Optimizers, Hessians, and Other Dangers

Benjamin S. Skrainka
Harris School of Public Policy
University of Chicago

July 16, 2012

Overview

We focus on how to get the most out of your optimizer(s):

1. Scaling
2. Initial Guess
3. Solver Options
4. Gradients & Hessians
5. Dangers with Hessians
6. Verification
7. Diagnosing Problems
8. Ipopt
9. Floating Point Issues
10. The Evils of the Logit

Scaling

Scaling can help solve convergence problems and improve numerical stability:

- Naive scaling: scale variables so their magnitudes are ~ 1
- Better: scale variables so solution has magnitude ~ 1
- For dynamic problems, choose a sensible time step
- A good solver may automatically scale the problem
- Goal: make problem equally sensitive to steps along any direction

Orthogonal Polynomials

Many researchers use simple powers of covariates $1, x, x^2, \ldots$

- Simple powers are highly collinear $\Rightarrow$
  - Huge condition number
  - Numerical problems
  - Ill-conditioned Vandermonde matrix
- Use orthogonal polynomials
- This has been known for a long time:
  - D.J. Hudson (1964) "Statistics lectures II: maximum likelihood and least squares theory"
  - Bradley & Srivastava (1979) "Correlation in Polynomial Regression"
  - Newman (1981) "Matrix mutual orthogonality and parameter independence"

Computing an Initial Guess

Computing a good initial guess is crucial:

- To avoid bad regions in parameter space
- To facilitate convergence
- To satisfy constraints
- Possible methods:
  - Use a simpler but consistent estimator such as OLS
  - Estimate a restricted version of the problem: e.g., use logit + 2SLS when estimating a mixed logit/BLP
  - Use Nelder-Mead or other derivative-free method (beware of `fminsearch`)
  - Use pseudo-Monte Carlo, quasi-Monte Carlo, or a 'voodoo' method (Simulated Annealing, Genetic Algorithm)
- Beware: the optimizer may only find a local max!

Solver Options

A state of the art optimizer such as knitro is highly tunable:

- You should configure the options to suit your problem: scale, linear or non-linear, concavity, constraints, etc.
- Experimentation is required:
  - Algorithm: Interior/CG, Interior/Direct, Active Set
  - Barrier parameters: `bar_murule`, `bar_feasible`
  - Tolerances: X, function, constraints
  - Diagnostics
- See Nocedal & Wright for the gory details of how optimizers work

Which Algorithm?

Different algorithms work better on different problems:

Interior/CG

- Direct step is poor quality
- There is negative curvature
- Large or dense Hessian

Interior/Direct

- Ill-conditioned Hessian of Lagrangian
- Large or dense Hessian
- Dependent or degenerate constraints

Active Set

- Small and medium scale problems
- You can choose a (good) initial guess

The default is that knitro chooses the algorithm.
$\Rightarrow$ There are no hard rules. You must experiment!!!

Knitro Configuration

Knitro is highly configurable:

- Set options via:
  - C, C++, FORTRAN, or Java API
  - MATLAB options file
- Documentation in ${KNITRO_DIR}/doc/html
- Example options file in ${KNITRO_DIR}/examples/Matlab/knitro.opt

Calling Knitro From MATLAB

To call Knitro from MATLAB:

1. Follow steps in InstallGuide.pdf I sent out
2. Call `ktrlink`:

```
% Call Knitro
[ xOpt, fval, exitflag, output, lambda ] = ktrlink( ...
  @(xFree) myLogLikelihood( xFree, myData ),          ...
  xFree, [], [], [], [], lb, ub, [], [], 'knitro.opt' )
% Check exit flag
if exitflag <= -100 | exitflag >= -199
  % Success
end
```

- Note: older versions of Knitro modify `fmincon` to call `ktrlink`
- Best to pass options via a file such as '`knitro.opt`'

Listing 1: knitro.opt Options File

```
# KNITRO 8.0.0 Options file
# http://ziena.com/documentation.html

# Which algorithm to use.
#   auto   = 0 = let KNITRO choose the algorithm
#   direct = 1 = use Interior (barrier) Direct algorithm
#   cg     = 2 = use Interior (barrier) CG algorithm
#   active = 3 = use Active Set algorithm
#   multi  = 5 = run multiple algorithms (perhaps in parallel)
algorithm     auto

# When using the Interior/Direct algorithm, this parameter
# controls the maximum number of consecutive CG steps before
# trying to force the algorithm to take a direct step again.
# (only used for alg=1).
bar_directinterval  10

# Whether feasibility is given special emphasis.
#   no       = 0 = no emphasis on feasibility
#   stay     = 1 = iterates must honor inequalities
#   get      = 2 = emphasize first getting feasible before optimiz
#   get_stay = 3 = implement both options 1 and 2 above
bar_feasible no
```

```
# Specifies the tolerance for entering the stay feasible mode
# (only valid when bar_feasible = stay or bar_feasible = get_stay)
bar_feasmodetol  0.0001

# Initial value for the barrier parameter.
bar_initmu   0.1

# Whether to use the initial point strategy with barrier algorithm
#   auto = 0 = let KNITRO choose the strategy
#   yes  = 1 = shift the initial point to improve barrier performa
#   no   = 2 = do not alter the initial point supplied by the user
bar_initpt   auto

# Maximum allowable number of backtracks during the linesearch of
# Interior Direct algorithm before reverting to a CG step.
# (only used for alg=1).
bar_maxbacktrack  3

# Maximum number of crossover iterations to allow for barrier algo
bar_maxcrossit   0

# Maximum number of refactorizations of the KKT system per iterati
# Interior Direct algorithm before reverting to a CG step.
# (only used for alg=1).
bar_maxrefactor  0
```

```
# Which barrier parameter update strategy.
#   auto      = 0 = let KNITRO choose the strategy
#   monotone = 1
#   adaptive = 2
#   probing   = 3
#   dampmpc   = 4
#   fullmpc   = 5
#   quality   = 6
bar_murule    auto

# Whether or not to penalize constraints in the barrier algorithms
#   auto      = 0 = let KNITRO choose the strategy
#   none      = 1 = Do not apply penalty approach to any constraint
#   all       = 2 = Apply a penalty approach to all general constra
bar_penaltycons    auto

# Which penalty parameter update strategy for barrier algorithms.
#   auto      = 0 = let KNITRO choose the strategy
#   single    = 1 = use single penalty parameter approach
#   flex      = 2 = use more tolerant flexible strategy
bar_penaltyrule    auto

# Switching rule strategy for barrier algorithms that controls# sw
#   auto      = 0 = let KNITRO choose the strategy
#   never     = 1 = never switch
#   level1    = 2 = allow moderate switching
#   level2    = 3 = more agressive switching
bar_switchrule     auto
```

```
# Which BLAS/LAPACK library to use.  Intel MKL library is only ava
# on some platforms; see the User Manual for details.
#   knitro  = 0 = use KNITRO version of netlib functions
#   intel   = 1 = use Intel MKL functions
#   dynamic = 2 = use dynamic library of functions
blasoption    intel

# Specifies debugging level of output.  Debugging output is intend
# developers.  Debugging mode may impact performance and is NOT re
# for production operation.
#   none      = 0 = no extra debugging
#   problem   = 1 = help debug solution of the problem
#   execution = 2 = help debug execution of the solver
debug         none

# Initial trust region radius scaling factor, used to determine
# the initial trust region size.
delta         1.0

# Specifies the final relative stopping tolerance for the feasibil
# error. Smaller values of feastol result in a higher degree of ac
# in the solution with respect to feasibility.
feastol       1e-06
```

```
# Specifies the final absolute stopping tolerance for the feasibil
# Smaller values of feastol_abs result in a higher degree of accur
# solution with respect to feasibility.
feastol_abs  0.0

# How to compute/approximate the gradient of the objective
# and constraint functions.
#   exact        = 1 = user supplies exact first derivatives
#   forward      = 2 = gradients computed by forward finite differ
#   central      = 3 = gradients computed by central finite differ
gradopt      exact

# How to compute/approximate the Hessian of the Lagrangian.
#   exact        = 1 = user supplies exact second derivatives
#   bfgs         = 2 = KNITRO computes a dense quasi-Newton BFGS H
#   sr1          = 3 = KNITRO computes a dense quasi-Newton SR1 He
#   finite_diff  = 4 = KNITRO computes Hessian-vector products by
#   product      = 5 = user supplies exact Hessian-vector products
#   lbfgs        = 6 = KNITRO computes a limited-memory quasi-Newt
hessopt      exact

# Whether to allow computing Hessian of the Lagrangian without obj
#   forbid       = 0 = not allowed
#   allow        = 1 = user can provide this version of the Hessia
hessian_no_f forbid
```

```
# Whether to enforce satisfaction of simple bounds at all iteratio
#   no       = 0 = allow iterations to violate the bounds
#   always   = 1 = enforce bounds satisfaction of all iterates
#   initpt   = 2 = enforce bounds satisfaction of initial point
honorbnds     initpt

# Specifies relative stopping tolerance used to declare infeasibil
infeastol     1e-08

# Which linear system solver to use.
#   auto     = 0 = let KNITRO choose the solver
#   internal = 1 = use internal solver provided with KNITRO
#                  (not currently active; reserved for future use)
#   hybrid   = 2 = use a mixture of linear solvers depending on th
#   qr       = 3 = use dense QR solver always (only for small prob
#   ma27     = 4 = use sparse HSL solver ma27 always
#   ma57     = 5 = use sparse HSL solver ma57 always
linsolver     auto

# Number of limited memory pairs to store when Hessian choice is l
lmsize        10

# Which LP solver to use in the Active algorithm.
#   internal = 1 = use internal LP solver
#   cplex    = 2 = CPLEX (if user has a valid license)
lpsolver      internal
```

```
# Maximum allowable CPU time in seconds for the complete multi alg
# solution when 'alg=multi'.  Use maxtime_cpu to additionally limi
# spent per each algorithm.
ma_maxtime_cpu  1e+08

# Maximum allowable real time in seconds for the complete multi al
# solution when 'alg=multi'.  Use maxtime_real to additionally lim
# spent per each algorithm.
ma_maxtime_real 1e+08

# Specifies multi algorithm subproblem solve output control.
#   0 = no output from subproblem solves
#   1 = Subproblem output enabled, controlled by option 'outlev'.
#       Output is directed to a file 'knitro_ma_*.log'
ma_outsub 0

# Specifies conditions for terminating when 'algorithm=multi'.
#   all       = 0 = terminate after all algorithms complete
#   optimal   = 1 = terminate at first local optimum
#   feasible  = 2 = terminate at first feasible solution estimate
ma_terminate optimal

# Maximum allowable CG iterations per trial step
# (if 0 then KNITRO determines the best value).
maxcgit       0
```

```
# Maximum number of iterations to allow
# (if 0 then KNITRO determines the best value).
# Default values are 10000 for NLP and 3000 for MIP.
maxit          0

# Maximum allowable CPU time in seconds one algorithm solve.
# If multistart, multi algorithm or MIP is active, this limits tim
# on just one subproblem solve.
maxtime_cpu  1e+08

# Maximum allowable real time in seconds for one algorithm solve.
# If multistart, multi algorithm or MIP is active, this limits tim
# on just one subproblem solve.
maxtime_real 1e+08

# Specifies the MIP branching rule for choosing a variable.
#   auto        = 0 = let KNITRO choose the rule
#   most_frac   = 1 = most fractional (most infeasible) variable
#   pseudocost  = 2 = use pseudo-cost value
#   strong      = 3 = use strong branching
mip_branchrule auto

# Specifies debugging level for MIP solution.
#   none = 0 = no MIP debugging info
#   all  = 1 = write debugging to the file kdbg_mip.log
mip_debug none
```

```
# Whether to branch on generalized upper bounds (GUBs).
#   no    = 0 = do not branch on GUBs
#   yes   = 1 = branch on GUBs
mip_gub_branch no

# Specifies which MIP heuristic search approach to apply
# to try to find an initial integer feasible point.
#   auto      = 0 = let KNITRO choose the heuristic
#   none      = 1 = no heuristic search applied
#   feaspump = 2 = apply feasibility pump heuristic
#   mpec      = 3 = apply MPEC heuristic
mip_heuristic auto

# Maximum number of iterations to allow for MIP heuristic.
mip_heuristic_maxit 100

# Whether to add logical implications deduced from
# branching decisions at a MIP node.
#   no    = 0 = do not add logical implications
#   yes   = 1 = add logical implications
mip_implications yes

# Threshold for deciding if a variable value is integral.
mip_integer_tol 1e-08

# Specifies absolute stop tolerance for sufficiently small integra
mip_integral_gap_abs 1e-06
```

```
# Specifies relative stop tolerance for sufficiently small integra
mip_integral_gap_rel 1e-06

# Specifies rules for adding MIP knapsack cuts.
#   none       = 0 = do not add knapsack cuts
#   ineqs      = 1 = add cuts derived from inequalities
#   ineqs_eqs = 2 = add cuts derived from inequalities and equalit
mip_knapsack ineqs

# Specifies which algorithm to use for LP subproblem solves in MIP
# (same options as algorithm option).
mip_lpalg auto

# Maximum number of nodes explored (0 means no limit).
mip_maxnodes 100000

# Maximum number of subproblem solves allowed (0 means no limit).
mip_maxsolves 200000

# Maximum allowable CPU time in seconds for the complete MIP solut
# Use maxtime_cpu to additionally limit time spent per subproblem
mip_maxtime_cpu 1e+08

# Maximum allowable real time in seconds for the complete MIP solu
# Use maxtime_real to additionally limit time spent per subproblem
mip_maxtime_real 1e+08
```

```
# Which MIP method to use.
#   auto = 0 = let KNITRO choose the method
#   BB   = 1 = standard branch and bound
#   HQG  = 2 = hybrid Quesada-Grossman
mip_method auto

# Specifies printing interval for mip_outlevel.
#   1 = print every node
#   2 = print every 2nd node
#   N = print every Nth node
mip_outinterval 10

# How much MIP information to print.
#   none  = 0 = nothing
#   iters = 1 = one line for every node
mip_outlevel iters

# Specifies MIP subproblem solve output control.
#   0 = no output from subproblem solves
#   1 = Subproblem output enabled, controlled by option 'outlev'
#   2 = Subproblem output enabled and print problem characteristic
mip_outsub 0

# How to initialize pseudo-costs.
#   auto   = 0 = let KNITRO choose the method
#   ave    = 1 = use average value
#   strong = 2 = use strong branching
mip_pseudoinit auto
```

```
# Specifies which algorithm to use for the root node solve in MIP
# (same options as algorithm option).
mip_rootalg auto

# Specifies the MIP rounding rule to apply.
#   auto           = 0 = let KNITRO choose the rule
#   none           = 1 = do not round if a node is infeasible
#   heur_only      = 2 = round using heuristic only (fast)
#   nlp_sometimes = 3 = round and solve NLP if likely to succeed
#   nlp_always     = 4 = always round and solve NLP
mip_rounding auto

# Specifies the MIP select rule for choosing a node.
#   auto         = 0 = let KNITRO choose the rule
#   depth_first = 1 = search the tree depth first
#   best_bound  = 2 = node with the best relaxation bound
#   combo_1      = 3 = depth first unless pruned, then best bound
mip_selectrule auto

# Maximum number of candidates to explore for MIP strong branching
mip_strong_candlim 10

# Maximum number of levels on which to perform MIP strong branchin
mip_strong_level 10

# Maximum number of iterations to allow for MIP strong branching s
mip_strong_maxit 1000
```

```
# Specifies conditions for terminating the MIP algorithm.
#   optimal   = 0 = terminate at optimum
#   feasible  = 1 = terminate at first integer feasible point
mip_terminate optimal

# Whether to enable multistart to find a better local minimum.
#   no    = 0 = KNITRO solves from a single initial point
#   yes   = 1 = KNITRO solves using multiple start points
ms_enable     no

# Specifies the maximum range that an unbounded variable can vary
# multistart computes new start points.
ms_maxbndrange 1000

# How many KNITRO solutions to compute if multistart is enabled.
#   choose any positive integer, or
#   0 = KNITRO sets it to min{200,10*n}
ms_maxsolves 0

# Maximum allowable CPU time in seconds for the complete multistar
# solution.  Use maxtime_cpu to additionally limit time spent per
ms_maxtime_cpu  1e+08

# Maximum allowable real time in seconds for the complete multista
# solution.  Use maxtime_real to additionally limit time spent per
ms_maxtime_real 1e+08
```

```
# How many feasible multistart points to save in file knitro_mspo
#   choose any positive integer, or
#   0 = save none
ms_num_to_save 0

# Specifies parallel multistart subproblem solve output control.
#   0 = no output from subproblem solves
#   1 = Subproblem output enabled, controlled by option 'outlev'.
#       Output is directed to a file 'knitro_ms_*.log'
ms_outsub 0

# Specifies the tolerance for deciding two feasible points are the
ms_savetol 1e-06

# Specifies the seed for random initialization of the multistart p
# Seed value should an integer >= 0.  Negative values will be rese
ms_seed 0

# Specifies the maximum range that any variable can vary over when
# multistart computes new start points.
ms_startptrange 1e+20

# Specifies conditions for terminating the multistart procedure.
#   maxsolves = 0 = terminate after maxsolves
#   optimal   = 1 = terminate at first local optimum
#   feasible  = 2 = terminate at first feasible solution estimate
ms_terminate maxsolves
```

```
# Specifies additional action to take after every iteration.
# Iterations result in a new point that is closer to a solution.
#   none      = 0 = no additional action
#   saveone   = 1 = save the latest new point to file knitro_newpoi
#   saveall   = 2 = append the latest new point to file knitro_newp
#   user      = 3 = allow a user-specified routine to run after ite
newpoint      none

# Valid range of obective values.
objrange      1e+20

# Specifies the final relative stopping tolerance for the KKT (opt
# error. Smaller values of opttol result in a higher degree of acc
# the solution with respect to optimality.
opttol        1e-06

# Specifies the final absolute stopping tolerance for the KKT (opt
# error. Smaller values of opttol_abs result in a higher degree of
# in the solution with respect to optimality.
opttol_abs    0.0

# Specifies whether to append to output files.
# This option should be set before calling KTR_init_problem().
#   no      = 0 = erase existing files when opening
#   yes     = 1 = append to existing files
outappend     no
```

```
# Directory for all output files.
# This option should be set before calling KTR_init_problem().
#outdir      .

# Specifies the verbosity of output.
#   none          = 0 = nothing
#   summary       = 1 = only final summary information
#   iter_10       = 2 = information every 10 iterations is printed
#   iter          = 3 = information at each iteration is printed
#   iter_verbose = 4 = more verbose information at each iteration
#   iter_x        = 5 = in addition, values of solution vector (x)
#   all           = 6 = in addition, constraints (c) and multiplier
outlev       iter_10

# Where to direct the output.
#   screen  = 0 = directed to stdout
#   file    = 1 = directed to a file called knitro.log
#   both    = 2 = both stdout and file called knitro.log
outmode      screen

# Whether to allow simultaneous evaluations in parallel.
#   no   = 0 = only one thread can perform an evaluation at a time
#   yes  = 1 = allow multi-threaded simultaneous evaluations
par_concurrent_evals  yes
```

```
# Number of threads to use in parallel features.
#   choose any positive integer, or
#    0 = value determined by OMP_NUM_THREADS environment variable
#   <0 = run sequential version of Knitro code
par_numthreads 1

# Specifies the initial pivot threshold used in the factorization
# The value must be in the range [0 0.5] with higher values result
# in more pivoting (more stable factorization). Values less than 0
# be set to 0 and values larger than 0.5 will be set to 0.5. If pi
# is non-positive initially no pivoting will be performed. Smaller
# may improve the speed of the code but higher values are recommen
# more stability.
pivot           1e-08

# Whether to apply a presolve operation to the model.
#   none   = 0 = no presolve
#   basic  = 1 = KNITRO performs basic presolve
presolve        basic

# Specifies the tolerance used to determine whether or not deduced
# from the presolve operation are infeasible.
presolve_tol 1e-06

# Whether to perform automatic scaling of the functions.
#   no     = 0 = no scaling done
#   yes    = 1 = KNITRO performs automatic scaling
scale           yes
```

```
# Whether to use the Second Order Correction (SOC) option.
#    no      = 0 = never do second order corrections
#    maybe   = 1 = SOC steps attempted on some iterations
#    yes     = 2 = SOC steps always attempted when constraints are n
soc             maybe

# Step size tolerance used for terminating the optimization.
xtol            1e-15
```

Numerical Gradients and Hessians Overview

Gradients and Hessians are often quite important:

- Choosing direction and step for Newtonian methods
- Evaluating convergence/non-convergence
- Estimating the information matrix (MLE)
- Note:
  - Solvers need accurate gradients to converge correctly
  - Solvers do not usually need precise Hessians
  - Must compute the information matrix accurately to get correct standard errors!
- Consequently, quick and accurate evaluation is important:
  - Hand-coded, analytic gradient/Hessian
  - Automatic differentiation
  - Numerical gradient/Hessian

Benefits of Analytic Gradient and Hessian

Where possible, you should use an analytic gradient and Hessian:

- Analytic Gradient
  - More accurate calculation of step and direction
  - Faster
- Analytic Hessian
  - Mostly provides faster convergence
  - Only code if used by your solver!
- An analytic gradient or Hessian is not a guarantee of numerical accuracy:
  - Numerical truncation from adding positive and negative numbers
  - Subtracting numbers is often dangerous
  - Summation error
  - Work in higher precision, e.g. in Matlab write a MEX file and use quad double

Forward Finite Difference Gradient

```
function [ fgrad ] = NumGrad( hFunc, x0, dx )
  x1    = x0 + dx ;
  f1    = feval( hFunc, x1 ) ;
  f0    = feval( hFunc, x0) ;
  fgrad = ( f1 - f0 ) / ( x1 - x0 ) ;
```

Need to tune step size: $h \sim 1e-6$ is a good start

Centered Finite Difference Gradient

```
function [ fgrad ] = NumGrad( hFunc, x0, dx )
  x1 = x0 + dx ;
  x2 = 2 * x0 - x1 ;
  f1 = feval( hFunc, x1 ) ;
  f2 = feval( hFunc, x2 ) ;
  fgrad = ( f1 - f2 ) / ( x1 - x2 ) ;
```

Overview of Hessian Pitfalls

'The only way to do a Hessian is to do a Hessian' – Ken Judd

- The 'Hessian' returned by `fmincon` is not a Hessian:
  - Computed by BFGS, sr1, or some other approximation scheme
  - A rank 1 update of the identity matrix
  - Requires at least as many iterations as the size of the problem
  - Dependent on quality of initial guess, $x0$
  - Often built with convexity restriction
- Therefore, you must compute the Hessian either numerically or analytically
- `fmincon`'s 'Hessian' often differs considerably from the true Hessian – just check eigenvalues or condition number

Condition Number

Use the condition number to evaluate the stability of your problem:

- $\texttt{cond}\,(A) = \log_{10}\left(\frac{\max\left[\texttt{eig}\,(A)\right]}{\min\left[\texttt{eig}\,(A)\right]}\right)$
- Large values $\Rightarrow$ trouble
- Also check eigenvalues: negative or nearly zero eigenvalues $\Rightarrow$ problem is not concave
- If the Hessian is not full rank, parameters will not be identified $\Rightarrow$ beware of problems which are not *numerically* identified
- Number of significant digits of precision lost == $\texttt{cond}\,(A)$

Estimating the Information Matrix

To estimate the information matrix:

1. Calculate the Hessian – either analytically or numerically
2. Invert the Hessian
3. Calculate standard errors

```
StandardErrors = sqrt( diag( inv( YourHessian ) ) ) ;
```

Assuming, of course, that your objective function is the likelihood...

Verification

Verifying your results is a crucial part of the scientific method:

- Generate a Monte Carlo data set: does your estimation code recover the target parameters?
- *T*est *D*riven *D*evelopment:
  1. Develop a unit test (code to exercise your function)
  2. Write your function
  3. Check that your function behaves correctly for all execution paths even if you have to write extra code to do so!
  4. The sooner you find a bug, the cheaper it is to fix!!!
- Start simple: e.g. logit with linear utility
- Then slowly add features one at a time, such as interactions or non-linearities
- Verify results via Monte Carlo
- Always compare analytic derivatives to finite difference
- Or, feed it a simple problem with an analytical solution

Diagnosing Problems

The solver provides information about its progress which can be used to diagnose problems:

- Enable diagnostic output during development
- The meaning of output depends on the type of solver: Interior Point, Active Set, etc.
- In general, you must RTFM: each solver is different

Information includes:

- Exit codes specifying type of failure
- Diagnositic output about progress
- Look for quadratic convergence – otherwise you may not have really solved the problem

Exit Codes

It is crucial that you check the optimizer's exit code and the gradient and Hessian of the objective function:

- Optimizer may not have converged:
  - Exceeded CPU time
  - Exceeded maximum number of iterations
  - Encountered numerical problems such as infeasible constraints, singular basis, ran out of memory
- Optimizer may not have found a global max
- Constraints may bind when they shouldn't (i.e., Lagrange multipliers $\lambda \neq 0$)
- Failure to check exit flags could lead to public humiliation and flogging

Interpreting Solver Output

Things to look for:

- Residual should decrease geometrically towards the end (Gaussian)
  - Then solver has converged
  - Geometric decrease followed by wandering around:
    - At limit of numerical precision
    - Increase precision and check scaling
- Linear convergence:
  - $\|residual\| \to 0$: rank deficient Jacobian $\Rightarrow$ lack of identification
  - Far from solution $\Rightarrow$ convergence to local min of $\|residual\|$
- Check values of Lagrange multipliers:
  - `lambda.{ upper, lower, ineqlin, eqlin, ineqnonlin, eqnonlin }`
  - Local min of constraint $\Rightarrow$ infeasible or locally inconsistent (IP)
  - Non convergence: failure of constraint qualification (NLP)
- Unbounded: λ or $x \to \pm\infty$

Solver Convergence

Listing 2: PATH: quadratic convergence

```
Major Iteration Log
major minor   func   grad   residual
    0     0      2      2 1.9982e+01
    1     1      3      3 5.3080e+00
    2     1      4      4 1.4611e+00
    3     1      5      5 2.6640e-01
    4     1      6      6 4.0062e-03
    5     1      7      7 1.0141e-06
    6     1      8      8 9.4265e-14
```

Solver Convergence

Listing 3: PATH: poor convergence

```
Major Iteration Log
major minor   func   grad   residual
    0     0      3      2 2.5101e+01
    1     1      4      3 1.0947e+01
    2    19      5      4 8.9594e+00
    3    21      6      5 1.8181e+00
    4    21      7      6 1.4533e+00
    5    21      8      7 1.2491e+00
    6    21      9      8 1.3063e+00
    7     1     10      9 0.0000e+00
```

Explore Your Objective Function

Visualizing your objective function will help you:

- Catch mistakes
- Choose an initial guess
- Determine if variable transformations, such as log or $x' = 1/x$, are necessary:
  - To change curvature
  - Impose a bound on a variable
  - To make problem more linear

Some tools:

- Plot objective function while holding all variables except one fixed
- Explore points near and far from the expected solution
- Contour plots better than 3-D plots
- Check for convexity at many points – can use inequalities

Ipopt

Ipopt is an alternative optimizer which you can use:

- Interior point algorithm
- Part of the COIN-OR collection of free optimization packages
- Supports C, C++, FORTRAN, AMPL, Java, MATLAB, and R
- Can be difficult to build – see me for details
- www.coin-or.org
- COIN-OR provides free software to facilitate optimization research

Floating Point Issues

A computer represents all numbers as a finite sequence of binary digits. Consequently, you can only represent a subset of the rational numbers which can lead to:

- Numerical roundoff errors
  - Machine epsilon provides upper bound on relative error
  - $\approx 2.220446049250313e-16$ in 64-bit MATLAB
  - Representation error: e.g., `(float) -3210.48 = -3210.4799804688`
  - Examine `eps()` in MATLAB or `std::numeric_limits<>::epsilon()` in C++
- Floating point exceptions: overflow & underflow
- Special numbers: `Inf` & `NaN`
- Some problems can be solved by using higher precision data types, e.g. `long double` or `quad double`.
- For more information:
  - IEEE 754 floating point specification
  - 'What Every Scientist Should Know About Floating-Point Arithmetic,' Goldberg, ACM, 1991.

Example: Overflow of short

```
iVal * 10 = 1000
iVal * 10 = 10000
iVal * 10 = -31072
iVal * 10 = 16960
iVal * 10 = -27008
iVal * 10 = -7936
iVal * 10 = -13824
iVal * 10 = -7168
iVal * 10 = -6144
iVal * 10 = 4096
iVal * 10 = -24576
iVal * 10 = 16384
iVal * 10 = -32768
iVal * 10 = 0
iVal * 10 = 0
iVal * 10 = 0
```

Example: Round-off Error for float

```
1.0 - (1.0 + 0.1) : -0.1
1.0 - (1.0 + 0.01) : -0.01
1.0 - (1.0 + 0.001) : -0.001
1.0 - (1.0 + 0.0001) : -0.0001
1.0 - (1.0 + 1e-05) : -1e-05
1.0 - (1.0 + 1e-06) : -1e-06
1.0 - (1.0 + 1e-07) : -1e-07
1.0 - (1.0 + 1e-08) : -1e-08
1.0 - (1.0 + 1e-09) : -1e-09
1.0 - (1.0 + 1e-10) : -1e-10
1.0 - (1.0 + 1e-11) : -1e-11
1.0 - (1.0 + 1e-12) : -1.00009e-12
1.0 - (1.0 + 1e-13) : -9.99201e-14
1.0 - (1.0 + 1e-14) : -9.99201e-15
1.0 - (1.0 + 1e-15) : -1.11022e-15
1.0 - (1.0 + 1e-16) : 0
1.0 - (1.0 + 1e-17) : 0
1.0 - (1.0 + 1e-18) : 0
```

The Evils of the Logit

Despite it's closed analytic form, the logit leads to many numerical problems:

- The root cause: $\exp(10) = \textit{large}$ and $\exp(1e3) = \text{Inf}$
- In addition, the exponential function is expensive to compute
- Renormalizing is can help....

$$s_j(x) = \frac{\exp(u_j - u_{max})}{\sum_k \exp(u_k - u_{max})}$$

but can lead to either overflow or underflow in the presence of an outside good

- Often shares are very small (e.g. BLP) leading to even smaller Jacobians (which are rank deficient) because $\frac{\partial s_j}{\partial p_k} = -\alpha_{price}(\mathbb{I}[j = k] - s_j)s_k$
- Going to higher numerical

Example: BLP Price Equilibrium

A great example of this problem is solving for the Bertrand-Nash price equilibrium in BLP:

- Highly non-linear for small p
- FOCs and shares $\to \infty$ exponentially as $p \to \infty$ so there are large flat regions near the optimum
- Symptom: solver converges poorly and to different points for different starts
- Solution: transform FOC so it is more linear:

$$foc_j = 1 + (p_j - c_j)\frac{\partial s_j}{\partial p_j} \cdot \frac{1}{s_j}$$

- Intuition comes from the logit where this is nearly linear for larger p

Exploit Structure

Often the problem has structure you can exploit to improve performance:

- Block Diagonal
- Solve blocks individually
- Avoids problems when blocks require
  - Different scaling
  - Different stopping conditions