Fittino2 status report

Mathias Uhlenbrock

Physikalisches Institut, University of Bonn

Fittino Workshop

DESY, November 24, 2010

Table of contents

Motivation

Motivation

Last year we decided to refactorize Fittino (essentially to rewrite the code from scratch) in order to

- improve readability
- profit from the benefits of an OOP language such as C++ (make the code object-oriented)
- enhance flexibility where needed
- simplify the development of extensions
- add much more documentation
- in brief, to fit the needs of a rather big and ambitious software project with quite a number of developers
- produce reliable physics results

Purpose of this presentation

Purpose of this presentation

- give you a coarse overview on what has been done so far
- involve as many users as possible at this early stage of development when many fundamental things can still be discussed/changed
- having at hand a working setup with some functionality to have something more concrete to talk about
- introduce you to various supporting tools
- encourage you to have a look at the program and its implementation yourself and see if it fits our goals in a better way
- invite you to give feedback and to contribute

Functionality

Optimization mode

In optimization mode Fittino2 can find the minimum of a Rosenbrock test problem with three different optimization algorithms

- Minuit
- Simulated annealing
- Particle swarm optimization

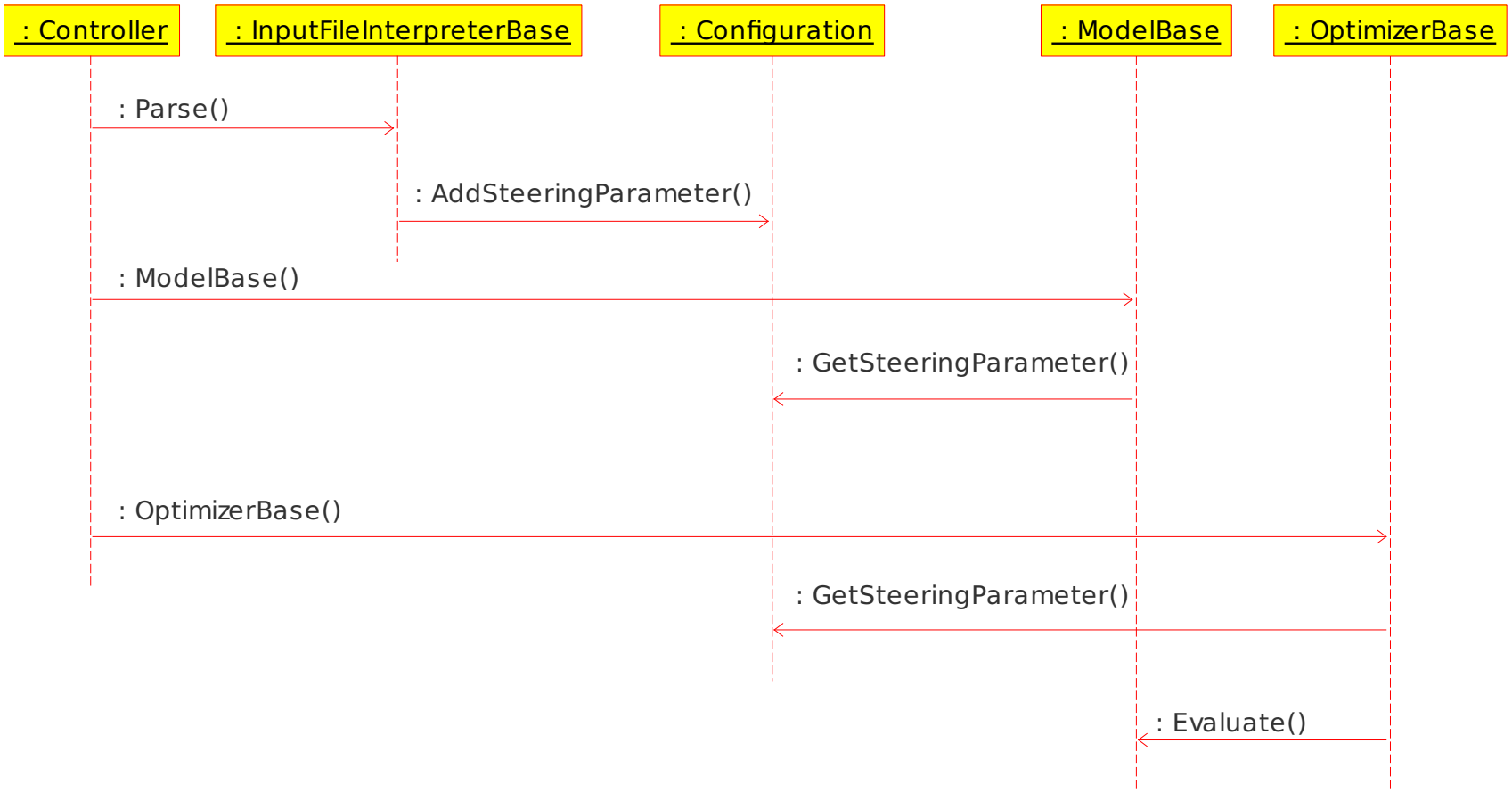
Sampling mode

In sampling mode it further can sample from the same fuction via a Metropolis-Hastings (MCMC) sampler.

Configuration

The tools and the model can be configured by the user via an XML-based input file.

Functionality: sequence diagram



Adressing the goals

Readability

- use coding conventions (one single style)
  supporting tool: `astyle`
- use speaking identifiers
- structurize code into objects
- use best practices: ”one function for one task”, etc.
- add documentation
  supporting tool: `doxygen`

Adressing the goals

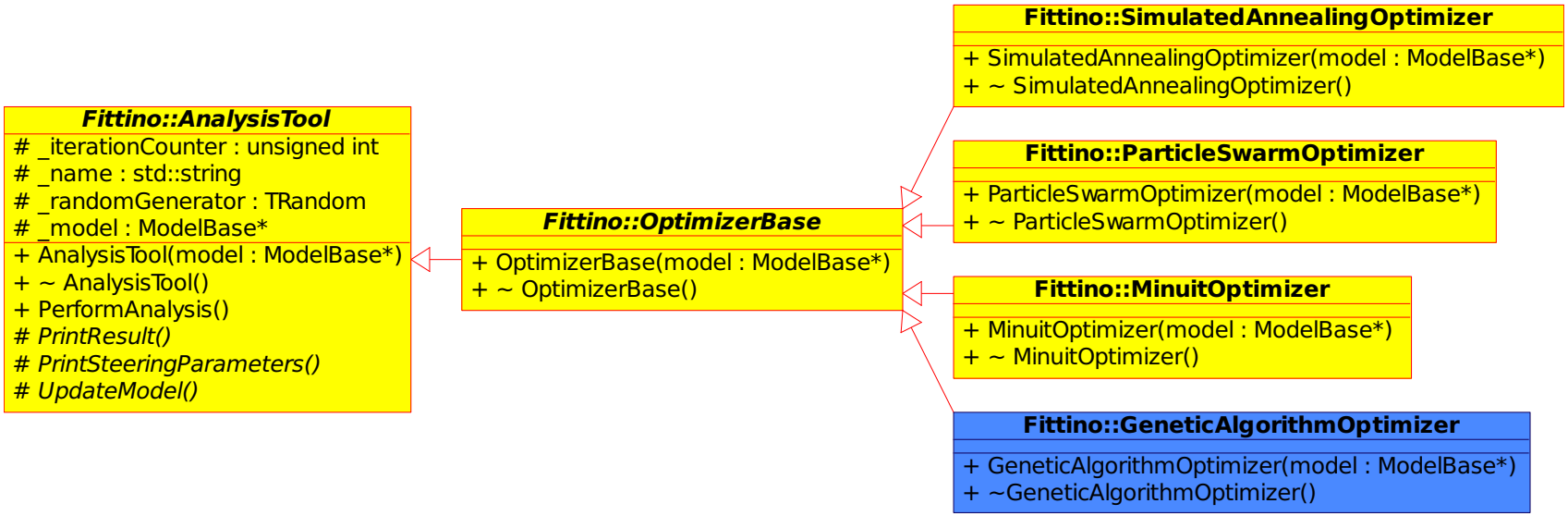
Object orientation

- restructurize code making extensive use of objects.
- make use of inheritance, share common needs/behaviour
- think about relationships among classes/objects: interfaces, patterns

List of classes

AnalysisTool	InputFileInterpreterBase	Parameter
Configuration	MarkovChainSampler	Particle
ConfigurationException	Messenger	ParticleSwarmOptimizer
Controller	MinuitOptimizer	RosenbrockFCN
ExceptionBase	ModelBase	RosenbrockModel
Factory	ModelCalculatorException	SamplerBase
FittinoInputFileInterpreter	MSUGRAFCN	SimulatedAnnealingOptimizer
GeneticAlgorithmOptimizer	MSUGRAModel	SLHAException
Individual	OptimizerBase	SLHAFileInterpreter
InputException	OptimizerException	XMLInputFileInterpreter

Selected design decisions 1: Inheritance



Inheritance

- example: optimizers
- share common needs/tools/interfaces, different levels of inter-class-relationships
- when extending the code only have to think about relevant functions
- minimize coding work with the help of templates

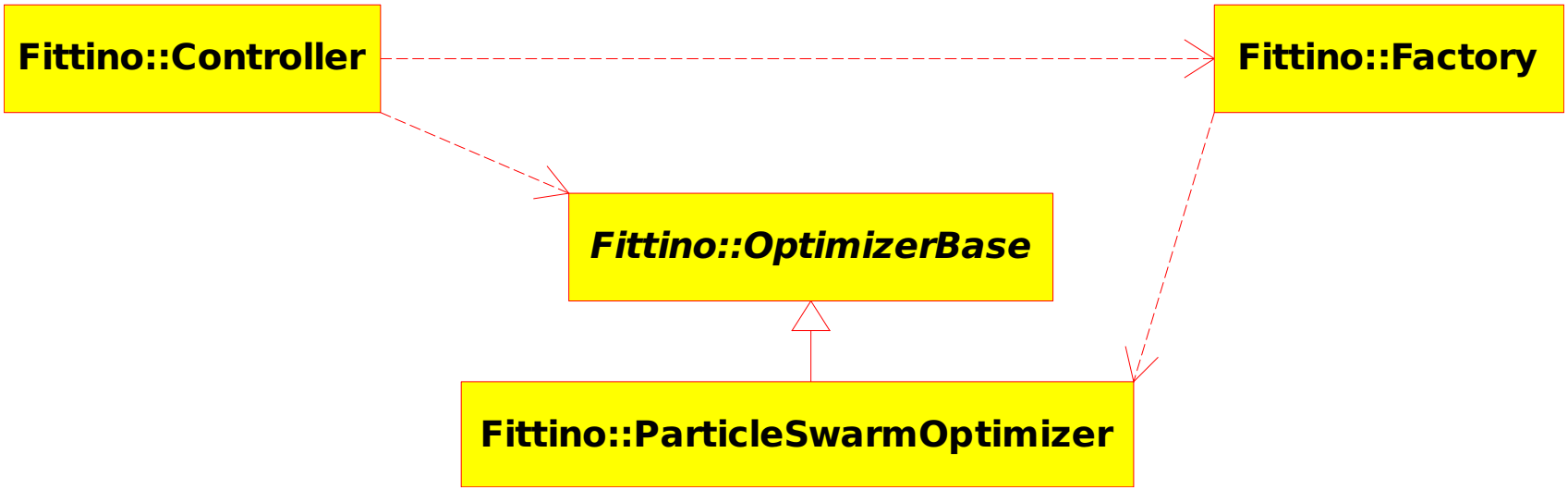
Selected design decisions 1: Inheritance

```
Fittino::SimulatedAnnealingOptimizer::SimulatedAnnealingOptimizer( Fittino::ModelBase
      * model )
         : OptimizerBase( model ) {

    _name = "simulated␣annealing␣optimization␣algorithm";
    _initialTemperature = Configuration::GetInstance()->GetSteeringParameter( "
          InitialTemperature", 5. );
    _temperatureReductionFactor = Configuration::GetInstance()->GetSteeringParameter(
           "TemperatureReductionFactor", 0.5 );
    _temperature = _initialTemperature;

}
```

Developer's guide: creating a new optimizer in two steps

- specify in the constructor the list of steering parameters
- implement UpdateModel()

Selected design decisions 2: Patterns



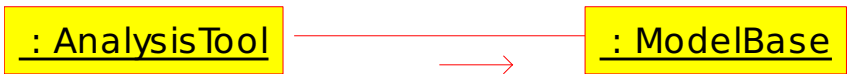
Flexibility

- make use of patterns (singletons, factories, template methods, etc.).
- (parametrized) factory pattern: generalizes concept of constructor
- used to build easily extendible list of combinations of objects

Selected design decisions 2: Patterns

```
// Create model.

const Factory factory;
ModelBase* const model = factory.CreateModel( Configuration::GetInstance()->
      GetModelType() );

if ( Configuration::GetInstance()->GetExecutionMode() == Configuration::OPTIMIZATION
      ) {

    // Create and run optimizer.

    OptimizerBase* const optimizer = factory.CreateOptimizer( Configuration::
          GetInstance()->GetOptimizerType(), model );
    optimizer->PerformAnalysis();
    delete optimizer;

}
else if ( Configuration::GetInstance()->GetExecutionMode() == Configuration::SAMPLING
       ) {

    // Create and run sampler.

    SamplerBase* const sampler = factory.CreateSampler( Configuration::GetInstance()
          ->GetSamplerType(), model );
    sampler->PerformAnalysis();
    delete sampler;

}
```

Selected design decisions 3: Associations




```
class AnalysisTool {
  ...
  protected:
    ...
    ModelBase* _model
    ...
  ...
}
```

Associations

- Associate analysis tools with models
- → one can combine every class deriving from AnalysisTool (all samplers, optimizers) with every class deriving from ModelBase (all models)

Development tools

astyle

- code beautifier
- gives code a more homogeneous view, can easily change layout
- usage: `astyle --options=devel/astyle-options <filename>`

UML diagrams

- graphical view of classes and their relationships/behaviour
- helps understanding class interplay and planning extensions
- project classes in `devel/Fittino.xmi`

Templates

- in `fittino2/devel` you can find generic class templates and optimizer templates

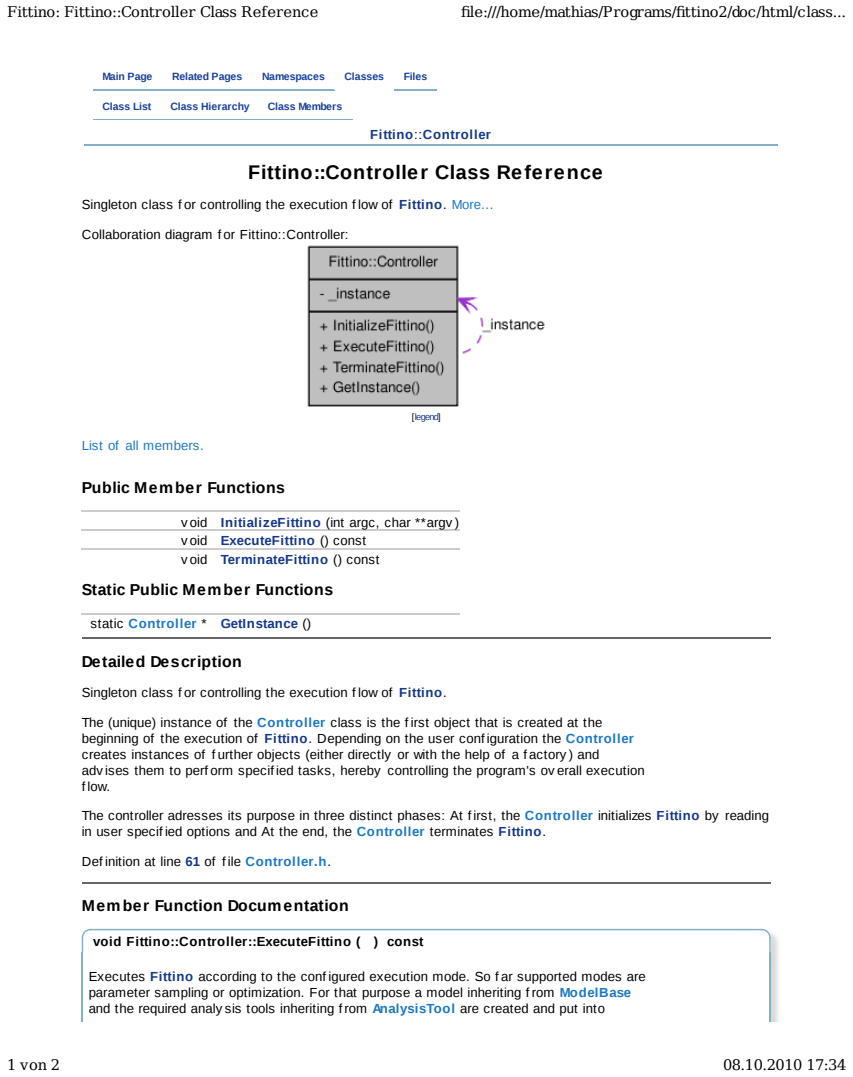
Documentation

Doxygen

- converts inline (and tagged) comments to hypertext pages
- like a (t)wiki page for code

Usage

- go to the `doc` directory:
  `cd fittino2/doc`
- run the program:
  `doxygen`
- open the documetation with your favourite browser:
  `firefox html/index.html`



Fittino: Fittino::Controller Class Reference file:///home/mathias/Programs/fittino2/doc/html/class...

Main Page | Related Pages | Namespaces | Classes | Files

Class List | Class Hierarchy | Class Members

Fittino::Controller

Fittino::Controller Class Reference

Singleton class for controlling the execution flow of **Fittino**. More...

Collaboration diagram for Fittino::Controller:

Fittino::Controller
- _instance
+ InitializeFittino()
+ ExecuteFittino()
+ TerminateFittino()
+ GetInstance()
_instance
[legend]

List of all members.

Public Member Functions

void	**InitializeFittino** (int argc, char **argv)
void	**ExecuteFittino** () const
void	**TerminateFittino** () const

Static Public Member Functions

static **Controller** *	**GetInstance** ()

Detailed Description

Singleton class for controlling the execution flow of **Fittino**.

The (unique) instance of the **Controller** class is the first object that is created at the beginning of the execution of **Fittino**. Depending on the user configuration the **Controller** creates instances of further objects (either directly or with the help of a factory) and advises them to perform specified tasks, hereby controlling the program's overall execution flow.

The controller adresses its purpose in three distinct phases: At first, the **Controller** initializes **Fittino** by reading in user specified options and At the end, the **Controller** terminates **Fittino**.

Definition at line **61** of file **Controller.h**.

Member Function Documentation

void Fittino::Controller::ExecuteFittino ( ) const

Executes **Fittino** according to the configured execution mode. So far supported modes are parameter sampling or optimization. For that purpose a model inheriting from **ModelBase** and the required analysis tools inheriting from **AnalysisTool** are created and put into

1 von 2 08.10.2010 17:34

Quick start guide

Installation of fittino2

Prerequisites

- root with libMinuit2.so, libXMLIO.so and libXMLParser.so libraries installed
- cmake

Installation

- Check out working copy of fittino2:
  `svn co svn://pi.physik.uni-bonn.de/fittino/branches/fittino2`
- Change to the fittino2/build directory:
  `cd fittino2/build`
- Configure cmake:
  `cmake ..`
- Build fittino2:
  `make`

Folder hierarchy

Directory name	Purpose
`fittino2`	Main directory
`+ build`	Location of the build tree
`+ CMakeModules`	Location of `cmake` scripts to add external programs (e.g. `root`, `SPheno`) as modules
`+ devel`	Location of supporting material for developers
`+ doc`	Location of documenting material
`+ input`	Location of various (example) input files
`+ logo`	Location fo the Fittino logo
`- sources`	Location of the source code files
`+ exceptions`	Location of the exception classes
`+ interpreters`	Location of the interpreter classes
`+ kernel`	Location of global classes
`+ models`	Location of the model classes
`+ optimizers`	Location of the optimizer classes
`- samplers`	Location of the sampler classes
`+ include`	Location of the header files
`+ src`	Location of the implementation files

First step: Running Fittino without arguments

```
fittino2/build$ ./sources/kernel/fittino

Usage: fittino [OPTION(S)] FILE

  A single given argument (different from "-h" or "--help") is
  interpreted as the name of an input file. The input file suffix
  must be .ftn (Fittino format) or .xml (XML format).
  Several example input files can be found at fittino2/input.

Supported options are:

  -h, --help
      Fittino prints this message.
  -i, --input-file=FILE
      Fittino uses the input file FILE. The input file suffix must
      be .ftn (Fittino format) or .xml (XML format).
      Several example input files can be found at fittino2/input.
  -s, --seed=SEED
      Fittino uses the given random number generator seed.
```

Second step: Running Fittino with an example input file

```
fittino2/build$ ./sources/kernel/fittino ../input/Example.optimization.in.xml

--------------------------------------------------------------------------------

  Welcome to Fittino

--------------------------------------------------------------------------------

  Reading configuration from file ../input/Example.optimization.in.xml

--------------------------------------------------------------------------------
.
.
.
```

Second step: Running Fittino with an example input file

```
.
.
.
------------------------------------------------------------------------------

  Initializing Rosenbrock model

   Starting values

    X              2.560000
    Y             -1.540000

------------------------------------------------------------------------------

  Initializing simulated annealing optimization algorithm

   Configuration

    Abort criterium              0.000010
    Number of Iterations         500
    Initial temperature          600.000000
    Temperature reduction factor 0.500000

------------------------------------------------------------------------------
.
.
```

Second step: Running Fittino with an example input file

```
.
.
.
------------------------------------------------------------------------------

  Running simulated annealing optimization algorithm

------------------------------------------------------------------------------

  Actual best set of Rosenbrock model parameters

    X            2.560000
    Y           -1.540000

    Chi2    6553.069696

------------------------------------------------------------------------------
.
.
.
```

Second step: Running Fittino with an example input file

```
.
.
.
-------------------------------------------------------------------------------

  Terminating simulated annealing optimization algorithm

-------------------------------------------------------------------------------

  Optimization converged after 252 iterations

-------------------------------------------------------------------------------

  Optimization results

   Final set of Rosenbrock model parameters

    X           1.002403
    Y           1.004813

-------------------------------------------------------------------------------
```

Third step: Running Fittino with a customized input file

```
+
<!--
  The ExecutionMode node specifies the main execution mode of Fittino. If
  OPTIMIZATION is set, Fittino tries to find the optimal parameter set of
  a given model w.r.t. experimental observation.
-->
<ExecutionMode
 Mode = "OPTIMIZATION"
 />
```

Third step

- have a look at the other example: Example.sampling.in.xml
- open an example file. The available steering parameters and their meaning should be sufficiently documented
- manipulate numerical values of various steering parameters an see how behaviour of Fittino changes

Todo list/next steps

Todo list

Feedback and help from your side is always highly welcome. Choose a task according to your motivational and/or time constraints:

- long-term: major strategic decisions
- mid-term: extensions supported by the framework, on class level
- short-term: small modifications, improvements on what basically exists, often on single function level

An up-to-date list should be available from the documentation under `Related Pages/Todo list`.

Next steps: get physics in

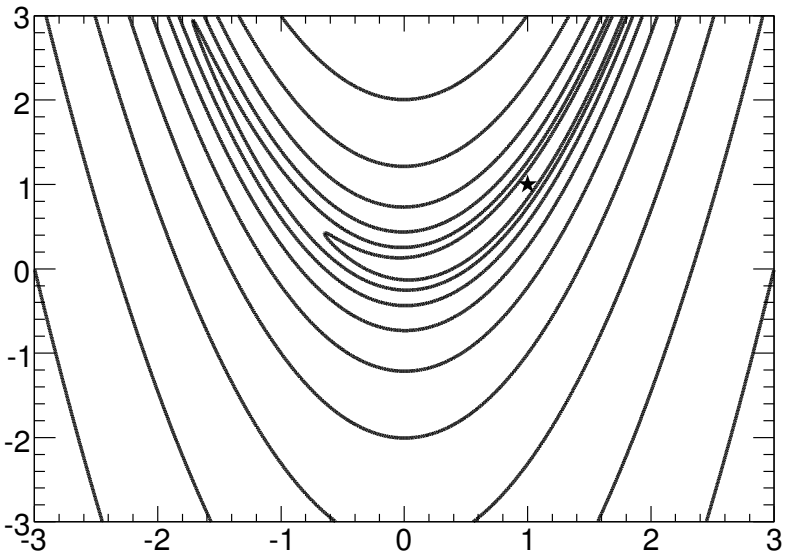
- write a SLHA interpreter, using much of Ben O'Leary's code. (ongoing)
- write output handling classes
- add physics observables
- add physics models

Summary

- showed you the status of the development of Fittino2
- showed you functionality, some design aspects, quick start guide
- introduced you to supporting tools
- invite you to give feedback
- if you are interested in contributing, don't hesitate to contact me
- further improve the existing code, make Fittino2 applicable to physics problems

Backup slides

Test problem: 2D Rosenbrock function



Properties

- narrow, parabolic shaped, flat valley
- Global minimum at (1,1)
- finding valley trivial, but finding minimum difficult

2D Rosenbrock function

$$f(x,y) = (1-x)^2 + 100(y-x^2)^2$$

$$f(1,1) = 0$$