An introduction into numerical optimization with KNITRO

Pawel Doligalski and Dominik Thaler

15 September 2014

KNITRO

	fval	fcount	time
fmincon	-103.6194	2197	1.578750
knitro a'la fmincon	-103.1450	144	0.094221
knitro a'la knitro	-151.1187	126	0.074921
knitro with 1st deriv	-151.1187	14	0.017827
knitro in AMPL	-151.1187	7	0.00111

Outline

Theory

1. Understanding an economic problem as an optimization problem
2. How to solve a simple optimization problem numerically
3. Global vs local optima
4. The crucial role of derivatives for derivative based solvers
5. Linear, quadratic, nonlinear and complementarity problems
6. Soft constraints vs hard bounds

Software

7. Software
8. Availability at the EUI

Implementation

9. Practical Example
10. Analytical derivatives in AMPL
11. Analytical derivatives with MATLAB

Practical Tips

12. The advantage of smooth problems and some tips how obtain them
13. Some options for fine tuning
14. Speed
15. Trouble shooting

1. Understanding an economic problem as an optimization problem

Unconstrained optimization

$$max_x f(x)$$

1. Understanding an economic problem as an optimization problem

Constrained optimization

$$max_x \ f(x)$$

$$s.t.$$

$$g(x) \geq 0$$

$$h(x) = 0$$

1. Understanding an economic problem as an optimization problem

Equation Solving as constrained optimization

$$max_x \ 0$$

$$s.t.$$

$$h(x) = 0$$

1. Understanding an economic problem as an optimization problem

A private economy as constrained optimization

$$max_x \; 0$$

$$s.t.$$

$$PrivateEquilibriumConditions(x) = 0$$

where x are the parameters of the policy function and $PrivateEquilibriumConditions(x)$ are the equilibrium conditions of the private sector (FOCs, market clearing)

1. Understanding an economic problem as an optimization problem

A Ramsey Problem as constrained optimization

$$max_x \; WelfareFunction(x)$$

$$s.t.$$

$$PrivateEquilibriumConditions(x) = 0$$

where $WelfareFunction(x)$ is the planners objective function

1. Understanding an economic problem as an optimization problem

Maximum Likelihood estimation as unconstrained optimization

$$max_y \ Likelihoodfunction(y, Data)$$

where y are the parameters to be estimated and $Likelihoodfunction(y, Data)$ is the likelihood function given some $Data$

1. Understanding an economic problem as an optimization problem

SMM Estimating a model as unconstrained optimization

$$min_y \ (Moments(Data) - SimulatedMoments(PolicyFunctions(y), y, Shocks))^2$$

where y are the deep parameters of the model and $PolicyFunctions(y)$ is the model solution as a function of the deep parameters (which is given), $Moments(Data)$ provides the moments of some given $Data$ and $SimulatedMoments(PolicyFunctions(y), y, Shocks)$ gives the simulated moments given the model solution, the model parameters and a long sequence of randomly drawn shocks)

1. Understanding an economic problem as an optimization problem

Simultaneously solving and SMM estimating a model as constrained optimization

$$min_{x,y} \ (Moments(Data) - SimulatedMoments(PolicyFunctions(x,y),y,Shocks))^2$$

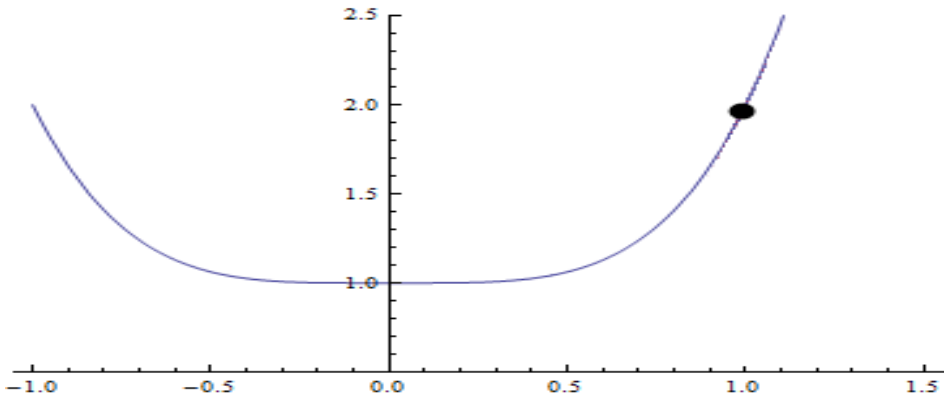
$$s.t.$$

$$PrivateEquilibriumConditions(x,y) = 0$$

2. How to solve a simple optimization problem numerically

Example 1: unconstrained optimization

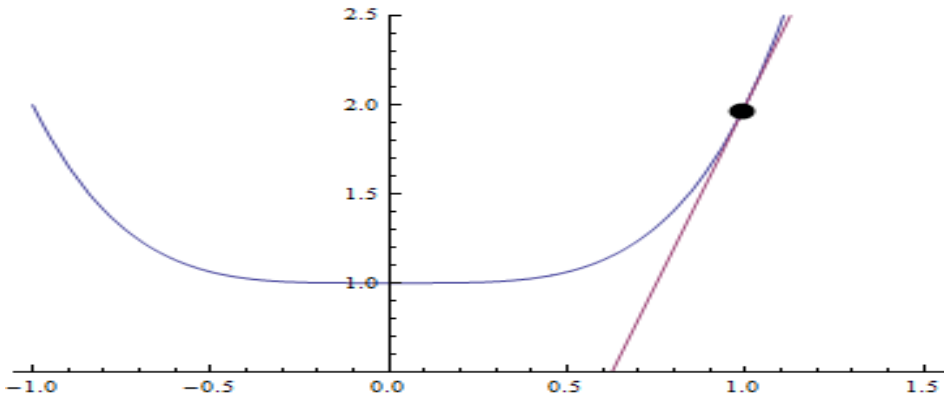
$$min_x \ f(x) = x^4 + 1$$



2. How to solve a simple optimization problem numerically

Example 1: unconstrained optimization

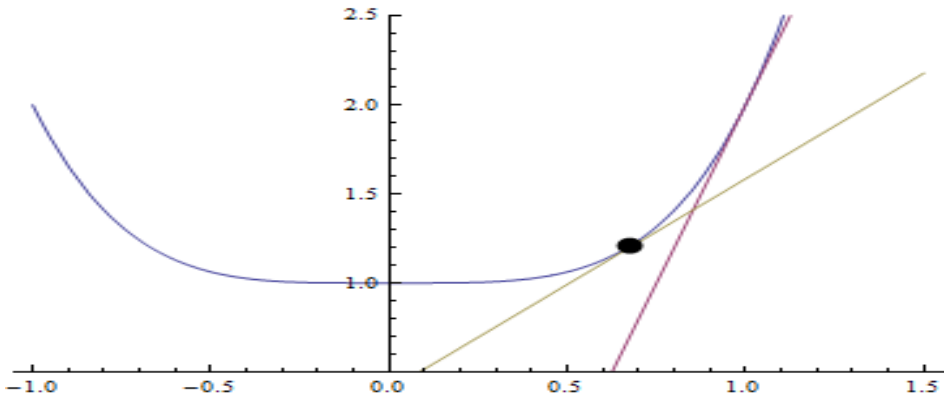
$$min_x \ f(x) = x^4 + 1$$



2. How to solve a simple optimization problem numerically

Example 1: unconstrained optimization

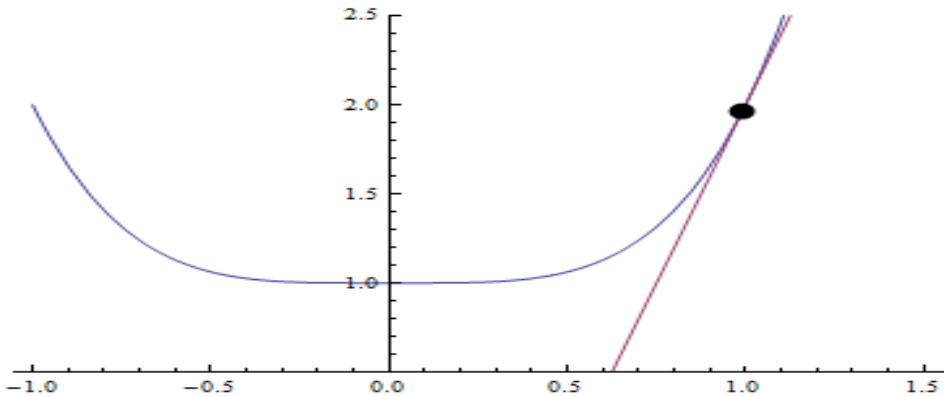
$$min_x \ f(x) = x^4 + 1$$



2. How to solve a simple optimization problem numerically

Example 1: unconstrained optimization

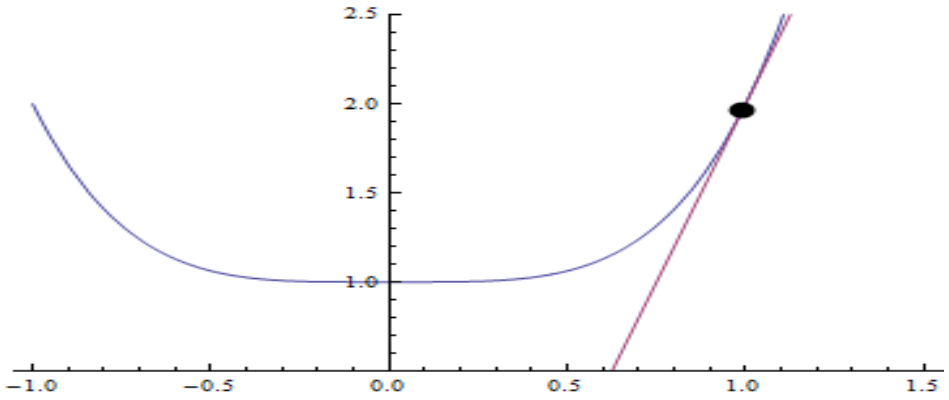
$$min_x \ f(x) = x^4 + 1$$



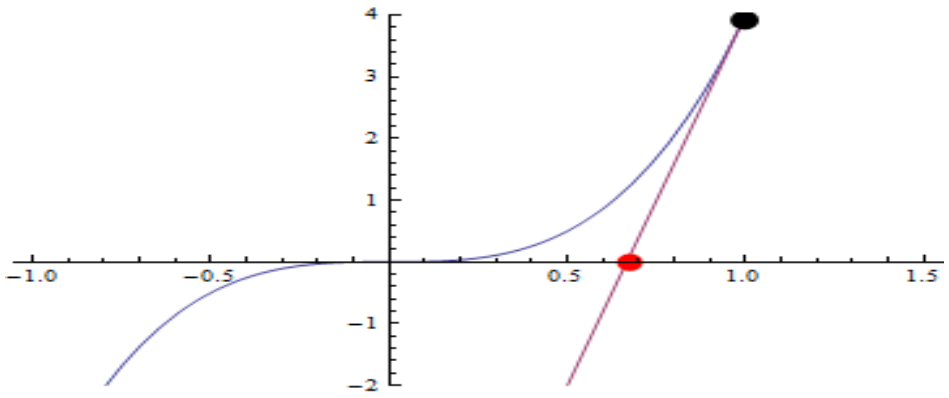
2. How to solve a simple optimization problem numerically

Example 1: unconstrained optimization (newton's method)

$$min_x \ f(x) = x^4 + 1$$



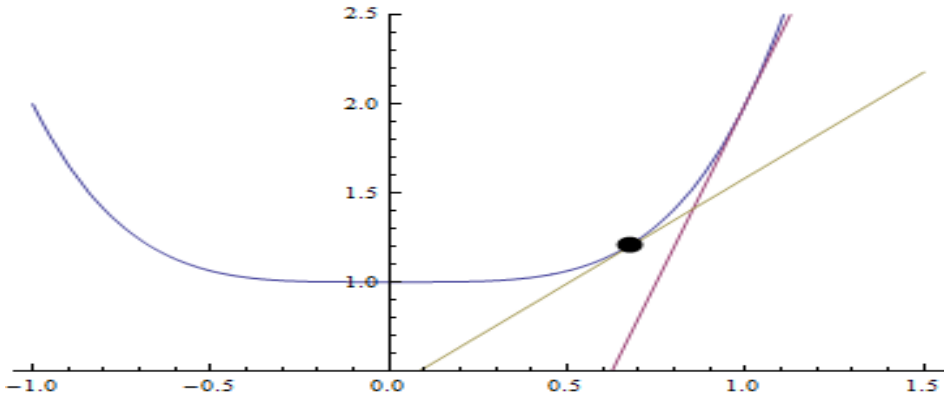
$$f'(x) = 4x^3$$



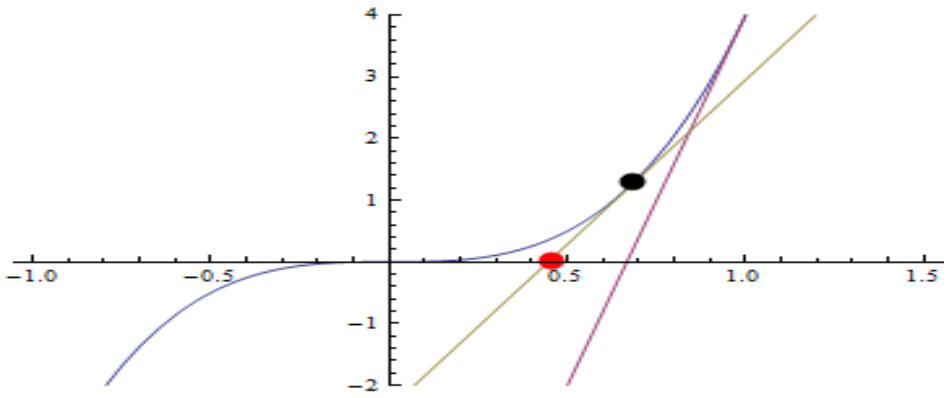
2. How to solve a simple optimization problem numerically

Example 1: unconstrained optimization (newton's method)

$$min_x \ f(x) = x^4 + 1$$



$$f'(x) = 4x^3$$



2. How to solve a simple optimization problem numerically

Example 1: unconstrained optimization

$$min_x \ f(x) = x^4 + 1$$

Example 2: equation solving

$$F(x) = 4x^3 = 0$$

$$F(x) = \frac{\partial f(x)}{\partial x} = 4x^3 = 0$$

- Optimization = Root finding of derivative = Equation solving
- Global convergence: for well behaved (smooth) functions (more complicated, conceptually similar) algorithms converge in finite number of iterations from any starting point

2. How to solve a simple optimization problem numerically

Example 1: unconstrained optimization

$$min_x \ f(x) = x^4 + 1$$

Example 2: equation solving

$$F(x) = 4x^3 = 0$$

$$F(x) = \frac{\partial f(x)}{\partial x} = 4x^3 = 0$$

- Optimization = Root finding of derivative = Equation solving
- Global convergence: for well behaved (smooth) functions (more complicated, conceptually similar) algorithms converge in finite number of iterations from any starting point

2. How to solve a simple optimization problem numerically

Example 1: unconstrained optimization

$$min_x \ f(x) = x^4 + 1$$

Example 2: equation solving

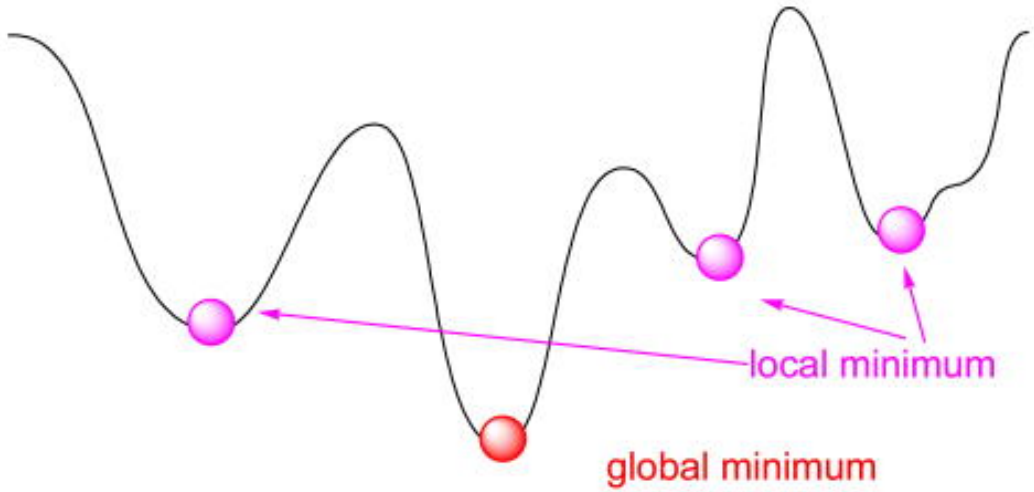
$$F(x) = 4x^3 = 0$$

$$F(x) = \frac{\partial f(x)}{\partial x} = 4x^3 = 0$$

- Optimization = Root finding of derivative = Equation solving
- Global convergence: for well behaved (smooth) functions (more complicated, conceptually similar) algorithms converge in finite number of iterations from any starting point

3. Global vs local optima

- The solutions numerical routines find are always local
  - an optimum found numerically is one local optimum of possibly many
  - a root found numerically is one root of possibly many
- Global solutions can not be found numerically
  - Remedies:
    - Start from different starting points and hope to find all optima
    - If you know how many local optima there are you can try to find them all (E.g. minimum of convex function over convex constraint set is unique)



4. The crucial role of derivatives for derivative based solvers

Direction

- 2 possibilities:
  - move in random direction continue if right direction (derivative free method, fminsearch, new KNITRO option)
  - know derivative, move downward
- derivatives (jacobian) can be found
  - analytically
    - analytical derivatives faster especially in higher dimensions (evaluation and convergence)
  - numerically (finite differences)
    - forward differences: $\frac{\partial f(x_i,x_{-i})}{\partial x} \approx \frac{f(x_i,x_{-i})+f(x_i+\Delta,x_{-i})}{\Delta}$
    - central differences: $\frac{\partial f(x_i,x_{-i})}{\partial x_i} \approx \frac{f(x_i-\Delta,x_{-i})+f(x_i+\Delta,x_{-i})}{2\Delta}$ (twice slower, more precise)
    - numerical derivatives are imprecise and requires 1 or 2 evaluations of the function per dimension
    - When using numerical derivatives save computing time by providing a pattern (zero, nonzero)

4. The crucial role of derivatives for derivative based solvers

Step size

- 2 possibilities:
  - heuristic step size, (e.g. as function of previous step)
  - second order derivatives (hessian) to approximate minimum
- second order derivatives (hessian) can be found either analytically or approximated numerically
  - Full hessian costly to evaluate
  - Approximation of hessian doesn't require full hessian and might be faster
  - Heuristics might be faster

4. The crucial role of derivatives for derivative based solvers

- My experience: supply derivatives, forget about hessian.
- Standard in computational macro, not sure about nonlinear estimation.
- Analytical derivatives impossible if your objective/constraint function contains black boxes like DYNARE (e.g. OSR or GMM)

5. The differences between linear, quadratic, nonlinear and complementarity problems

- Linear optimization
  - $f(x)$, $g(x)$, $h(x)$ can be written as $Ax - b$
  - has either none, one or a continuum of solutions
  - can be solved in one iteration
  - solving a linear equation system is same as matrix inversion (is feasible where the standard inversion becomes infeasible due to memory constraints (for large matrices))
- Quadratic problems
  - are special cases of nonlinear problems that are easier to solve (not possible to declare in KNITRO-MATLAB)
- General nonlinear problems
- Complementarity problems (MPEC)
  - $h(x) = 0$ can be written as $min[\ i(x)\ ,\ j(x)\ ] = 0$
- Tell the solver what problem u have (automatic in AMPL, not in MATLAB)

6. Soft constraints vs hard bounds

- bounds are of the type $x_i \geq a$
- bounds could be written as linear constraints
- writing them as bounds makes sure that they are never violated during the iterative solution algorithm (in KNITRO, unless otherwise specified)
- important for functions that are not defined over $\mathbb{R}$ like log, sqrt etc.
  - e.g. utility $u(c) = log(c)$ requires $c > 0$
    - implement as $c \geq 1e-10$

7. Software

Mathematical coding languages (MATLAB, FORTRAN, AMPL)

$\Updownarrow$ Interface

Solver (fmincon, fminsearch, KNITRO, ...)

- Mathematical coding language is used to evaluate the objective and constraint functions $f(x)$, $g(x)$, $h(x)$ and their derivatives
- Solver decides where to evaluate them: x
  - Typically most computing time used for evaluation

7. Software

- One could write its own solver in any language, but good solves are very complicated
- Some ready made solvers are product specific (fmincon - MATLAB)
- Others have interfaces to many products (KNITRO - MATLAB/FORTRAN/C++/AMPL/PYTHON...)
- Almost all that is discussed in this course about KNITRO is true for many other solvers
- KNITRO is probably the best product in the industry
- Advantages of KNITRO compared to fmincon:
  - faster and more reliable
  - more options
  - reliably solves complementarity problems

8. Availability at the EUI

- KNITRO (Solver)
  - 10 user licenses administered by Arpad (1 user, 1 computer, unlimited threads/cores)
  - 1 floating license (1 user and computer at a time, 1 thread/core)
  - free student licenses
    - max 300 variables and constraints
    - for 6 month, only once per computer
  - Interfaces to MATLAB/AMPL/PYTHON/FORTRAN/C++...
- AMPL (Mathematical coding language)
  - free student licenses
    - max 300 variables and constraints
    - unlimited time
  - Interfaces to KNITRO and many other solvers

8. Availability at the EUI

- NEOS Server
  - server with over 60 solvers, incl. KNITRO
  - no solver license required
  - huge computing power available for free (apparently incl. parallel)
  - easy to use with AMPL
    - write code in AMPL
    - send to NEOs server who executes immediately (either via website www.neos-server.org or via AMPL (Kestrel))
    - receive results (in AMPL (Kestrel) or via website)
    - Kestrel is currently blocked by EUI firewall
  - works with other languages as well (FORTRAN, GAMS, ...) but not with MATLAB

9. Practical Example
Complementarity constraints

Ramsey problem of setting a linear income tax t and lump-sum transfer T in the economy with 2 different agents.

$$\max_{\{c_1, n_1, c_2, n_2, t, T\}} U_1(c_1, n_1) + U_2(c_2, n_2)$$

$$\begin{array}{lccc} s.t. & & & \\ \text{for } i = 1,2 & c_i & = & (1-t) w_i n_i + T \\ \text{for } i = 1,2 & n_i & = & \arg\max_{n \geq 0} U_i((1-t) w_i n + T, n) \\ & t(w_1 n_1 + w_2 n_2) & = & E + 2T \end{array}$$

9. Practical Example
Complementarity constraints

How to express $n_i = \arg\max_{n \geq 0} U_i\left((1-t)\, w_i n + T, n\right)$?
We can use the first order condition. Define

$$slack_i = -\left(\frac{\partial U_i}{\partial n_i} + (1-t)\, w_i \frac{\partial U_i}{\partial c_i}\right).$$

Now the constraint is equivalent to

$$\underbrace{slack_i = 0,\ n_i \geq 0}_{\text{interior solution}} \quad \text{or} \quad \underbrace{slack_i \geq 0,\ n_i = 0}_{\text{corner solution}}$$

We can write it more compactly as

$$slack_i \times n_i = 0,\ slack_i \geq 0,\ n_i \geq 0.$$

9. Practical Example
Writing the problem a'la fmincon

Solve

$$\min_{\{c_1, n_1, c_2, n_2, t, T\}} -\left(U_1\left(c_1, n_1\right) + U_2\left(c_2, n_2\right)\right)$$

s.t. equality constraints

$$\begin{array}{lrcl}
\text{for } i = 1, 2 & c_i - (1-t)\, w_i n_i + T & = & 0 \\
& t\left(w_1 n_1 + w_2 n_2\right) - E + 2T & = & 0 \\
\text{for } i = 1, 2 & n_i \times \left(\frac{\partial U_i}{\partial n_i} + (1-t)\, w_i \frac{\partial U_i}{\partial c_i}\right) & = & 0
\end{array}$$

s.t. inequality constraints

$$\text{for } i = 1, 2 \quad \frac{\partial U_i}{\partial n_i} + (1-t)\, w_i \frac{\partial U_i}{\partial c_i} \leq 0.$$

9. Practical Example
Writing the problem a'la knitro

Solve

$$\min_{\{c_1, n_1, c_2, n_2, t, T, slack_1, slack_2\}} -\left(U_1\left(c_1, n_1\right) + U_2\left(c_2, n_2\right)\right)$$

s.t. equality constraints

$$\begin{array}{lrcl}
\text{for } i = 1,2 & c_i - (1-t)\, w_i n_i + T & = & 0 \\
& t\left(w_1 n_1 + w_2 n_2\right) - E + 2T & = & 0 \\
\text{for } i = 1,2 & slack_i & = & -\left(\frac{\partial U_i}{\partial n_i} + (1-t)\, w_i \frac{\partial U_i}{\partial c_i}\right)
\end{array}$$

s.t. complementarity constraints

$$\text{for } i = 1,2 \quad n_i \quad \text{"complements"} \quad slack_i.$$

9. Practical Example
Results

	fval	**fcount**	**time**
fmincon	-103.6194	2197	1.578750
knitro a'la fmincon	-103.1450	144	0.094221
knitro a'la knitro	-151.1187	126	0.074921
knitro with 1st deriv	-151.1187	14	0.017827
knitro in AMPL	-151.1187	7	0.00111

10. Analytical derivatives in AMPL: advantages and limitations

- Calculating derivatives by hand is time consuming and prone to algebraic and coding errors
- In AMPL derivatives are calculated automatically.
- Everything you can do in AMPL is smooth
- Limitations:
  - The calculation of these derivatives is done automatically and not necessarily efficiently
    - Function and derivative evaluation be slower than MATLAB
  - Very limited set of functions (no interpolation exception: 1D linear)
  - No graphical user interface

11. Obtaining analytical derivatives with MATLAB's symbolic toolbox and the matlabfunction command

- Calculating derivatives by hand is time consuming and prone to algebraic and coding errors.
- In MATLAB the symbolic toolbox can help
  - Write objective function and constraint functions as symbolic expressions
  - Derive them using symbolic toolbox
  - Automatically create a function mfile that evaluates the resulting expressions in an efficient manner with matlabfunction
  - Use them directly with anonymous functions or copy paste these functions into your nested function matlab optimization mfile
  - If necessary add adjustments (manually entered computations)
- Limitations:
  - symbolic toolbox can only handle scalars and elementary functions. Do rest by hand (use derivativeCheck)
  - Not fully automated as in AMPL
- Example can be found in MATLAB help (search "Using Symbolic Mathematics with Optimization Toolbox™ Solvers")

12. The advantage of smooth problems and some tips how obtain them

- Numerical solvers like KNITRO are suitable only for smooth problems ($f(x)$, $g(x)$ and $h(x)$ are smooth , i.e. continuous and differentiable $\forall x \in [lb, ub]$)
- The solver looks for local optima and identifies them by points with zero derivatives
  - Zero derivatives do not identify minima at kinks
- The solver chooses the direction of search and the step size using derivatives
  - Derivatives do not approximate discontinuous or nondifferentiable functions
- Most real economic models are smooth in their nature
- I speculate that most empirical optimizations are also smooth (e.g. OLS, L1)

12. The advantage of smooth problems and some tips how obtain them

- Problems can be non continuous because...
  - they are discrete because...
    - They are really discrete (e.g. game theory): do not use KNITRO
    - We discretize a decision variable (e.g. in VFI): Either use interpolation instead of discretizing or do not use KNITRO (Exogenous states don't matter normally)
    - If mixed discrete continuous problem: use Mixed Integer Nonlinearly Constrained Optimization (KNITRO can apparently do that too) or split in a discrete number of continuous sub-problems and use KNITRO normally on sub-problems
  - there are jumps:
    - Split in smooth sub-problems at the jump point
    - Avoid jumps if anyhow possible (e.g. approximate by smooth function)
    - Never introduce jumps as a "penalty" for undefined regions. Use bounds.

12. The advantage of smooth problems and some tips how obtain them

- Problems can be non differentiable because
  - We use non differentiable functions like linear interpolation to approximate functions (this might include the numerical integration method in special cases):
    - Use smooth interpolation (Chebychev polynomials, cubic splines; in MATLAB do by hand to be able to differentiate and for speed)
  - We do not extrapolate:
    - extrapolate carefully or better make sure we never need to (especially with Chebychev polynomials)
  - Our economic problem contains fundamental non-differentiabilities like max functions:
    - Write as complementarity problem
    - Approximate by smooth functions

12. The advantage of smooth problems and some tips how obtain them

- You may be lucky and get away with some non-smoothness if its at the right places (away from the solution) or if its almost smooth (small kinks, jumps).
  - It's always worth a try to work with a non smooth problem if the problem is already coded
  - But when you formulate the problem from scratch avoid non-smoothness wherever you can
- If the solver fails and the last point of evaluation was at a kink you know that kink is the problem

13. Some options for fine tuning

- All options can be passed to KNITRO via an .ops file. Most also via a MATLAB options structure (optimset)
- GradDeriv, ObjDeriv: whether to supply 1st derivatives
- Algorithms: 4 different algorithms available
- Tolerances (TolX, TolFun, TolCon): defines what is considered a solution
- MaxFunEvals, MaxIter: when to give up
- Output: what to print to the screen
- Multistart (only .ops file): run repeatedly from multiple starting points
- Parallel: evaluate finite differences in parallel, irrelevant if analytical derivative provided
- Tuner (only .ops file): automatically looks for best options
- Nonlinear least squares: special algorithm available using command "knitromatlab_lsqnonlin"

14. Speed

General:

- Initial guesses (warm start)
  - Homotopy
  - When doing things on a grid obtain guesses from close grid points that are already solved
  - In MPECs avoid initializing complementarity variables at 0
  - In AMPL you can initialize Lagrangian multipliers too
- Supply derivatives
- Guarantee smoothness

In MATLAB:

- Don't do unnecessary calculations (use if)
  - Reuse calculations that are common in the constraint and in the objective
  - Only calculate the derivative when needed (use nargout)
- To pass parameters to constraint and objective functions use shared variables in nested functions instead of anonymous functions
- General advice: use profiler

15. Trouble shooting

- The fact that you don't get a red error message in MATLAB doesnt mean the solver was sucessful. Always check exitflag!
- If the solver fails
  - Look at the exitflag and read the manual
    - No precises solution (not smooth, no analytical derivatives)
    - No feasible point (mistake in constraints)
    - Evaluation not possible (encountered NaN, maybe due to missing bound or coding error)
  - Set option output to "iter-detailed"
  - Look at the last guess, the solver might stop at a Kink or Jump
  - In MATLAB have crucial diagnostics printed at each evaluation of the constraint or objective function
  - In MATLAB set breakpoints to analyze what happens in the constraint or objective function at each iteration

15. Trouble shooting

- If you use KNITRO repeatedly and it happens to fail for a small fraction:
  - The 1% failures can cost you 99% of the computation time...
  - Don't allow too many iterations to avoid time consuming dead ends (restrict maxiter, maxfuneval)
  - Try to find out whats special about the failing points
  - Use "if exitflag~=0" to try out different options (algorithms), starting values etc.