Optimization and related nonlinear modelling computations in R

John C. Nash
Tefler School of Management
University of Ottawa
Canada
nashjc _at_ uottawa.ca

Materials: http://macnash.telfer.uottawa.ca/~nashjc/Nash_UseR2010/

What is possible in this session?

An overview of the (large, rapidly changing, yet incomplete) set of tools in R for optimization

An appreciation of the types of problems and types of methods to solve them

Some advice on setting up problems

Some suggestions for interpreting results

But ... unlikely to create instant experts

... and JN still has lots to learn too!

The formal problem(s)

Find $\boldsymbol{x}$ = argmin f($\boldsymbol{x}$) s.t. $\boldsymbol{q}(\boldsymbol{x})$ >= $\boldsymbol{0}$ (NLP)

- NOT very common (yet?) in statistics
- General version not prominent in this tutorial
- BUT variants / special cases of this problem are very important and common in statistics
  - Unconstrained problems or box-constrained ones
  - Special forms e.g., sums of squares (NLS)
- NOTE: We will “minimize” functions. Maximization is by min (-f($\boldsymbol{x}$))

Or ...

- Writing an objective function properly and checking it
- Scaling and constraints
- Starting values
- Understanding the output. Did I get a good answer?
- Common error messages and how to address them?
- Which package(s) should I use?

Caveats

- Some (most?) of you know
  - R better than I, or
  - more stats than I, or
  - more about some optimization tools than I
- Our problems shape our skills and views
- I like to “make things work”
  - s/w design and usability as much as application
  - Prefer clean, clear methods (but sometimes one needs complicated, messy ones)

The formal problem(s)

Find $\boldsymbol{x}$ = argmin f($\boldsymbol{x}$) s.t. $\boldsymbol{q}(\boldsymbol{x})$ >= $\boldsymbol{0}$ (NLP)

- NOT very common (yet?) in statistics
- General version not prominent in this tutorial
- BUT variants / special cases of this problem are very important and common in statistics
  - Unconstrained problems or box-constrained ones
  - Special forms e.g., sums of squares (NLS)
- NOTE: We will “minimize” functions. Maximization is by min (-f($\boldsymbol{x}$))

Conclusions

for those wanting to be first to the bar

- Most methods work most of the time
  - Generally quite easy to use, but setup of objective function often very user-specific
  - BUT ... all of them fail sometimes
  - And not so easy to swap methods
  - Needed “extra” computations may be missing and awkward to supply, especially “nicely”
  - AND ... everyone wants to make a better optimizer (but only for the problem at hand)
  - So there are (too!) many choices

Conclusions

for those wanting to be first to the bar

- Methods are iterative
  - Choice of start can be critical
    - To performance
    - To success – getting the “right” answer
  - Convergence vs. Termination
    - Algorithms converge
    - Programs terminate
    - Neither may be at the “right” solution
  - The “right” answer is usually the one the user knows is right

My Own View

- Optimization tools are extremely useful
- But take work and need a lot of caution
- R is the best framework I have found for exploring and using optimization tools
  - I prefer it to MATLAB, GAMS, etc.
  - No problem has yet proved impossible to approach in R, but much effort is needed
- Still plenty of room for improvement in R
  - Methods; Interfaces, Documentation; User Ed.

Rest of tutorial

- Try to provide some background to these conclusions
- Try to establish a better dialog with users to help improve R tools for optimization
- Try to develop the tutorial information so it can help the R community in general
    - Vignette collaboration invited
- Your problems welcome!

The formal problem (reminder)

Find $\boldsymbol{x}$ = argmin f($\boldsymbol{x}$) s.t. $\boldsymbol{q}(\boldsymbol{x})$ >= $\boldsymbol{0}$ (NLP)

Characterizations of problems

Number of constraints:

Lots: Math programming

Few: Function minimization (my main experience)

Number of solutions

none (no feasible solution due to constraints) 

one and only one 

several 

many 

plateaux and saddles 

Multiple real or "near" minima very frequently a source of difficulties ==> global optimization

Characterizations of problems (2)

By smoothness or reproducibility of function

By math / algorithmic approach to solution

- Descent method (gradient based)
- Newton approach (Hessian based)
- Direct search
- "derivative-free" methods may implicitly use gradient ideas

By statistical versus optimization viewpoint

Side-note: Deciding if we have a solution

- KKT conditions
- Other indicators

KKT conditions (local min.)

- Gradient is zero
  - Hilltop, valley bottom or saddle?
  - How small is zero?
- Hessian is positive-definite
  - Curvature is “upwards” ==> valley bottom
  - How do we decide?
- Issues are tolerances for small gradient and negative/zero eigenvalues of Hessian
  - Complications of scaling

Other indicators

- Grid search
  - e.g., cleversearch in svcm package
    - But see code – max [3D], references?
- Axial search (+ / - steps only)
  - Tilt and curvature
- Dispersion estimates. With constraints?
- Room for more work on good “indicators”

“Are we there yet?”

Global optima

- What users want
- What they (almost) never get!

- Mathematical conditions for global optimum rarely available, and computational implementations even less so, i.e., Lipshitz conditions using bounds on gradients

  e.g., http://ab-initio.mit.edu/wiki/index.php/NLopt_Algorithms

R view of optimization problems

Expressions (as in `nls`)

y ~ a1 / (1 + a2 * exp(- a3 * t)) [parameters a1,a2,a3]

Mainly least squares problems.

BUT: Not all sums of squares are from expressions

Functions (as in optim and descendents)

```
objfn <- function(x, ...) {
        (code)
        something <-.....
        return( something)
   }
```

Strategic issues in methods

Single algorithms, possibly safeguarded

- allows package developers to match to problems
- "user-developed" specialized polyalgorithms
- requires user knowledge for best use

All-purpose polyalgorithms ("always work")

- possibly approach of nlminb and nlm creators
- difficult to understand / debug
- may be best suited to experience/style of the creators

JN leans more to first style, esp. for R packages

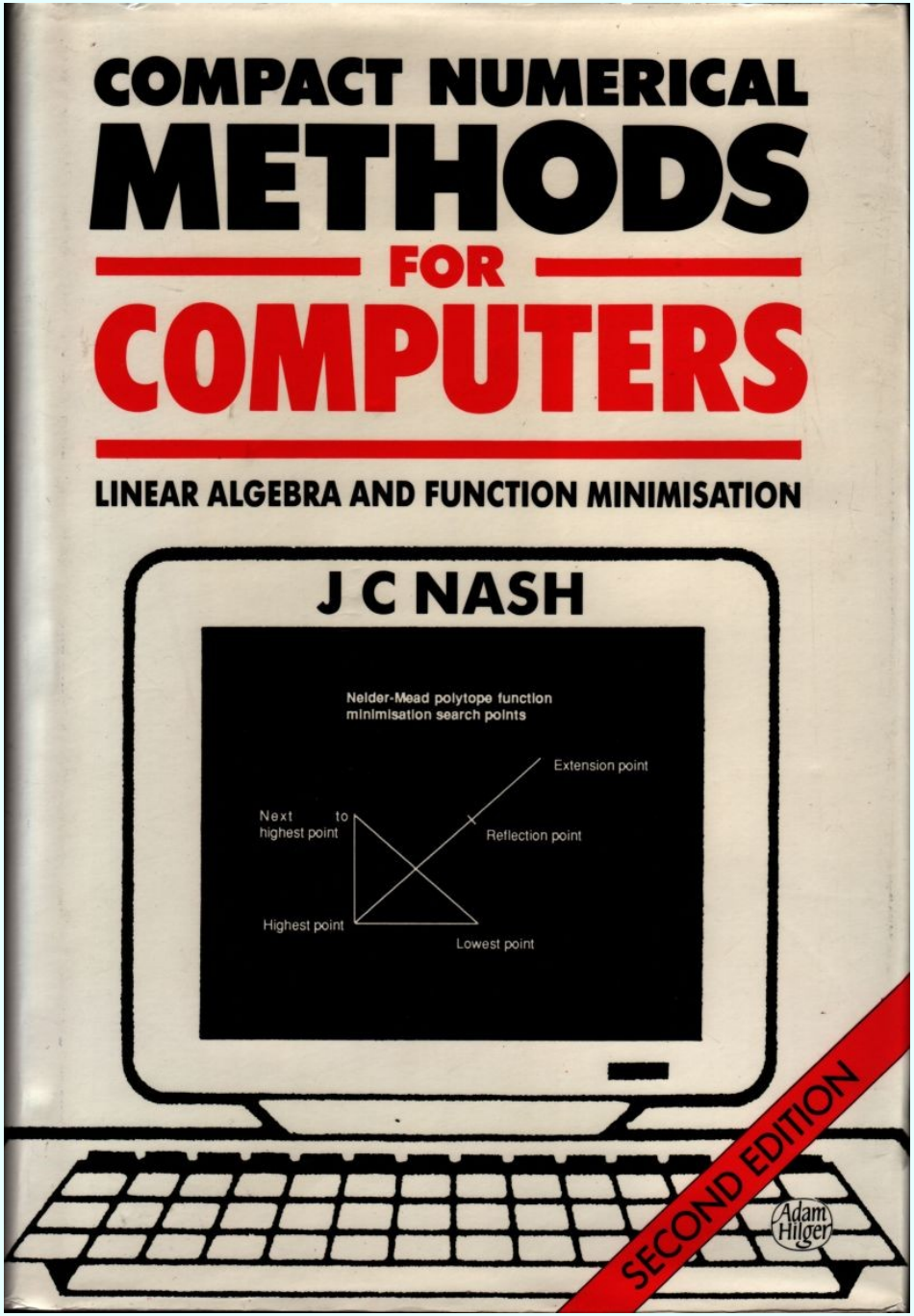
Tactical choices for R

- Interface existing codes in Fortran or C or ...
  - R <--> C (<--> Fortran) for each fn evaluation!!!
  - Difficult to debug
  - .Call, .Fortran, SEXP's etc.
  - Rcpp for C++ (another layer)
- Or all in R
  - Need to vectorize for best performance
  - Improvements to R interpreter/compiler?
  - Easier to understand and debug if all in R
  - Not necessarily so slow (but more tests needed)

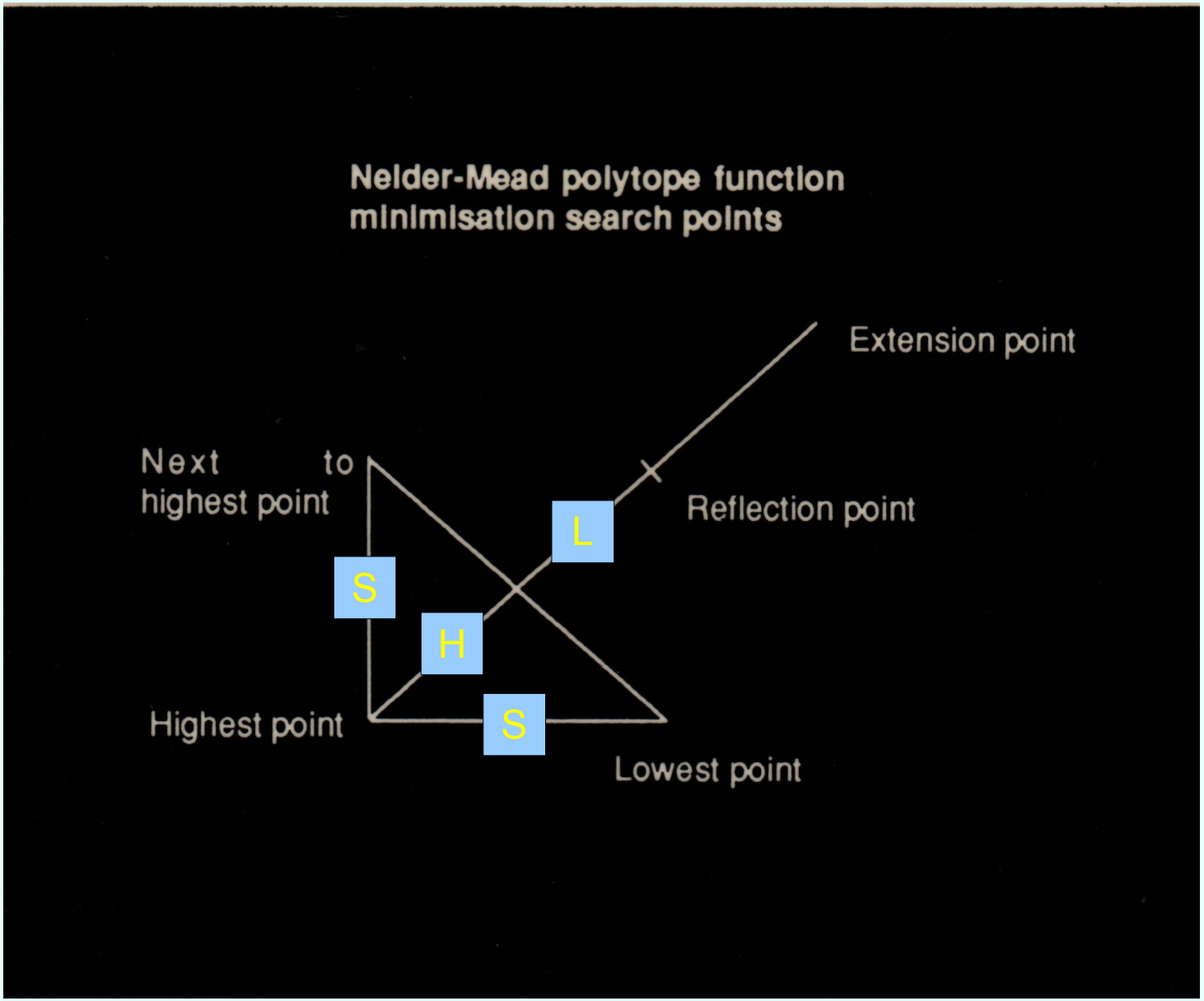
How methods work

- Heuristic
- Random
- Motivated by Newton
- Descent direction
- Approximating surface

- Mixture of above


COMPACT NUMERICAL
METHODS
FOR
COMPUTERS
LINEAR ALGEBRA AND FUNCTION MINIMISATION
J C NASH
Nelder-Mead polytope function
minimisation search points
Extension point
Next to
highest point
Reflection point
Highest point
Lowest point
SECOND EDITION
Adam
Hilger

Nelder Mead


Nelder-Mead polytope function
minimisation search points
Extension point
Next to
highest point
Reflection point
L
S
H
Highest point
S
Lowest point

“Random”

- Almost always have heuristics or local minimizers too
- 'SANN' – warning: NO convergence indicator
- DEOptim
- Lots of “simulated annealing”, “genetic algorithms”, “tabu search”, etc.
  - VERY hard to be sure where differences lie
  - Developers have faith, users have hope, and critics may have charity

Newton family

- Starting point ***x***
- Gradient ***g***, Hessian **H**, solve **H** ***delta*** = - ***g***
- ***x_new*** = ***x*** + ***delta*** . . . sort of ...
- Thousands of modifications
  - Line searches
  - Safeguards
  - Kitchen sinks – Gauss-Newton etc.

Quasi Newton / Variable Metric

- **H** is expensive to compute, so fake it
  - "approximate" with **1**, then improve at each iteration (thus start with steepest descents)
  - Updating formulae for **H** and $\mathbf{H}^{-1}$
    - 'BFGS' and 'L-BFGS-B' update inverse
    - Some methods update approx. **H**
- Lots of approaches to "line" search, trust region, "acceptable point", etc.

Descent direction

- Find a “downhill” direction e.g., ***-g***, step, repeat.
- Nice to have directions somehow “conjugate”
  - Gram-Schmidt like approaches (??)
  - Conjugate gradients
  - Lots of issues with “loss of conjugacy” and restarting
- Some overlap with Truncated-Newton methods

Approximating surface

- Sample surface, find approximating function
- Move to min of approximant and “repeat”
- Lots of choices in how to select points, what points to use in approximation, how to build the approximation, line searches, heuristics etc.
- Complicated by imprecise functions (RSMIN)
- Sometimes can be very efficient (Powell methods) but “touchy” to use

Common issue: when to stop

- Many methods would be much more efficient if we had a cheap and reliable “minimum” test.
- Nelder Mead can spend more than half the function evaluations deciding to stop
- Most methods do more work than needed because we cannot be sure we are finished

Some problems

- ***Hobbs*** – where I came in (1974) -- HV.Rnw
- Cobb Douglas – CD.Rnw
- Exponential fits: shifted and multiple
  - optex.R, expontest.Rnw, bvlstest.R
- Distribution fits – PoissLikJO.Rnw
- Large n problems
  - Rayleigh Quotient eigprob.Rnw
  - Artificial tests artificial.Rnw
- Non-smooth / imprecise

Looking at problems: an idealized agenda for each

- The obvious approach
- Can we do better?
- Discussion of the issues?
- Recommendations
  - Preferably avoiding “you ought to use ....”
  - Hopefully using methods that are familiar, at least in how they are used and what they return

Hobbs problem

- Weed infestation over 12 seasons
- Initial request for 3 parameter logistic

```
y ~ b1/(1 + b2 * exp(-b3 * t))
```
where

```
y<-c(5.308, 7.24, 9.638, 12.866, 17.069, 23.192,
 31.443, 38.558, 50.156, 62.948, 75.995, 91.972)
t<-1:12
```

- But `nls` often fails unless we have “good” starting parameters, while optimization methods mostly get “near” the solution -- HV vignette

Scaling

Want parameters x[i] all between 1 and 10

Zero parameters may not be “there”

Also give rise to scaling issues.

Part of the overall issue of ***reparametrization***

xnew **= *z(x)*** (***z*** vector valued, invertible)

Try simple case

xnew = Z ***x*** $\qquad\qquad$ ***x*** **=** Z^{-1} ***xnew***

Z is a simple non-singular diagonal matrix

Scaling in (cont.)

$f(\boldsymbol{x}, ...) = f(Z^{-1}\ \boldsymbol{xnew}, ...) = fnew(\boldsymbol{xnew}, ...)$

But we often end up doing the algebra, and it can be error prone, particularly for the derivatives.

$\partial fnew(\boldsymbol{xnew},...)/\partial xnew_i = \partial f(\boldsymbol{x}, ...)/\partial x_i * \partial x_i/\partial xnew_i = \partial f(\boldsymbol{x}, ...)/\partial x_i \ \ Z_{ii}^{-1}$

Hobbs: ***x***<-c(100,10,.1) ***xnew*** <- c(1,1,1)

Z^{-1} = diag(100, 10, .1)

g(***x***) = c(-100.9131, 783.5327, -82341.5897)

gnew(***xnew***) = c(-10091.312, 7835.327, -8234.159)

Why Bad Scaling "Hurts"

hobbs.r: 12 data points to be fitted to
$y \sim x_1/(1+x_2*\exp(-x_3*t))$ (3 parameter logistic)

Function base = 23520.58 at 1 1 1
Percent changes for 1 % change in each parameter are *0.03503 0.00020 0.00046*

Function base = 2.587542 at 196.5079544 49.1138533 0.3133611
Percent changes for 1 % change in each parameter are *94.117 39.695 391.27*
Hessian eigenvalues -- unscaled function
At start: 41.618914 16.635191 -3.700846 (INDEFINITE) Ratio -11.24579
At solution: 2.047414e+06 4.252238e-01 4.376540e-03 Ratio 467815596

Hence scale check in optimx().
JN should put it in funcheck() too (on r-forge).

“Simple” rescaling

$$y \sim 100\, x_1/(1+10\, x_2 * \exp(-0.1\, x_3 * t))$$

Function base = 23520.58 at [1] 0.01 0.10 10.00
Percent changes for 1 % change in each parameter are *0.03503 0.00020 0.00046*

Function base = 2.587543 at 1.965080 4.911385 3.133611
Percent changes for 1 % change in each parameter are *94.112 39.698 391.26*

No change. This is as it should be!

Hessian eigenvalues -- scaled function
At start: 223294.0 .5599862 -204.9109 (INDEFINITE) Ratio 398749.1
At solution: 33859.37019 76.55200 14.70142 Ratio 2303.137

Reparametrization

- D Bates version

```
y ~ c1/(1 + exp((c2 - t) / c3) )
```

  - Still needs care in starting
  - Has parameters that can be interpreted
    - Asymptote is c1
    - Time t at midpoint is c2
    - Sharpness of “stepup” inversely related to c3 Self-starting “models” extremely useful
  - Sometimes use linear approximations
  - Note selfStart – and work to create such tools

Estimating start

- Guess asymptote; scale: yy = y/(1.05*max(y))
- Linearize: z = log(yy/(1-yy)) ~ (t – c2) / c3
- Get c2, c3 from lm(t ~ z)
- Use nls(..., algorithm=”plinear”) to get c1 and refine c2, c3

Summary of results: original model

`y ~ b1/(1 + b2 * exp(-b3 * t))`

- Unscaled: 196.1862582 49.0916390 0.3135697

  S.E.s 11.307 1.6884 0.0068633

  by Hess 8.0230 1.1983 0.0048600
- Scaled: 1.961863 4.909164 3.135697

  S.E.s 0.11307 0.16884 0.068633

  by Hess 0.080230 0.11983 0.048600

S.E. Estimates: sqrt(RSS * diag(solve(t(J) %*% J))/(n-npar))

By Hessian: sqrt(RSS * diag(solve(Hessian)) / (n – npar))

J (jacobian) and Hessian evaluated at parameter estimates.

Summary of results: Bates model

Formula: yy ~ 100 * c1/(1 + 10 * exp(0.1 * (c2 - tt)/c3))

```
   Estimate Std. Error t value Pr(>|t|)
c1  1.96186    0.11307   17.35 3.17e-08
c2  5.07416    0.18714   27.11 6.11e-10
c3  0.31891    0.00698   45.69 5.77e-12
```

Formula: yy ~ c1/(1 + exp((c2 - tt)/c3))

```
c1 196.1863    11.3069   17.35 3.17e-08
c2  12.4173     0.3346   37.11 3.72e-11
c3   3.1891     0.0698   45.69 5.77e-12
RSS for all models ~= 2.587
```

Hobbs: lessons

- Because “raw” problem has near singularities in Hessian, NM may succeed when Newton fails, and provide starting values
- Good starting values – nls() needs them
- Scaling helps
- Reparametrization helps more, but users may want the original model
- How to get dispersion estimates for model parameters?

Why things go wrong

- Objective function set up badly
  - Just plain wrong – mistakes in design or coding
  - Poorly scaled
  - Overparametrized
  - No control of inadmissible inputs

    (...)/0; log(0) or log(negative); sqrt(negative)

    exp(big) or x ^ big

    x > 2 and x<1 style infeasibilities

I am currently trying to solve a Maximum Likelihood optimization problem in R. Below you can find the output from R, when I use the "BFGS" method. The problem is that the parameters that I get are very unreasonable, ...

(some code)

(response)

Two possible problems:

(a) If you're working with a normal likelihood---and it seems that you are---the exponent should be squared.

(b) lag may not be working like you think it should. Consider this silly example ...

Why things go wrong - 2

- "Solutions" to math, not to real-world problem
  - Try to build in "admissibility", but that is difficult!
- Programs, including R packages, have too many control settings for even a small subset of possibilities to have been tested
  - "Tests" are only an infinitesimal subsample of the possible domain
- Some problems are ill-posed (e.g. Hassan18.2)

There is a contradiction between what the help page says and what constrOptim actually does with the constraints. The issue is what happens on the boundary.
The help page says
The feasible region is defined by ?ui %% theta - ci >= 0?,*
but the R code for constrOptim reads

```
if (any(ui %*% theta - ci <= 0))
    stop("initial value not feasible")
```

Why things go wrong - 3

- Gradients mis-specified (if at all)
  - /home/john/R-optimtest/2010tutorial/Rhelp-gradient-091130.txt
  - Sometimes we DO need analytic derivatives
- “Bad” starting values
  - genrosestart100708.R
  - Infeasible start
- Bad control settings – check iteration limits
  - Sometimes we're “almost” there, but ...
  - Different controls in different methods
  - optimx() tries to unify, but ...

I am doing a optimization problem using nlminb. It seems to me that the result is kind of sensitive to the starting value.

I have constructed the function mml2 (below) based on the likelihood function described in the minimal latex I have pasted below for anyone who wants to look at it. This function finds parameter estimates for a basic Rasch (IRT) model. Using the function without the gradient, using either nlminb or optim returns the correct parameter estimates and, in the case of optim, the correct standard errors.

By correct, I mean they match another software program as well as the rasch() function in the ltm package.

Your function named 'gradient' is not the correct gradient.

Annoyances

- Structuring of the problem input/output
  - How functions / expressions must be provided
  - Names / availability of outputs not consistent
  - Attributes vs. Regular returned values
- Getting at the ancillary information “easily”
- Finding information about the methods and approaches e.g., How are SE's computed?
- Everything a little more difficult than we like!
- Options, Options, Options! -- WHY?

Complaint!

Tutorial proposal suggested covering

- "Common error messages and how to address them"

- Not easy to do
- Especially when code is ***not*** in R
- Do any .Rd files include a list of error messages?
  - A mirror would show me one culprit.

Cobb-Douglas models

- Model of production as function of labour (L) and Capital(K)
  - $Y \sim beta1 * L^{beta2} * K^{beta3}$
- Issue: What should be the loss function

$$Y = beta1 * L^{beta2} * K^{beta3} + add\_error$$

or

$$\log Y = \log(beta1) + beta2 * \log(L) + beta3 * \log(K) + mult\_error$$

- "Standard errors"

Cobb-Douglas Examples

Data from Pedro Arroyo – coef from log model has different sign from unlogged one

Data from http://www.sts.uzh.ch/data/cobb.html and André Oliviera very similar

All show parameters with high estimated dispersion

3D graphs may be helpful to see sparsity of data

- But don't give much help with estimation

Partial linearity: sum of exponentials

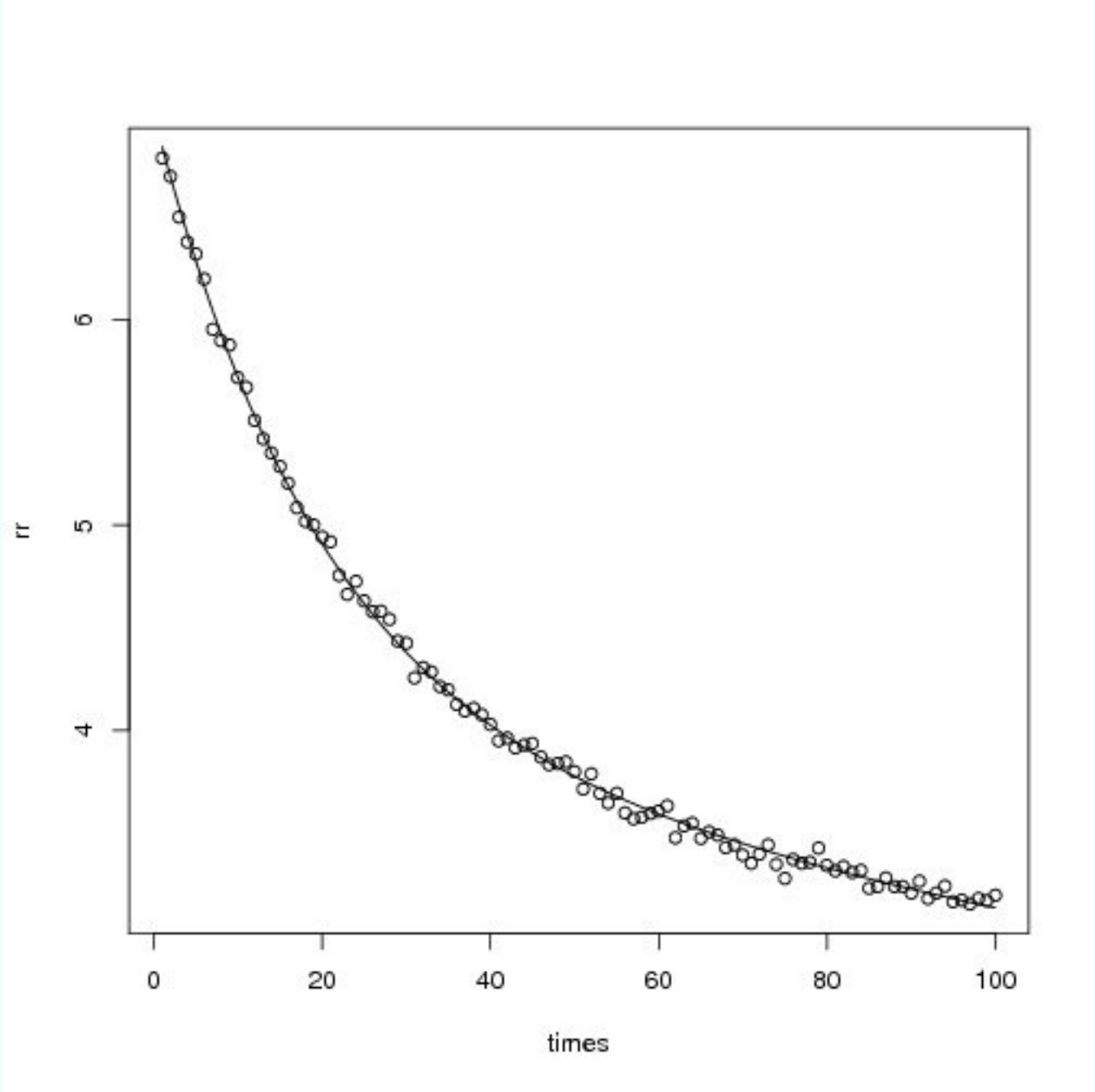
- Sum of exponentials ALWAYS difficult
  - Lanczos (1956) shows why;
    - other refs in Nash and Walker-Smith, 1987, http://macnash.telfer.uottawa.ca/nlpe/
  - Is this a realistic problem? e.g., chemical kinetics
- NIST Lanczos problems – useful tests or not?
- Plain approach to optimization does very poorly
  - Need to have VERY close starting values
  - Pays to use the partial linearity (gradient?)

Example – simplified Lanczos

Model: y ~ b3 * exp(-b1 * t) + b4 * exp(-b2 * t)

- put parameters = c(0.05, 0.0025, 3, 4)
- Generate t = 1:100 and calculate model values
- Input sd (default 0.04) and set.seed(918273645)
- Generate rnorm(100) and add to model values
- This is similar to competing chemical reactions data.

optex.R example script


rr
6
5
4
0
20
40
60
80
100
times

Approaches

nls(rr ~ I1 * exp(-e1*times) + I2 * exp(-e2*times), ...

This does better than optim( ) i.e., Nelder Mead.

Even better is

nls(rr ~ cbind(exp(-e1 * times), exp(-e2 * times)),, data = mydata, algorithm='plinear', ...

BUT zero residual problems cause nls() difficulty

nls.lm from minpack.lm may do somewhat better

And optim() does “not too badly” when only nonlinear parameters are varied, i.e., solve for linear model as per 'plinear'.

Problem gets increasingly difficult with number of exponentials

Methods may get ordering of parameters swapped.

Opinions

- Avoid multiple exponentials problem if you can!
  - Guaranteed multiple minima
  - Inherently ill-conditioned – small changes in inputs --> big changes in outputs (parameters)
- Use the linearity
  - Note that there may be efficiencies in NOT fully solving the problem. (Q' * y)[n+1:m] gives the residual sum of squares from qr()
- Be cautious about problems of this type.
  - How to interpret the linear parameters?

Shifted Exponential - Lievens

Model: y ~ y0 + alpha * E^t

Data:

```
y = c(2018.34, 2012.54, 2018.85, 2023.52, 2054.58, 2132.61, 2247.17, 2468.32, 2778.47)
t = c(17, 18, 19, 20, 21, 22, 23, 24, 25)
```

Note expontest.Rnw (Sweave file)

- Naive attempts fail “singular gradient”
- Can see approx y0 from graph and use linear model to get starting values.
- DEoptim for finding starting values
- qpcR package – big learning curve (>90 pages)

Distribution fitting

- Tools:
  - MASS::fitdistr, stats4::mle, bbmle::mle2
  - Marie Laure Delignette-Muller: fitdistrplus to extend fitdistr with some extra tools
  - http://openmx.psyc.virginia.edu/ OpenMX
    - Meant for Structural Equation modeling
  - Can sometimes use glm()
- Biggest issue is learning cost & getting the setup correct

Examples

- See fitdistr, mle, and fitdistrplus
- Jens Oehlschlägel problem (Poisson glm)
  - Shows Powell's bobyqa quite useful here
    - But other examples give trouble e.g. bvlstest.R
  - PoissLikJO.Rnw vignette

Large n problems

- Statistical problems tend to be complicated, with difficult code
- Math Programming problems have rather different structure and focus on constraints
- Here will use the eigenvalue problem and some artificial test problems
    - Large as we want
    - Illustrative of the issues
    - Easier to explain the problems and provide tests

Rayleigh Quotient Minimization

- Given matrix A, find eigensolution with most positive or most negative eigenvalue by optimizing Rayleigh quotient.
  - RQ(u) <- t(u) %*% A %*% u / (t(u) %*% u)
- Do not need A, though in our tests it will be "around"
  - Should have routine that forms v <- A %*% u implicitly

Rayleigh Quotient minimization

- For symmetric matrix A of dimension n, find the vector **x** that minimizes

  $$Q = (\mathbf{x}' A \mathbf{x}) / (\mathbf{x}' \mathbf{x})$$

  Subject to some constraint on the size of **x**,
- Typically constrain $\mathbf{x}' \mathbf{x} = 1$
- Vignette – eigprob.Rnw
  - Need to specialize the optimization to get good results

Artificial tests

- Almost always sums of squares
- Sometimes hard to find “real” version (broydt.R and genrose.R) – several variants
- But relatively easy to set up and use, including gradients and other derivatives

artificial.Rnw (incomplete)

Non-smooth / imprecise

- Functions can be non-smooth, i.e., the function or gradient is non-continuous
- Imprecise functions cannot be evaluated exactly e.g., Schumacher time to lap as function of racing car settings
- We tend to use similar – and largely stochastic and heuristic – methods for both classes of problems, but should really differentiate between approaches.

H Joe problems

Maximum likelihood type problems where the objective function can be thought of as a multi-dimensional integral approximately computed by Monte Carlo techniques

- JN is NOT familiar with the real-world problems
- Harry and I spent > 10 years developing RSMIN which worked rather well, but “nasty” to use

  Joe & Nash, Statistics & Computing, 13, 277-286, 2005
- Hope: optimizing imprecise function faster than traditional method on accurate function
- NOT in R; need interested users

Handling constraints

- Bounds – tools available that are relatively easy to use
  - Masks – fixed parameters – Rcgmin & Rvmmin
- Equality Constraints – can be tricky, as really want to solve for parameters if possible
- Inequality constraints
  - linear inequalities – ConstrOptim
  - Projection method -- spg from BB
  - Penalty and Barrier functions – user coded?

Examples of constraints

- Nonlinear equality constraint – hassan182.R
  - Linear model with constraint on parameters
  - Cannot replicate my own work from ~ 1977 possibly due to typo in data table
  - nls.lm from minpack.lm seems best tool
    - Similar result in 1970s (Marquardt best)
    - Eliminate 1 parameter by solving in constraint
    - Penalty fn method “works” moderately well
    - Parameters ill-conditioned in problem

Linear inequality constraints

- Could use math programming tools, especially if many constraints.
- Penalty or barrier methods when just a few

  See Dixon72.R
- Other examples?

Issues raised by constraints

- How should we interpret measures of dispersion
  - Beta > 0. Does this mean interval ends at 0?
  - How to define & compute dispersion measures
- Setup for constraints is generally non-trivial
- Infeasibility? How do we know?
- Introduced ill-conditioning from constraints
- Disjoint parameter regions
- Inconsistent handling of fn <- Inf or NA

DANGER!

Advice about to be given.

Objective function setup

- Keep it as simple as possible
- Scale if possible – poor scaling creates trouble
- Check, check, check
  - Build in checks 'debug<-TRUE' etc.
  - R debug tools?
- Graphs where they make sense

Can we eliminate many “extra” minima? Other “bad” situations?

  - Most issues require attention to ***details***

Gradients

- Important
  - Sometimes better solutions
  - Speedup, esp. large-n problems
- Using deriv or D is helpful but not trivial
- Check with `numDeriv()`
- Automatic Differentiation -- work in progress
  - ADMB approach fairly well-developed
  - General tools “under construction” -- rdax
- BUT ... lots of work

Starting values

- Use of linearizing approximations
- Use of “last” values for repetitive estimations
- DEOptim(), optim/SANN
- Random starts
- Use of bounds (and midpoints; random in [a,b])
  - Often don't want to be on the bounds
  - May need “local” knowledge of problem
  - Force user to think about problem

Control settings

- Set iterations > 50 for `nls` via

```
control=list(maxit = 500,trace=TRUE)
```

- A serious issue for different optimization tools is that the controls are different
  - One reason for `optimx()`
- Some methods have more controls than others
  - Often not well-documented; examine code (!?)
- Package defaults may not suit your problem

Subject specific packages

Polymerase chain reaction models – qpcR

Analysis of dose-response curves – drc

Others ....

- Great if you are doing “same” work
- Not so good if your setup a bit different
- Bad if you don't know the subject
- In any event – lots to learn, so time cost

Special Methods Packages

Nonlinear mixed effects models – nlme (+ gnls)

Ben Bolkers maxlik package – bbmle (+ mle2)

Bates et al – lme4

Others?

- May offer useful tools and examples
- BUT ... things we want may be missing
- Focus is always on the developer's needs

Automatic starting values

- SelfStart ideas
- Useful if there is a model already worked out that you need
- Otherwise you have work to do
- Handling exceptions takes most of the work

Reverse communication

- Attempt to avoid passing “large” structures to subroutines
- MESSY!
- But does simplify setup in some ways, while creating spaghetti in another
- Main routine gets “return” from optimizer with “instruction” -- usually an integer
  - Does work and calls optimizer again
  - Loop until “instruction” is to stop

Finding Help

- CRAN Task View on Optimization (S Theussl)
- Rhelp – including archives (? how ?)
    - http://finzi.psych.upenn.edu/search.html
- Rseek – but does it work?
- Rwiki – Should use it more!
- Nash optimx wiki – for bleeding edge ideas of both users and developers
    - http://macnash.telfer.uottawa.ca/optimx/

Future directions and needs

- USERS!
  - Trying things out & organizing tests
  - Helping with documentation
  - Complaining constructively
- Developers
  - Integration of methods and tools
  - Better interfaces
- “Educators”
  - To help organize our understandings