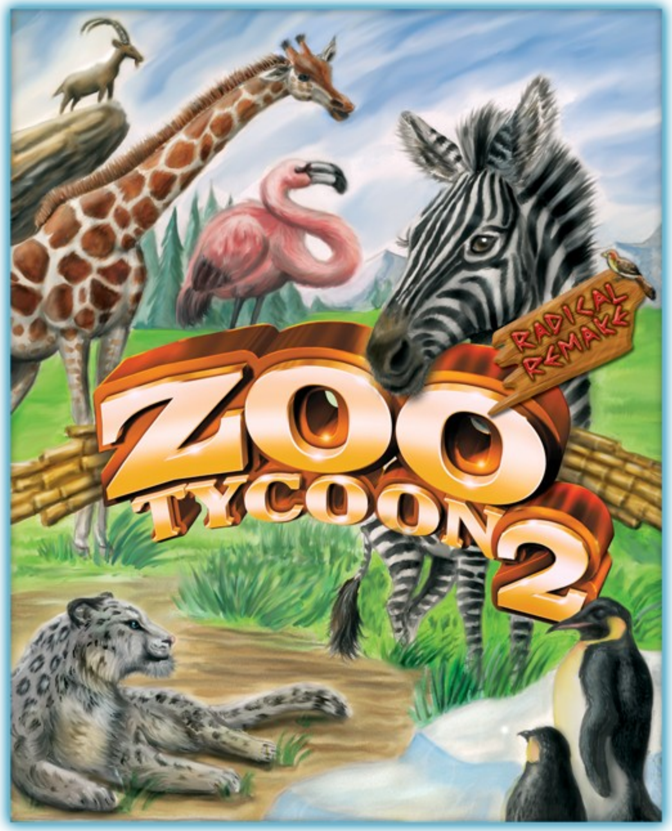
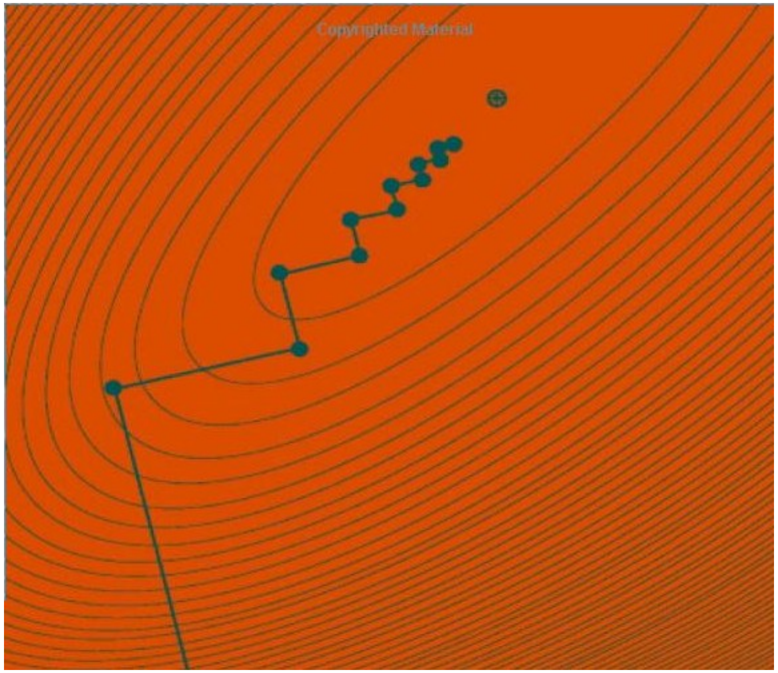
(6) iFit/iOptim:
Optimizer zoo



Optimizers: concepts

« Houston, we have a problem ! »



A problem has solutions.

We quantify a problem with a *criteria* (*objective* or *cost function*), which is a measure of the problem. Usually we wish to minimize it.

The handles for minimization are the *parameters*. The number of parameters is the *dimensionality* of the problem.

We change the parameters according to an *algorithm (optimizer)*, and monitor the evolution of the criteria. We choose the solutions that e.g. lower the criteria. The *success ratio* of the optimizer measures the probability to solve a problem. The number of iterations (criteria evaluations) is the *cost* or *budget*. The search for a solution usually starts from a *guess* in the parameter space.

Various type of optimizers

The optimization algorithms can be classified in a number of categories:

- **Gradient** and Hessian based methods are, mostly, deterministic. Their convergence is granted in a limited number of iterations. They are pretty **fast** for small-medium dimensionality. However, as they require derivatives, they are very **sensitive to noise**, and get very slow for many parameters. They also can easily be **trapped** around non-global solutions, depending on noise and starting parameter set. *Newton* and *Levenberg-Marquardt* optimizers are the most famous in this class.
- **Deterministic derivative-free** methods are usually slower, but more **robust**, and can search for global optimization (less chances to be trapped). *Simplex*.
- **Line-search methods** identify preferred directions for the search (e.g. gradient), and find local minima along the line, before changing direction.
- **Trust-region** algorithms use a scale-down search. The search region is gradually reduced in size.
- **Heuristic** methods do not ensure convergence, but as most of them use random numbers, they are well suited for **global optimization**, and hardly get trapped. Very **robust**. However, they are usually slow (high *cost*). *Swarms, anneal, genetic.*

Optimizers in iFit: syntax

We first define the **objective** function, with input argument *p*, which we want to minimize. The vector *p* length is the **dimensionality**.

```
function y=objective(p)
  y =... % a function of p
```

Then we choose a starting parameter set (guess), ***p0***.

Then we start the optimizer with syntax:

```
fminsearch('objective', p0)
```

And we wait... (not for ever).
It returns the solution which minimizes objective.

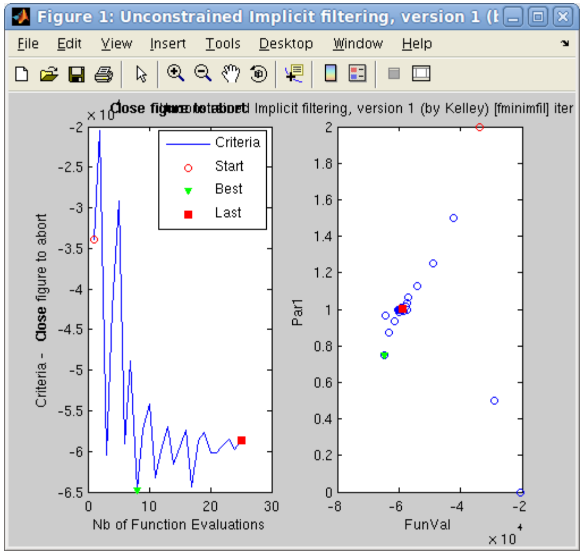
Configuration of optimizers

It is possible to configure the optimizer with an additional argument

```
fminsearch('objective', p0, options)
```

Were *options* is a structure (type *help optimset* for doc)

options.OutputFcn='fminplot' will monitor the optimization.



Optimization results

The optimizer returns up to 4 output arguments.

```
[parameters,criteria,message,output] = fminsearch('objective',p0 ...)
```

Optimization results

The optimizer returns up to 4 output arguments.

```
[parameters,criteria,message,output] = fminsearch('objective',p0 ...)
```

What you are looking for

Optimization results

The optimizer returns up to 4 output arguments.

```
[parameters,criteria,message,output] = fminsearch('objective',p0 ...)
```

objective(parameters)
Minimal criteria

Optimization results

The optimizer returns up to 4 output arguments.

```
[parameters,criteria,message,output] = fminsearch('objective',p0 ...)
```

Final state
of the optimizer

Optimization results

The optimizer returns up to 4 output arguments.

```
[parameters,criteria,message,output] = fminsearch('objective',p0 ...)
```

All the rest !

Constraints (restraints)

It is possible to define constraints on the parameters (aka *restraints*) with 4th and 5th input arguments

To fix parameters, use:

```
fminpowell('objective',p0, options, [ 1 0 0 1 ...])
```

were the 4th input is a vector of 0 (free) and 1 (fixed), with length that of *p0*.

To limit the parameter search to an hypercube:

```
fminpowell('objective',p0,options,[min1 min2...], [max1 max2...])
```

were min and max values define the range (*nan* and *inf* are possible)

There is also a way to limit the parameter change, but I never used it.

Available optimizers in iFit

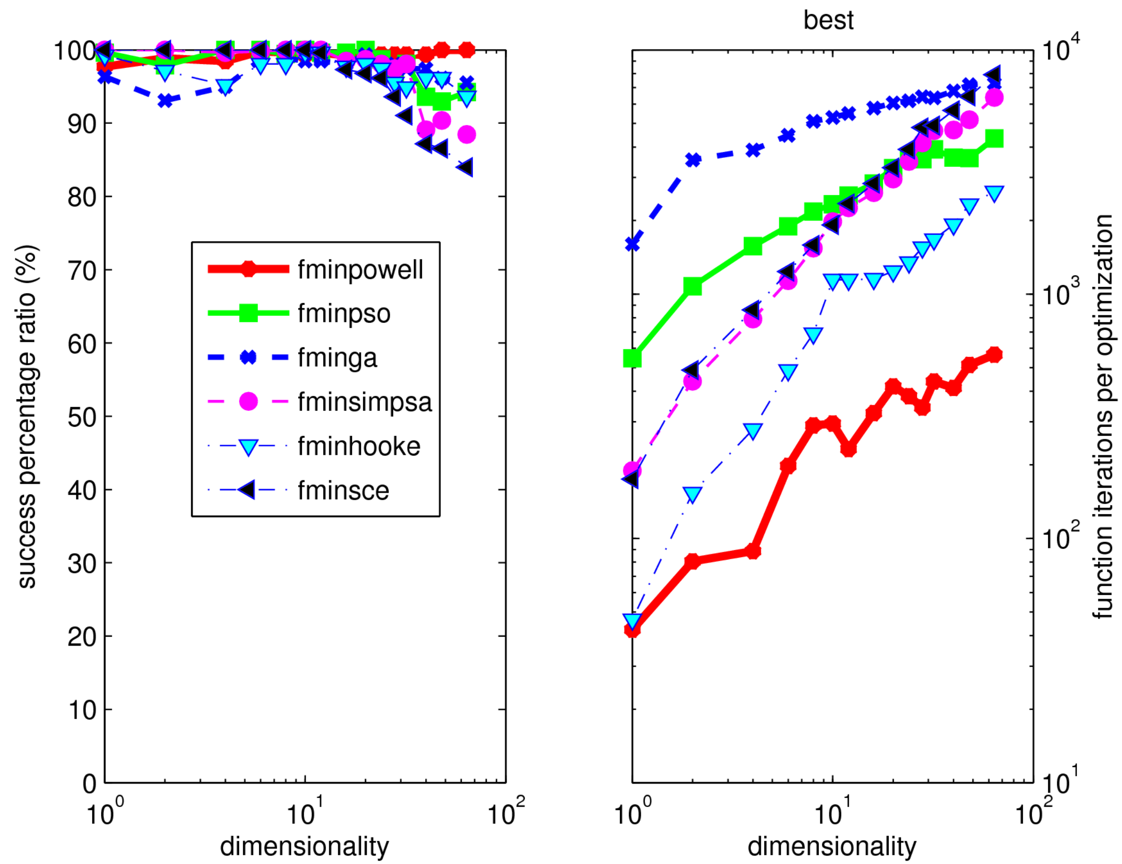
		Continuous problems		Noisy problems	
Function Name	***Description***	***Success rate [%]***	***Solving time [s]***	***Success rate [%]***	***Solving time [s]***
fminanneal	Simulated annealing [19]	53.6	1.46	5.3	0.87
fminbfgs	Broyden-Fletcher-Goldfarb-Shanno [20,21,22,23]	83.9	0.94	2.5	0.04
fmincgtrust	Steihaug Newton-CG-Trust [21,22,23,24,25]	87.4	0.47	4.1	0.37
fmincmaes	Evolution Strategy with Covariance Matrix Adaptation [15,26]	86.3	15.7	59.5	9.8
fminga	Genetic Algorithm (real coding)	84.1	66.2	55.5	38.08
fmingradrand	Random Gradient [27]	62.6	9.5	13.1	1.96
fminhooke	Hooke-Jeeves direct search [25,28,29]	94.6	8.97	38.8	8.13
fminimfil	Implicit filtering [25]	92.7	9.81	40.5	6.02
fminkalman	unscented Kalman filter [30]	63.6	29.1	7.6	14.35
fminlm	Levenberg-Maquardt [21,31]	14.2	12.4	1.8	72.85
fminnewton	Newton gradient search [25]	79.1	0.02	1.6	0.01
fminpowell	Powell Search [32]	96.6	0.66	30.7	17.79
fminpso	Particle Swarm Optimization [41,57]	97.0	18	69.7	13.76
fminralg	Shor r-algorithm [33]	88.3	0.03	3.3	0.55
fminrand	adaptive random search [34]	60.7	47.8	44.7	30.79
fminsce	Shuffled Complex Evolution [35,57]	88.0	46.3	65.7	18.7
fminsearchbnd	Nelder-Mead simplex (*fminsearch*) [36]	55.3	1.37	5.4	1.61
fminsimplex	Nelder-Mead simplex (alternate implementation than *fminsearch*) [37]	73.3	1.22	30.0	0.54
fminsimpsa	simplex/simulated annealing [38,39,57]	94.7	26.5	67.7	24.16
fminswarm	Particule Swarm Optimizer (alternate implementation than *fminpso*) [40]	78.4	23.1	50.2	21.88
fminswarmhybrid	Hybrid Particule Swarm Optimizer [40,41]	80.5	12.7	24.1	18.56

Choosing an optimizer: continuous problems

Gradient methods are good for continuous problems.

Fast and high success ratio:
- Powell (fminpowell)
- R-algorithm (fminralg)
- Steihaug Newton-CG-Trust (fmincgtrust)
- Royden-Fletcher-Goldfarb-Shanno (fminbfgs)

Slower but very robust:
- Particle swarm (fminpso)

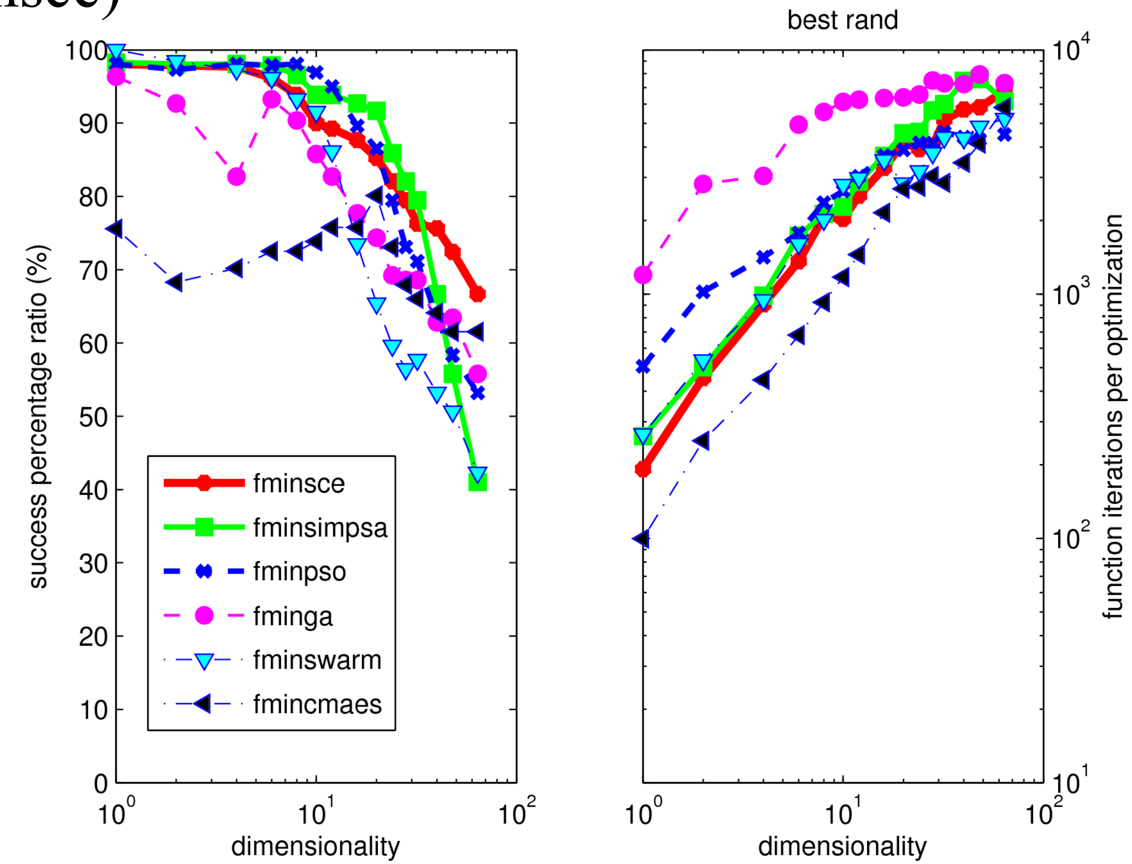


Choosing an optimizer: noisy problems

Heuristic methods are good for noisy problems.

Not too slow and high success ratio:

- Particle swarm (fminpso)
- Simplex+simulated annealing (fminsimpsa)
- Shuffled Complex Evolution (fminsce)



A noisy objective: McStas

If you wish to optimize a McStas simulation, the required objective function has been written in a general way. The syntax is

```
[parameters, monitors, message,output] = mcstas('instr', p0, options)
```

- parameters: What you are looking for
- monitors: Final McStas data
- 'instr': The instrument file name
- p0: Instr. Parameters as a **structure**

A noisy objective: McStas

If you wish to optimize a McStas simulation, the required objective function has been written in a general way. The syntax is

```
[parameters, monitors, message,output] = mcstas('instr', p0, options)
```

The simulation settings

The options specify the type of simulation you want:

- McStas usual fields (ncount, dir, mpi, …)
- Optimization fields (TolFun, Display, OutputFcn)
- Type of computation (Simulation, Optimization)
- The definition of the criteria: options.monitors

Exercise: try optimizers

Define the function:

banana = @(x)100*(x(2)-x(1)^2)^2+(1-x(1))^2;

Define two parameter vectors x1=x2=linspace(-2,2,50);

Map the banana criteria value with 2 loops

b(i1,i2)=banana([x1(i1) x2(i2)])

Plot the surface and identify where the minimum is

surf(x2,x1,b); caxis([0 5])

Now launch any optimizer and compare execution time, found solutions

fminpowell(banana, [-1 1])

fminpso(banana, [-1 1])

Tune the options.TolFun value, set options.TolX=0.

Get the 4^{th} optimizer output parameter and plot the search history from output.parsHistory and output.criteriaHistory.

What is your conclusion about optimizers ?

Exercise: optimize a McStas simulation

Open McStas and load the instrument
Neutron site/Templates/templateDIFF
Copy this instrument to your Matlab location
Execute a single simulation with p0.RV=-1 as parameter, **from Matlab**.
Plot the monitors from the simulation from Matlab.

Repeat the simulation with p0.RV=0.5:0.1:1.5 to do a **parameter scan**. Plot the results.
What is the optimal RV value ?

Restart with an **optimization**, using the same p0 and options.optimizer='fminpso' (or an other one).

What is the optimal RV value ? Compare with the optical value, indicated when the instrument simulation starts.