GammaLib - Feature #2733

Make function fitting with GOptimizer possible for python scripts

11/09/2018 10:54 AM - Specovius Andreas

Status:	Closed	**Start date:**	11/09/2018
Priority:	Normal	**Due date:**	
Assigned To:	Knödlseder Jürgen	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	1.6.0		

Description

Adapt the Swig interface of GOptimizer (and correlated classes) to allow function fitting with gammalib from withing python.

Related issues:

Blocks ctools - Feature # 2723: Add example script to show significance distr...	**Closed**	**11/07/2018**

History

#1 - 11/09/2018 10:57 AM - Specovius Andreas

- Blocks Feature #2723: Add example script to show significance distribution in significance map added

#2 - 12/12/2018 10:55 AM - Knödlseder Jürgen

- Status changed from New to Pull request

- Assigned To set to Knödlseder Jürgen

- Target version set to 1.6.0

- % Done changed from 0 to 100

I added the GPythonOptimizerFunction class that implements a call back to GOptimizerFunction in Python. Below is an example script that fits a Gaussian function to some data points. Note that the eval() method has no arguments, and that the parameters are recovered using the private self._pars() method. I implemented this logic since I could not find out how to pass a GOptimizerPars instance in the C callback function.

The callback interface is implemented in the @GPythonOptimizerFunction.i file.

Note that appropriate high-level methods could be added that take care of the gradient and covariance matrix computation for a Chi-squared statistics. For the time being, this needs to be done by hand.

```
import math
import gammalib

# ============== #
# function class #
# ============== #
class funct(gammalib.GPythonOptimizerFunction):

    # Constructor
    def __init__(self, x_vals, y_vals):

        # Call base class constructor
        gammalib.GPythonOptimizerFunction.__init__(self)

        # Set eval method
        self._set_eval(self.eval)

        # Set data
        self._x_vals = x_vals
        self._y_vals = y_vals

    # Methods
    def eval(self):
        """
        Evaluate function
        """
        # Recover parameters
        pars  = self._pars()
```

```
        norm  = pars[0].value()
        mean  = pars[1].value()
        sigma = pars[2].value()

        # Evaluate function values
        y = [norm * math.exp(-0.5*(x-mean)**2/(sigma**2)) for x in self._x_vals]

        # Compute weights (1/sqrt(y))
        weight = [1.0/val if val > 0.0 else 0.0 for val in self._y_vals]

        # Compute Chi Square
        value = 0.0
        for i in range(len(self._x_vals)):
            arg    = self._y_vals[i] - y[i]
            value += arg * arg * weight[i]

        # Evaluate gradient and curvature
        sigma2 = sigma  * sigma
        sigma3 = sigma2 * sigma
        for i in range(len(x_vals)):

            # Evaluate function gradients
            dx     = self._x_vals[i] - mean
            dnorm  = y[i]         / norm   * pars[0].scale()
            dmean  = y[i] * dx    / sigma2 * pars[1].scale()
            dsigma = y[i] * dx**2 / sigma3 * pars[2].scale()

            # Setup gradient vector
            arg                  = (self._y_vals[i] - y[i]) * weight[i]
            self.gradient()[0] -= arg * dnorm
            self.gradient()[1] -= arg * dmean
            self.gradient()[2] -= arg * dsigma

            # Setup curvature matrix
            self.curvature()[0,0] +=  dnorm  * dnorm   * weight[i]
            self.curvature()[0,1] +=  dnorm  * dmean   * weight[i]
            self.curvature()[0,2] +=  dnorm  * dsigma  * weight[i]
            self.curvature()[1,0] +=  dmean  * dnorm   * weight[i]
            self.curvature()[1,1] +=  dmean  * dmean   * weight[i]
            self.curvature()[1,2] +=  dmean  * dsigma  * weight[i]
            self.curvature()[2,0] +=  dsigma * dnorm   * weight[i]
            self.curvature()[2,1] +=  dsigma * dmean   * weight[i]
            self.curvature()[2,2] +=  dsigma * dsigma  * weight[i]

        # Set value
        self._set_value(value)

        # Return
        return

# ======================== #
# Main routine entry point #
# ======================== #
if __name__ == '__main__':

    # Set parameters
    par1 = gammalib.GOptimizerPar('Norm',  300.0)
    par2 = gammalib.GOptimizerPar('Mean',  3.4)
    par3 = gammalib.GOptimizerPar('Sigma', 1.0)
    pars = gammalib.GOptimizerPars()
    pars.append(par1)
    pars.append(par2)
    pars.append(par3)

    # Set data
    x_vals = [1.0,   2.0,   3.0,   4.0,   5.0,  6.0]
    y_vals = [0.0, 100.0, 200.0, 200.0, 100.0, 10.0]

    # Set function
    fct = funct(x_vals, y_vals)

    # Optimize function
    opt = gammalib.GOptimizerLM()
    opt.optimize(fct, pars)
```

```
# Compute errors
opt.errors(fct, pars)

# Print function
print(opt)

# Print parameters
print(pars)
```

#3 - 12/12/2018 11:37 AM - Knödlseder Jürgen

- Status changed from Pull request to Closed

Merged into devel.