559

Non-linear Minimization in R

Fish 559; Lecture 89

Non-linear Minimization Functions in R

- Single-dimension methods
  - **optimize** – find the minimum (or maximum) of a univariate function within pre-specified bounds.

559

Non-linear Minimization Functions in R

(Multi-dimension methods)

- **nls** – fit a regression model using non-linear least squares. This function produces almost all of the diagnostics associated with **lm**, **lme** and **nlme**.
- **dfp** – minimize an objective function using the Davidson-Fletcher-Powell algorithm and compute 95% intervals (in library **Bhat**).
- **mle -** minimize a real-valued function *f* subject to constraints (in library **stats4**). Assumes fx in nll, computes hessian and inverts it for you – gives you variance
- **newton** – minimize an objective function using a Newton-Raphson algorithm and compute 95% intervals (in library **Bhat**).
- **optim** – minimize a real-valued function *f* subject to constraints. Family of functions optimx() – calls multiple minimization methods for you.
- **nlminb** – minimizes an constrained (or unconstrained) function.

559

Non-linear Minimization in R

- **Optim/mle** applies one of several methods (quasi-Newton [BFGS], simplex [Nelder-Mead], and conjugate-gradient[CG])and can make use of derivatives and the Hessian matrix (if they are available). It also includes options to use Simulated Annealing (SANN) – monte-carlo method, very robust but very slow).

559

Calling **optim**-I

optim(pars, fn, gr, method, control, lower, upper, hessian)

- Required parameters:
  - **pars** – a vector of parameters (the initial guesses).
  - **fn** – the objective function – that which is to be minimized.
- Optional parameters:
  - **gr** – function that computes the gradient.
  - **hessian** – should the Hessian matrix be computed?

559

Calling **optim**-II

- Optional parameters:
  - **method** – which of the various types of minimization algorithm to use.
  - **control** – parameters used to control the minimization (maximum iterations, function calls, etc.).
  - **rel.tol** – how much change in the function before you stop 10e-10 means it's going to try very hard to get a minimum
  - **lower** / **upper** – vector of lower and upper bounds for the parameters (including –Inf / Inf).

559

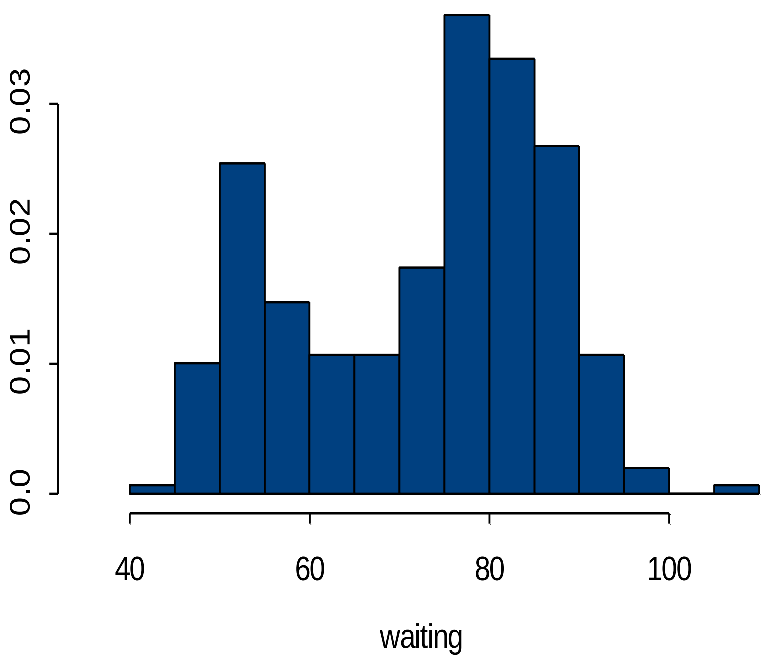
Calling **optim**-III

- (Key) outputs:
  - **par** – the estimates of the parameters.
  - **value** – the lowest value of the objective function.
  - **counts** – number of function and gradient calls.
  - **convergence** – code indicating whether convergence occurred. 0 means it thinks it converged >1 means not
  - **message** – additional information
  - **hessian** – estimate of the Hessian matrix at the solution.
  - **Deviance** – twice the nll making 3.84 the critical value, should be 1.92 for likelihood profile. 1.96 is 2*se assuming asymptotic normal.

A first example-I (The waiting data)

559

- The times between eruptions of "Old Faithfull" (dataset "waiting") appear to be bimodal. We would like to fit these data to a model that is a mixture of two normals.



$$P(x) = \pi N(x \mid \mu_1, \sigma_1) + (1-\pi) N(x \mid \mu_2, \sigma_2)$$

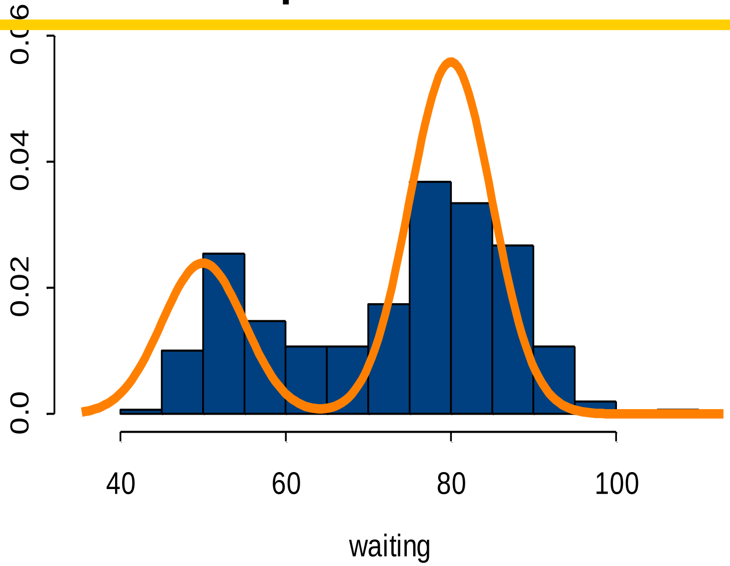
We can get sensible estimates for the parameters of this model (0.3, 50, 5, 80, 5) from the histogram.

truehist(geyser$waiting,xlim=c(35,110),h=5)

559

A first example-II (The waiting data)

The initial guesses for the parameters are not bad but we can improve on them.



We first write a function that computes the negative log-likelihood.

$$\sum_{i=1}^{n} \ell n\left[\frac{\pi}{\sigma_1}\phi\left(\frac{y_i-\mu_1}{\sigma_1}\right)+\frac{1-\pi}{\sigma_2}\phi\left(\frac{y_i-\mu_2}{\sigma_2}\right)\right]$$

```
mix.obj <- function(pp,x) {
 e <- pp[1]*dnorm((x-pp[2])/pp[3])/pp[3]+
        (1-pp[1])*dnorm((x-pp[4])/pp[5])/pp[5]
 return(-sum(log(e)))  }
```

A first example-III (The waiting data)

559

We can now use **optim** to minimize the negative log-likelihood function.

Initial values

```
wait.init <- c(0.3,50,5,80,5)

mix.n1 <- optim(wait.init, mix.obj,method='L-BFGS-B',
lower=c(0,-Inf,0,-Inf,0),upper=c(1,rep(Inf,4)),x=geyser$waiting)
```

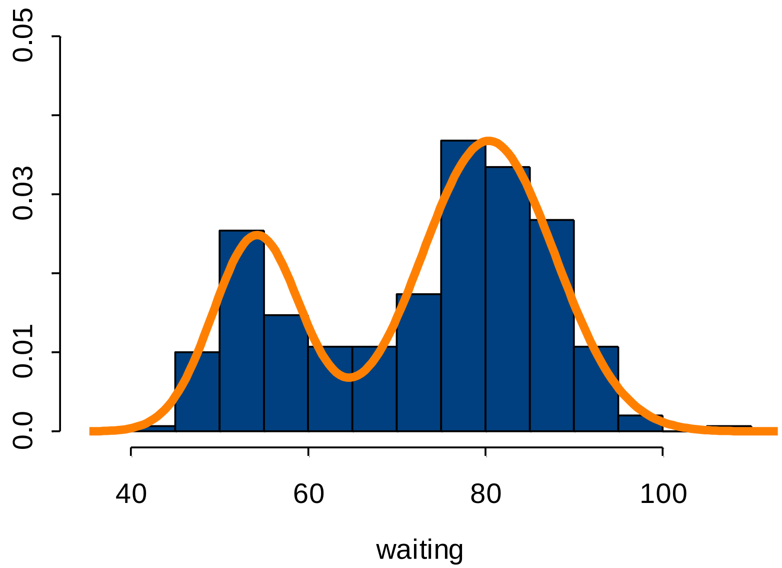
Place bounds on the parameter values

Pass the data

A first example-IV (The waiting data)

559

- The results object should be examined to identify that the minimum has indeed been found.



A first example-V (The waiting data)

Our analysis did not use information on the derivatives of the likelihood function. This is a case where it is possible (and relatively straightforward) to make use of derivative information.

```
lmix2a <- deriv( ~-log(p*dnorm((x-u1)/s1)/s1 + (1-p)*dnorm((x-u2)/s2)/s2),
      c("p","u1","s1","u2","s2"),   function(x,p,u1,s1,u2,s2) NULL)

 mix.gr <- function(p,x) {
   u1 <- p[2]; s1 <- p[3]; u2 <- p[4]; s2 <- p[5]; p <- p[1]
   e <- lmix2a(x,p,u1,s1,u2,s2)
   rep(1,length(x)) %*% attr(e,"gradient") }


mix.n11 <-optim(wait.init, mix.obj, mix.gr, method="L-BFGS-B",
       lower=c(0,-Inf,0,-Inf,0),upper=c(1,Inf,Inf,Inf,Inf),x=waiting)
```

Calling **mle** - I

mle(minuslogl, start, method, fixed,lower,upper)

- **minuslogl** – the negative log-likelihood function – that which is to be minimized.
- **start** – a named list of estimated parameters (the initial guesses).
- **fixed** – a named list of parameters which are passed to the objective function.
- **method** – which of the various types of minimization algorithm to use.

559

Calling **mle** - II

- The methods associated with **mle** can be used to:
  - Extract the log-likelihood (**logLik**)
  - Extract the estimated parameters (**coef**)
  - Extract the variance-covariance matrix (**vcov**)
- **mle** provides estimates of the standard errors of the estimates automatically (unlike **optim**, which require you invert the Hessian matrix).
- Hint: **mle** makes computing profile likelihood very easy (compared to **optim**)

559

The Example Problem

- Fit the dynamic Schaefer model to the catch and effort data for Cape hake off the west coast of South Africa.

$$B_{t+1} = B_t + rB_t(1 - B_t / K) - C_t; \; B_{1917} = K$$

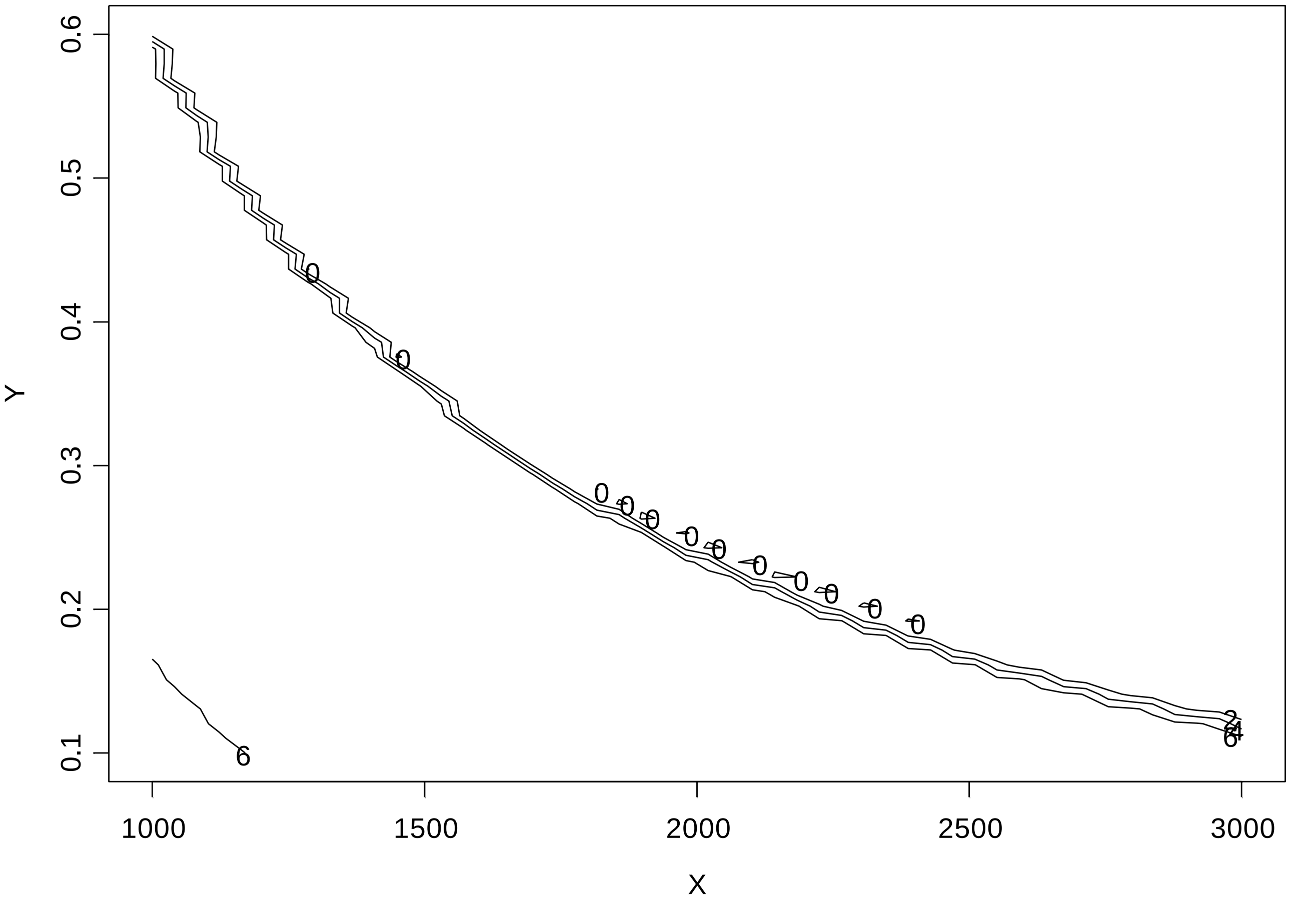
$$-\ell nL = \sum_y \left(\ell n I_y - \ell n(q\, B_t) \right)^2$$

- We will make use of the maximum likelihood estimate for *q*:

$$\ell n q = \tfrac{1}{n} \sum_y \ell n(I_y / B_y)$$

The Example Problem (The sum of squares surface)

559



559

The Example Problem (The methods)

- We could apply methods that are:
  - Derivative free:
    - Simplex / Powell's method
  - Derivative based:
    - optim
- Note: that we will not supply **optim** with analytic derivatives nor use **deriv** to compute them for us (why?)

559

Passing Parameters

- The likelihood function depends on data (catches, CPUE) as well as on the values for the model parameters (*r*, *q* and *K*).
- It is necessary therefore to pass the data to the function being minimized.
- There are several ways to do this but I prefer to place common parameters in frame 1 so that my function can "see" them.

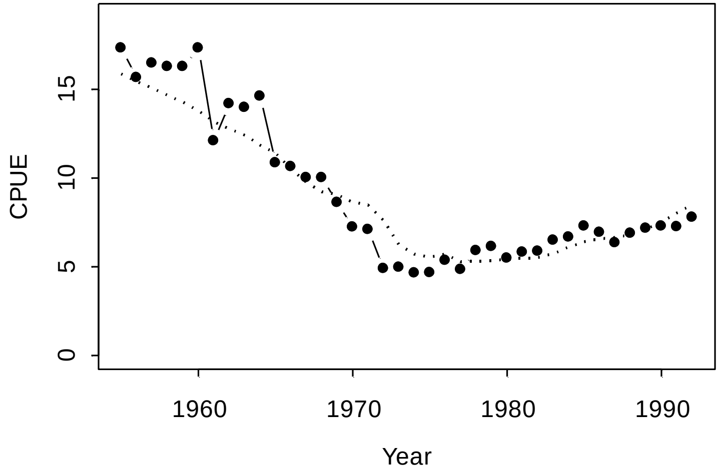
```
co <- list()
co$Final <- F
co$Nyear <- Nyear
co$Catch <- TheData$Catch
co$CPUEObs <- TheData$CPUE
assign("co",co,pos=1)
```

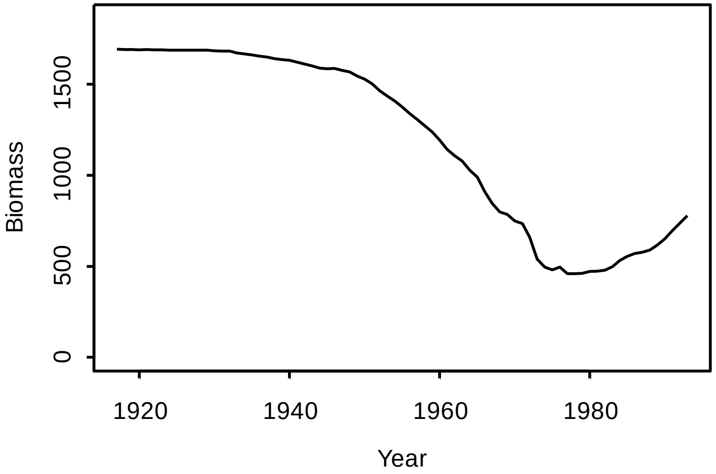
559

The Function to Minimize

```
f1 <- function(x)
{
 K <- exp(x[1]);  r <- exp(x[2])
 CpuePred <- rep(0,co$Nyear); Biomass <- rep(0,co$Nyear+1)
 # Do projections
 Biomass[1] <- K
 for (Year in 1:co$Nyear)
  {  Biomass[Year+1] <- Biomass[Year] + r*Biomass[Year]*(1.0-Biomass[Year]/K) -
co$Catch[Year]
if (Biomass[Year+1] < 0.01) Biomass[Year+1] <- 0.01   }
 # Calculate the ML estimate of q
 qbar <- 0; npar <- 0
 for (Year in 1:co$Nyear)
  if (co$CPUE[Year] > 0)
   { npar <- npar + 1; qbar <- qbar + log(co$CPUE[Year]/Biomass[Year]) }
 qbar <- exp(qbar / npar)
 # Find the SS
 SS <- 0
 for (Year in 1:co$Nyear)
  if (co$CPUE[Year] > 0)
   {  CpuePred[Year] <- qbar*Biomass[Year]
                SS <- SS + log(CpuePred[Year]/co$CPUE[Year])^2   }
 return(SS) }
```

Results - **optim**



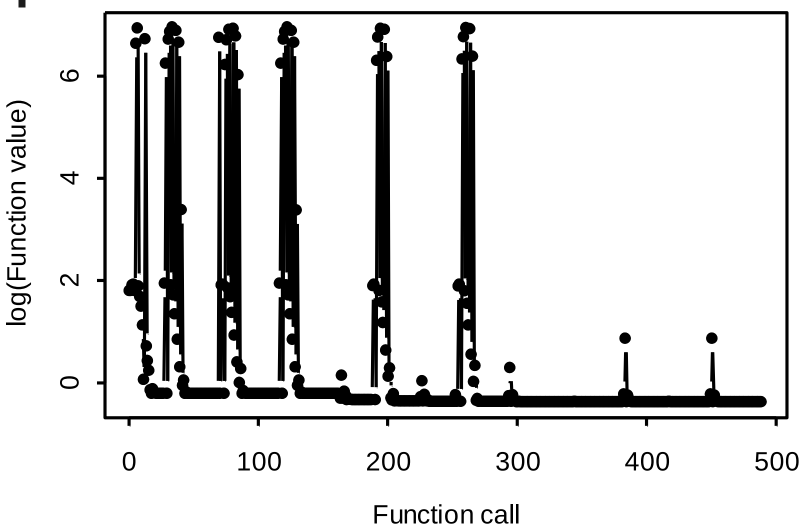


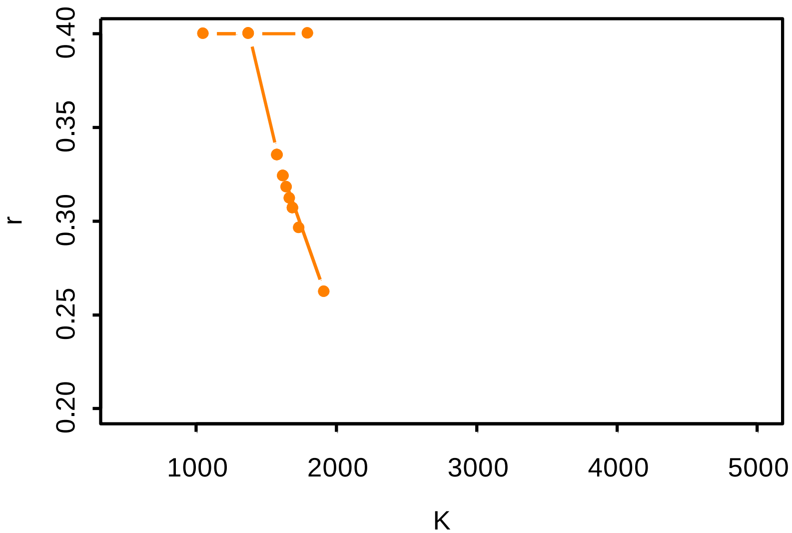
SS=0.6924
r=0.335
K=1562

Function evaluations:65
Gradient evaluations:65

Note: the graphs were produced by adding code to the function. The graphs are output if 'final'=T

Results - **Powell**

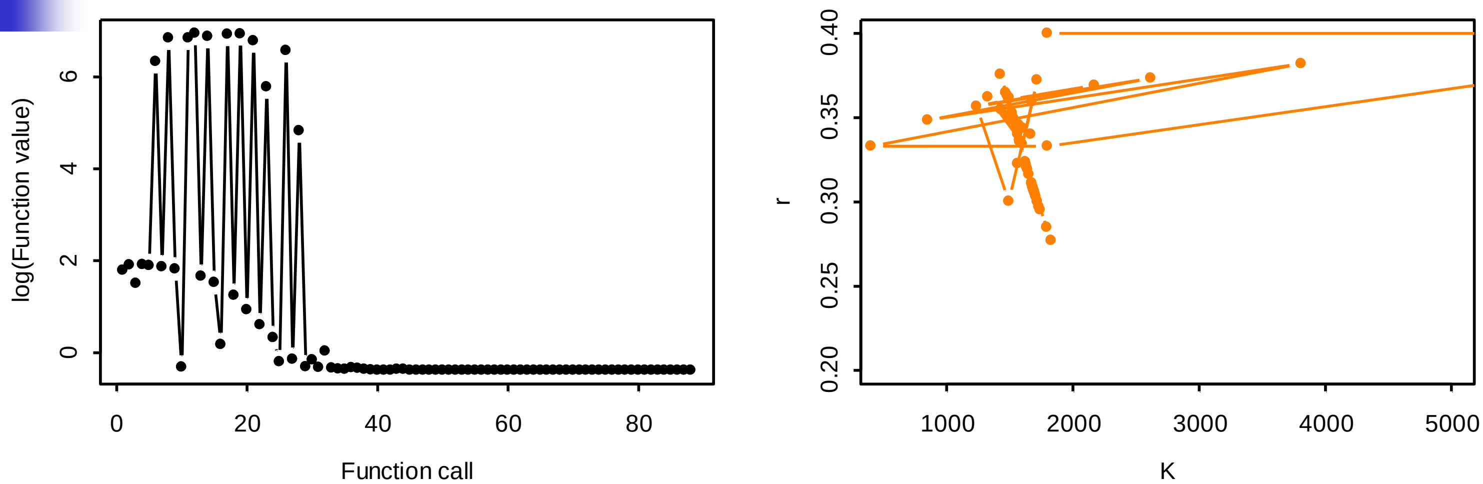




SS=0.6798
r=0.307
K=1692

468 function evaluations!
The algorithm requires some tuning!

Results - **Simplex**



SS=0.6798
r=0.307
K=1692

88 function evaluations
45 iterations

559

Comparison of Packages (from Schnute *et al.*, 1998)

Parameters	Product	Calls	Times
37	ADMB	161	4.6 s
	Gauss	4041	2.8 min
	MatLab	1936	5.8 min
	S-plus	n/a	n/a
100	ADMB	291	38 s
	Gauss	23,365	1.08 hr
	MatLab	18,360	3.25 hr
	S-plus	n/a	n/a

The trials involved fitting an age-structured model

559

Hints

- Methods may work better if the variables are all scaled to approximately 1.
- If some of the parameters are linear functions of the parameters (e.g. the q and σ parameters in a likelihood function for the Schaefer model), find analytical solutions for them – this reduces the dimension of the problem.
- Never trust a non-linear minimizer!
  - Try different starting values.
  - Combine different methods (e.g. apply Simplex and then a conjugate gradient method). The function optimx in the package optimx provides a wrapper for the algorithims in optim (except SANN) as well in other packages.
  - Test your code with deterministic data, then simulated data with stochastic processes.