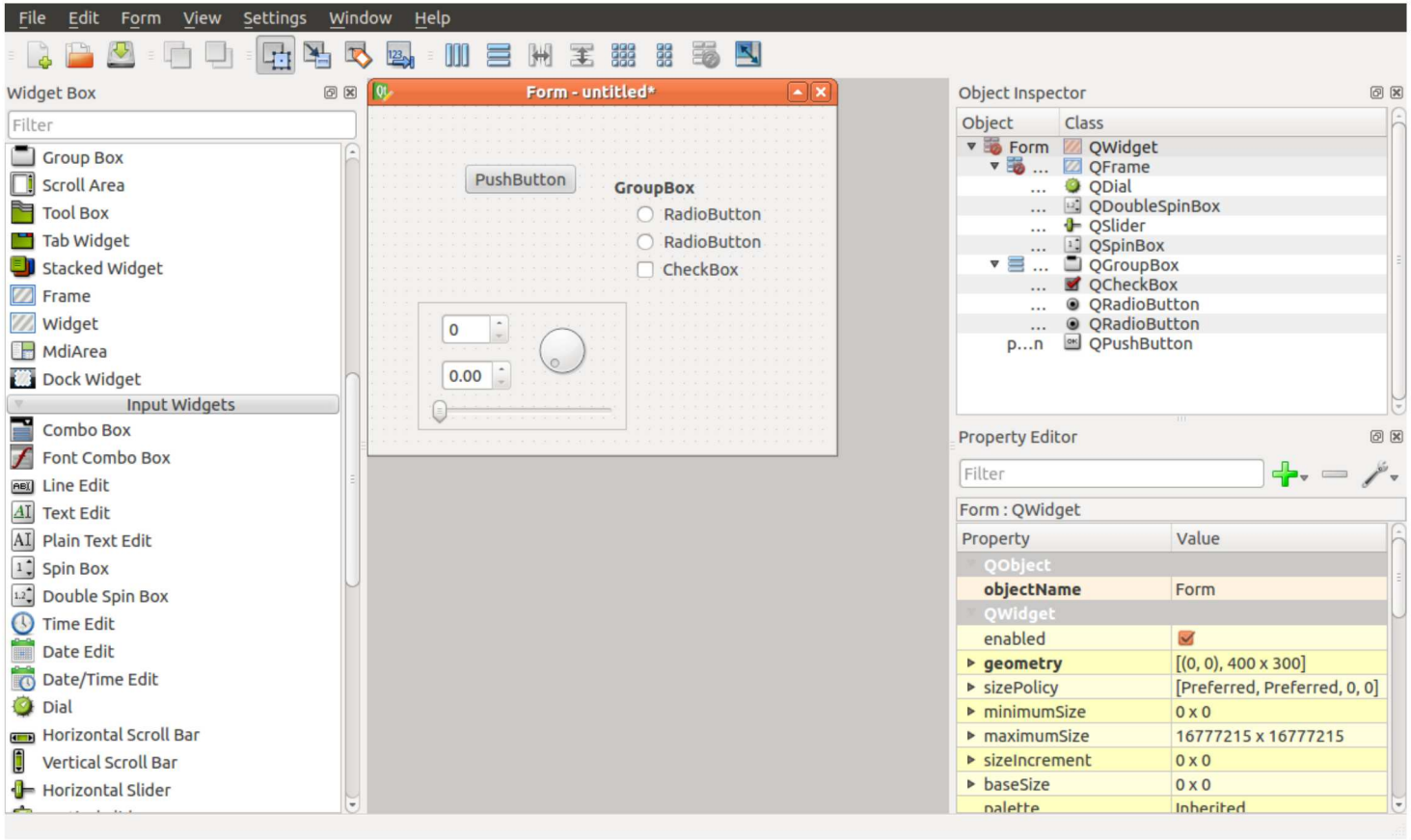
GUI from scratch

- Signal : emit ↔ connect
  - un peu similaire au concept des exceptions
  - le widget actif l'émet à son parent
  - qui le traite ou le ré-émet
  - certains émettent des arguments


```
my_spinbox.changed[int]connect(func)
def func(int_arg) :
    pass
```

- une GUI se conçoit donc en "réactions à chaque événement"
- conception non-linéaire, nécessite une bonne réflexion sur papier

QT Designer

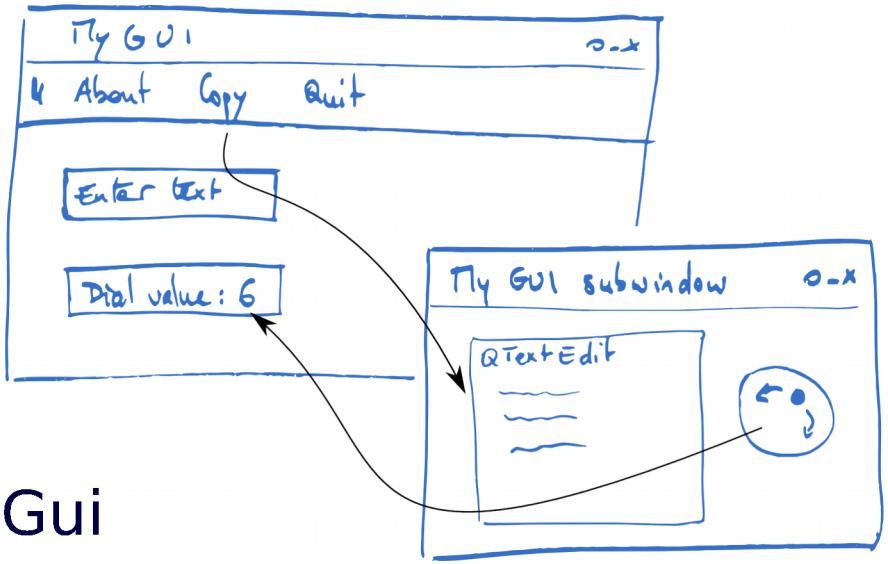


- Save as → my_gui.ui

```
pyside-uic -x my_widget.ui -o my_widget.py
python my_widget.py
```

TP 9 : créer une GUI

= 60mn



- Dans *designer*
  - créer un widget
  - avec un QTextEdit, un QSlider
- pyside-uic

- reprendre le canevas de MyGui
- ajouter un bouton
- le connecter pour afficher un message dans le QTextEdit

- dans MyGUI : self.setCentralWidget(QLabel())
- connecter la modification du QSlider au QLabel