

Deterministic Models

Perfect foresight, nonlinearities and occasionally binding constraints

Sébastien Villemot



28 May 2024

Introduction

- Perfect foresight = agents perfectly anticipate all future shocks
- Concretely, at period 1:
 - ▶ agents learn the value of all future shocks;
 - ▶ since there is shared knowledge of the model and of future shocks, agents can compute their optimal plans for all future periods;
 - ▶ optimal plans are not adjusted in periods 2 and later
⇒ the model behaves as if it were deterministic.
- Cost of this approach: the effect of future uncertainty is not taken into account (e.g. no precautionary motive)
- Advantage: numerical solution can be computed exactly (up to rounding errors), contrarily to perturbation or global solution methods for rational expectations models
- In particular, nonlinearities fully taken into account (e.g. occasionally binding constraints)

Outline

- 1 Presentation of the problem
- 2 Solution techniques
- 3 Shocks: temporary/permanent, unexpected/pre-announced
- 4 Occasionally binding constraints
- 5 More unexpected shocks
- 6 Extended path

Outline

- 1 Presentation of the problem
- 2 Solution techniques
- 3 Shocks: temporary/permanent, unexpected/pre-announced
- 4 Occasionally binding constraints
- 5 More unexpected shocks
- 6 Extended path

The (deterministic) neoclassical growth model

$$\max_{\{c_t\}_{t=1}^{\infty}} \sum_{t=1}^{\infty} \beta^{t-1} \frac{c_t^{1-\sigma}}{1-\sigma}$$

s.t.

$$c_t + k_t = A_t k_{t-1}^{\alpha} + (1 - \delta)k_{t-1}$$

First order conditions:

$$c_t^{-\sigma} = \beta c_{t+1}^{-\sigma} (\alpha A_{t+1} k_t^{\alpha-1} + 1 - \delta)$$

$$c_t + k_t = A_t k_{t-1}^{\alpha} + (1 - \delta)k_{t-1}$$

Steady state:

$$\bar{k} = \left(\frac{1 - \beta(1 - \delta)}{\beta \alpha \bar{A}} \right)^{\frac{1}{\alpha-1}}$$

$$\bar{c} = \bar{A} \bar{k}^{\alpha} - \delta \bar{k}$$

Note the absence of stochastic elements! No expectancy term, no probability distribution

Dynare code (1/3)

rcb_basic.mod

```
var c k;  
varexo A;  
parameters alpha beta gamma delta;  
  
alpha=0.5;  
beta=0.95;  
gamma=0.5;  
delta=0.02;  
  
model;  
  c + k = A*k(-1)^alpha + (1-delta)*k(-1);  
  c^(-gamma) = beta*c(+1)^(-gamma)*(alpha*A(+1)*k^(alpha-1) + 1 - delta);  
end;
```

Dynare code (2/3)

rcb_basic.mod

```
// Give an initial guess for the steady state solver
initval;
  A = 1;
  k = 40;
  c = 5;
end;

// Actually compute the steady state
steady;
```

Dynare code (3/3)

rcb_basic.mod

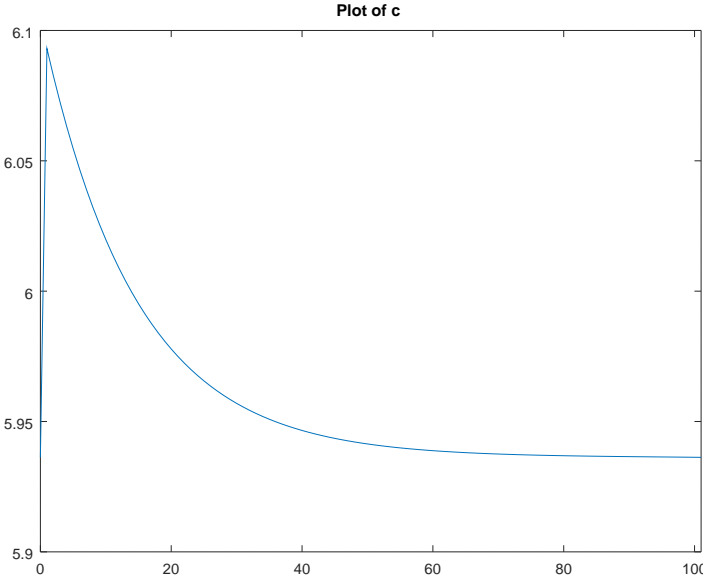
```
// Declare a positive technological shock in period 1
shocks;
  var A;
  periods 1;
  values 1.2;
end;

// Prepare the deterministic simulation over 100 periods
perfect_foresight_setup(periods=100);

// Perform the simulation
perfect_foresight_solver;

// Display the path of consumption
rplot c;
```


Simulated consumption path



The general problem

Deterministic, perfect foresight, case:

$$f(y_{t+1}, y_t, y_{t-1}, u_t) = 0$$

y : vector of endogenous variables

u : vector of exogenous shocks

Identification rule: as many endogenous (y) as equations (f)

Return to the neoclassical growth model

$$y_t = \begin{pmatrix} c_t \\ k_t \end{pmatrix}$$

$$u_t = A_t$$

$$f(y_{t+1}, y_t, y_{t-1}, u_t) = \begin{pmatrix} c_t^{-\sigma} - \beta c_{t+1}^{-\sigma} (\alpha A_{t+1} k_t^{\alpha-1} + 1 - \delta) \\ c_t + k_t - A_t k_{t-1}^{\alpha} + (1 - \delta) k_{t-1} \end{pmatrix}$$

What if more than one lead or one lag?

- A model with more than one lead or lag can be transformed in the form with one lead and one lag using *auxiliary* variables
- Transformation done automatically by Dynare
- For example, if there is a variable with two leads x_{t+2} :
 - ▶ create a new auxiliary variable a
 - ▶ replace all occurrences of x_{t+2} by a_{t+1}
 - ▶ add a new equation: $a_t = x_{t+1}$
- Symmetric process for variables with more than one lag
- With future uncertainty, the transformation is more elaborate (but still possible) on variables with leads
- `write_latex_original_model` and `write_latex_dynamic_model` show model before and after substitution

Steady state

- A steady state, \bar{y} , for the model satisfies

$$f(\bar{y}, \bar{y}, \bar{y}, \bar{u}) = 0$$

- Note that a steady state is conditional to:
 - ▶ The steady state values of exogenous variables \bar{u}
 - ▶ The value of parameters (implicit in the above definition)
- Even for a given set of exogenous and parameter values, some (nonlinear) models have several steady states
- Two possibilities for the steady state in Dynare:
 - ▶ If analytical steady state is known, can be given in `steady_state_model` block
 - ▶ Otherwise, numerical procedure based on a nonlinear solver, with the `steady` command

Towards a solution

- In each period, agent needs to choose control variables (e.g. consumption) today, taking into account:
 - ▶ the known time path of the exogenous variables
 - ▶ the current endogenous states that cannot be altered anymore (e.g. capital)
 - ▶ the consequences of today's actions for future decisions
 - Assume finite horizon problem that terminates in period T with
 - ▶ initial states in y_0 given
 - ▶ terminal choice y_{T+1} known
- ⇒ problem boils down to simultaneous equations system with $n_y \times T$ equations in $n_y \times T$ unknowns (where n_y is the number of endogenous)

A two-boundary value problem

- Stacked system for a perfect foresight simulation over T periods:

$$\left\{ \begin{array}{l} f(y_2, y_1, y_0, u_1) = 0 \\ f(y_3, y_2, y_1, u_2) = 0 \\ \vdots \\ f(y_{T+1}, y_T, y_{T-1}, u_T) = 0 \end{array} \right.$$

for y_0 and y_{T+1} given.

- Compact representation:

$$F(Y) = 0$$

where $Y = [y'_1 \ y'_2 \ \dots \ y'_T]'$
and $y_0, y_{T+1}, u_1 \dots u_T$ are implicit

- Resolution uses a Newton-type method on the stacked system

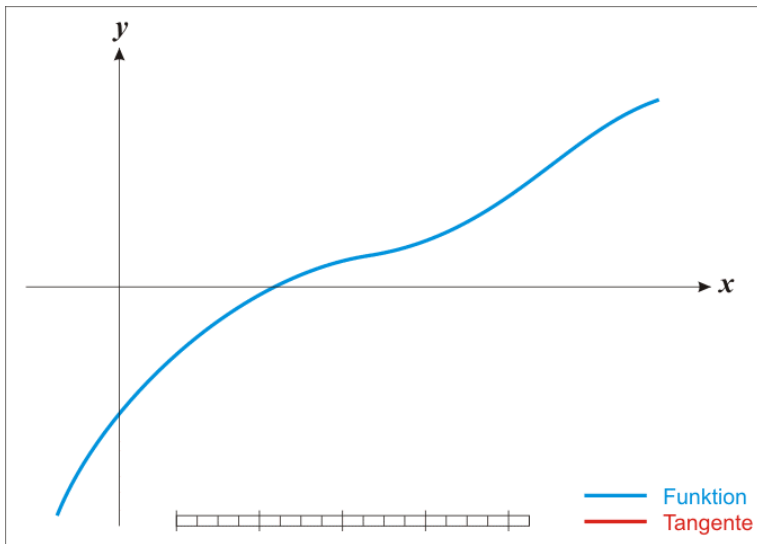
Approximating infinite-horizon problems

- The above technique numerically computes trajectories for given shocks over a *finite* number of periods
- Suppose you are rather interested in solving an *infinite*-horizon problem
- One option consists in computing the recursive policy function (as with perturbation methods), but this is challenging
 - ▶ in the general case, this function is defined over an infinite-dimensional space (because all future shocks are state variables)
 - ▶ in the particular case of a return to equilibrium, the state-space is finite (starting from the date where all shocks are zero), but a projection method would still be needed
 - ▶ in any case, Dynare does not do that
- An easier way, in the case of a return to equilibrium, is to approximate the solution by a finite-horizon problem
 - ▶ consists in computing the trajectory with $y_{T+1} = \bar{y}$ and T large enough
 - ▶ drawback compared to the policy function approach: the solution is specific to a given sequence of shocks, and not generic

Outline

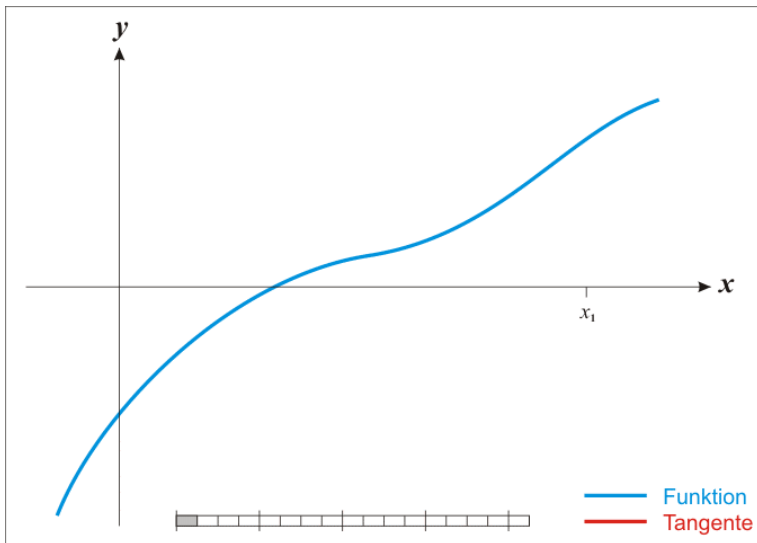
- 1 Presentation of the problem
- 2 Solution techniques**
- 3 Shocks: temporary/permanent, unexpected/pre-announced
- 4 Occasionally binding constraints
- 5 More unexpected shocks
- 6 Extended path

The Newton method (unidimensional)



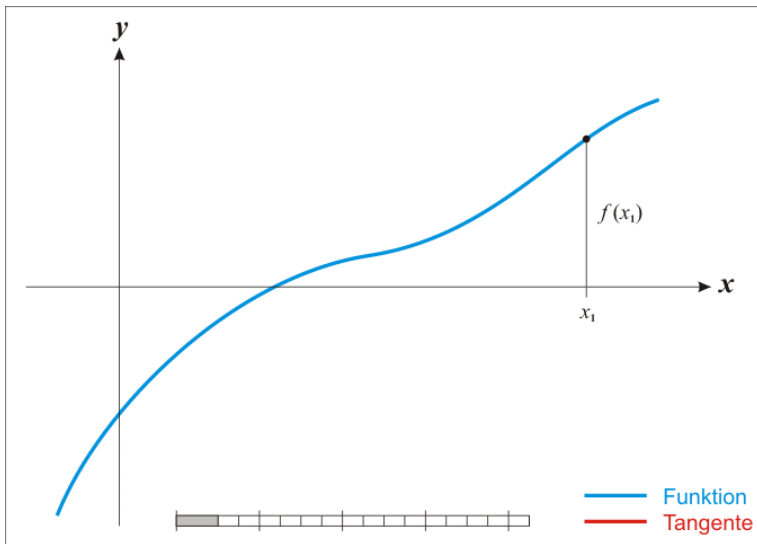
Copyright © 2005 Ralf Pfeifer / Creative Commons Attribution-ShareAlike 3.0

The Newton method (unidimensional)



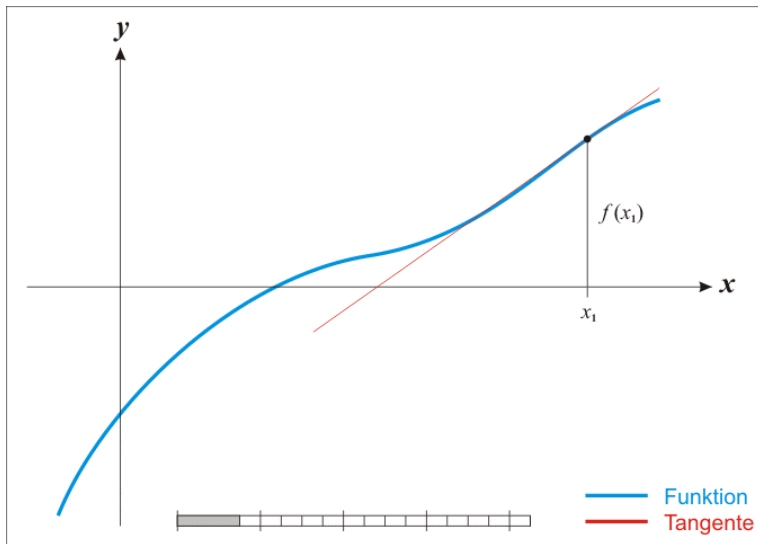
Copyright © 2005 Ralf Pfeifer / Creative Commons Attribution-ShareAlike 3.0

The Newton method (unidimensional)



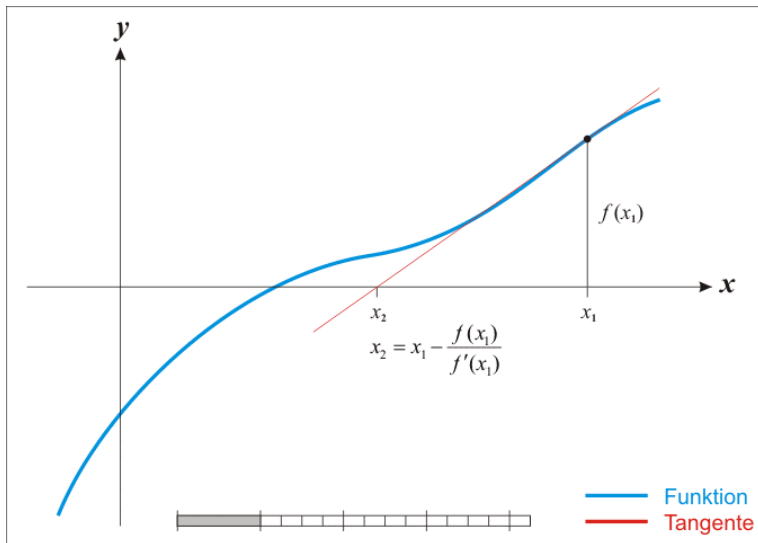
Copyright © 2005 Ralf Pfeifer / Creative Commons Attribution-ShareAlike 3.0

The Newton method (unidimensional)



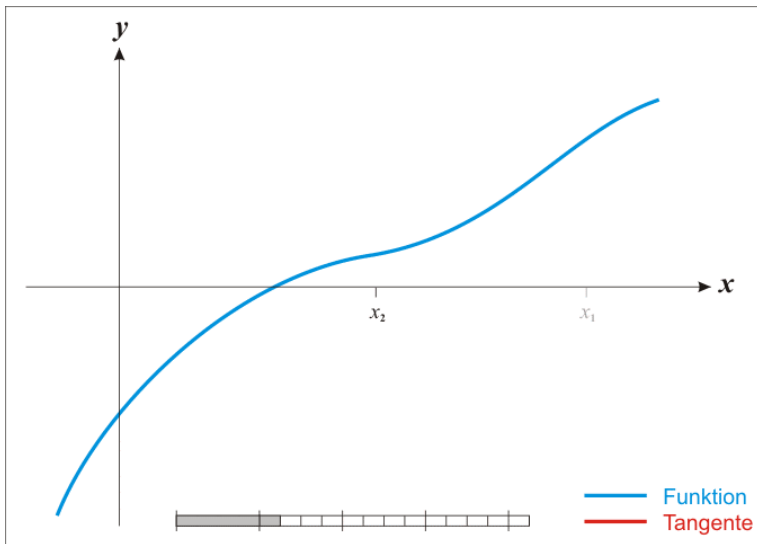
Copyright © 2005 Ralf Pfeifer / Creative Commons Attribution-ShareAlike 3.0

The Newton method (unidimensional)



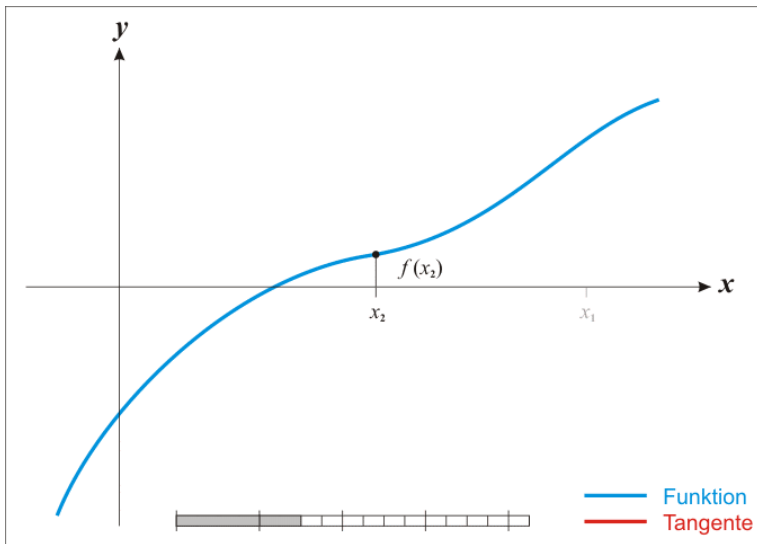
Copyright © 2005 Ralf Pfeifer / Creative Commons Attribution-ShareAlike 3.0

The Newton method (unidimensional)



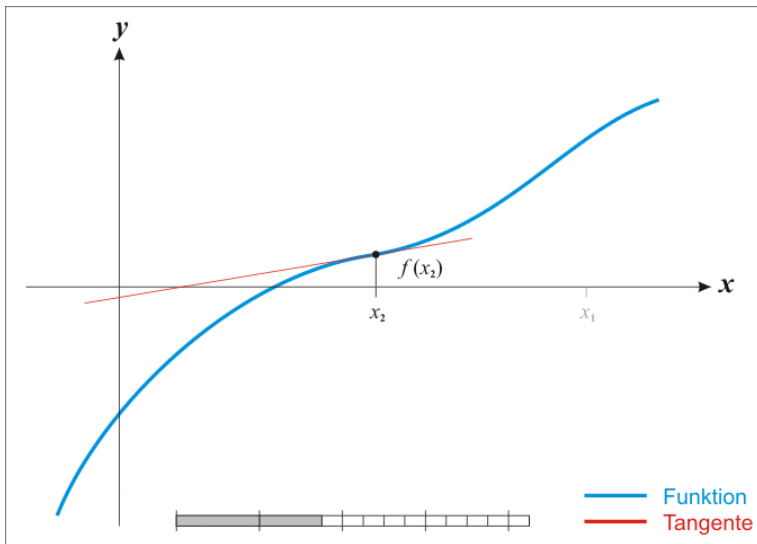
Copyright © 2005 Ralf Pfeifer / Creative Commons Attribution-ShareAlike 3.0

The Newton method (unidimensional)



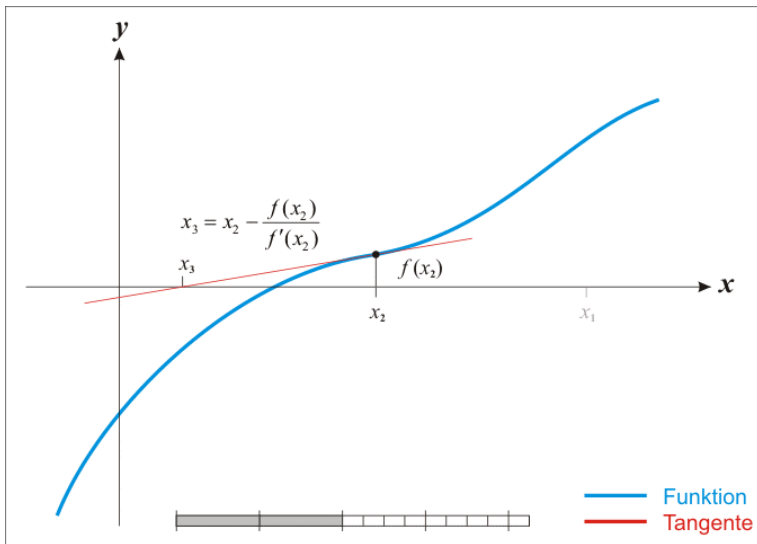
Copyright © 2005 Ralf Pfeifer / Creative Commons Attribution-ShareAlike 3.0

The Newton method (unidimensional)



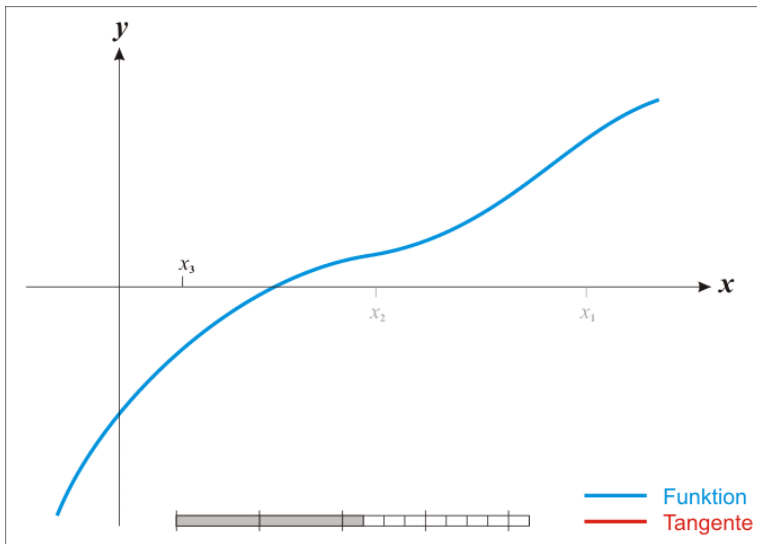
Copyright © 2005 Ralf Pfeifer / Creative Commons Attribution-ShareAlike 3.0

The Newton method (unidimensional)



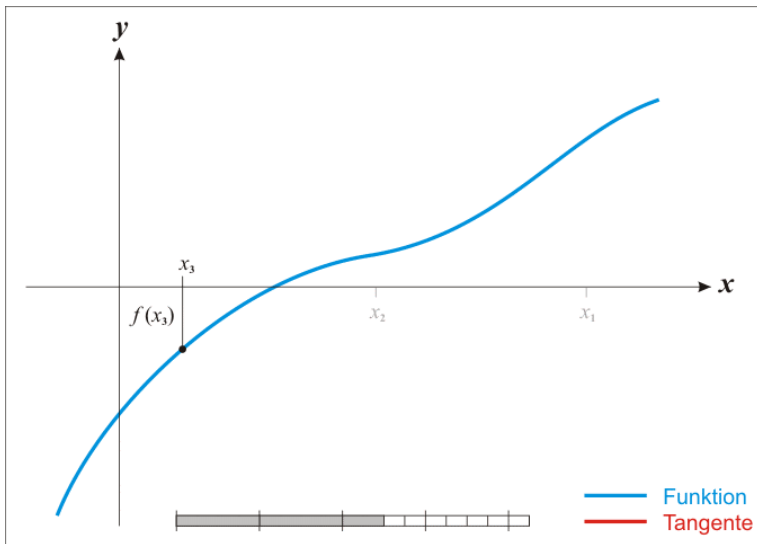
Copyright © 2005 Ralf Pfeifer / Creative Commons Attribution-ShareAlike 3.0

The Newton method (unidimensional)



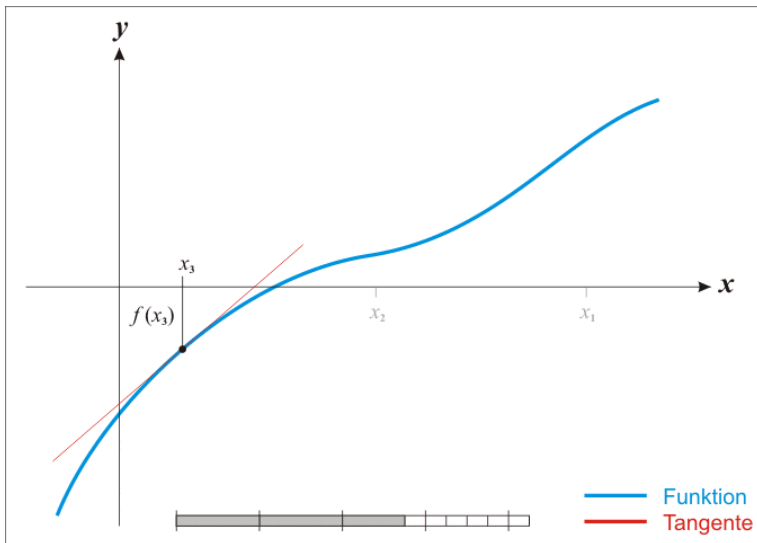
Copyright © 2005 Ralf Pfeifer / Creative Commons Attribution-ShareAlike 3.0

The Newton method (unidimensional)



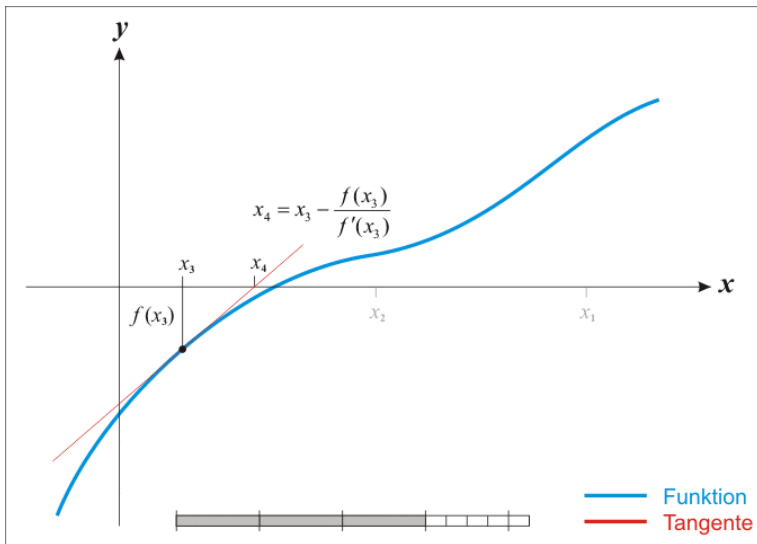
Copyright © 2005 Ralf Pfeifer / Creative Commons Attribution-ShareAlike 3.0

The Newton method (unidimensional)



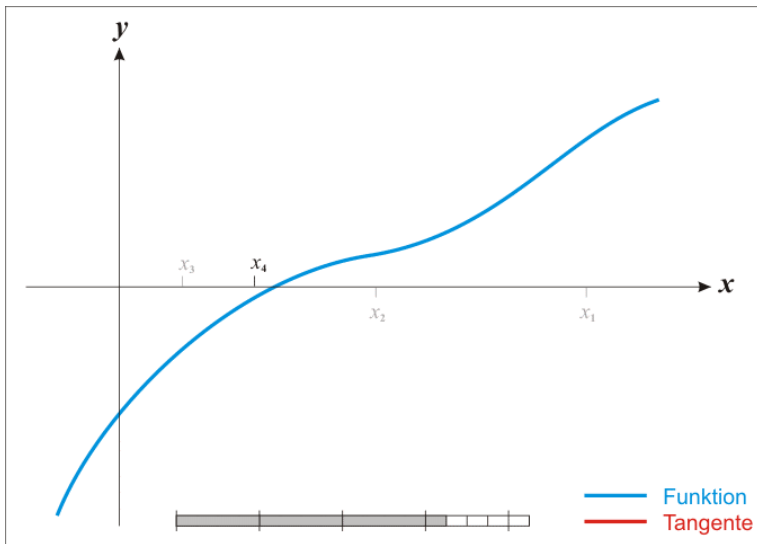
Copyright © 2005 Ralf Pfeifer / Creative Commons Attribution-ShareAlike 3.0

The Newton method (unidimensional)



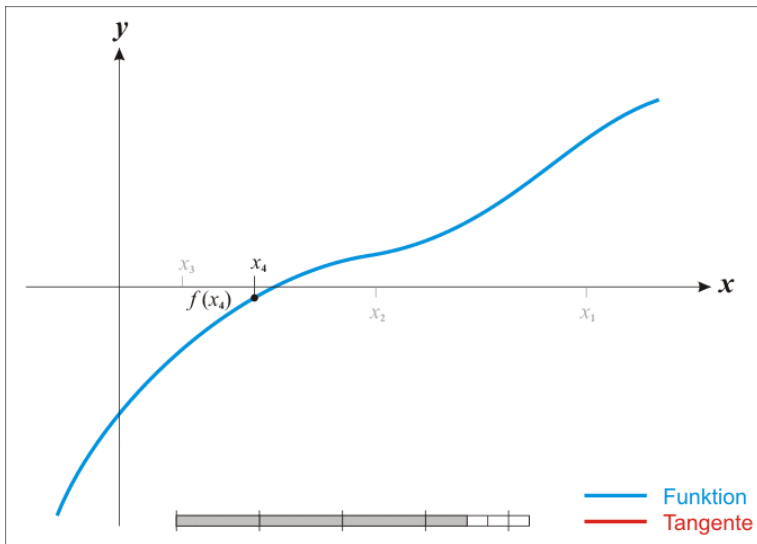
Copyright © 2005 Ralf Pfeifer / Creative Commons Attribution-ShareAlike 3.0

The Newton method (unidimensional)



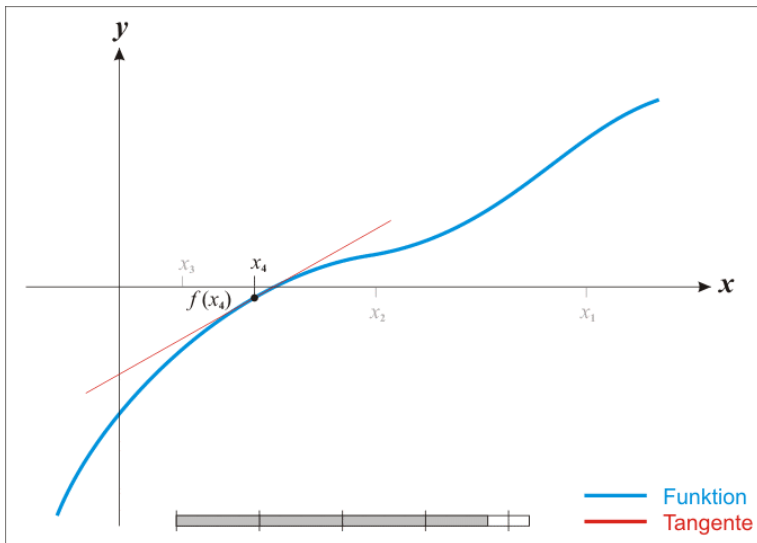
Copyright © 2005 Ralf Pfeifer / Creative Commons Attribution-ShareAlike 3.0

The Newton method (unidimensional)



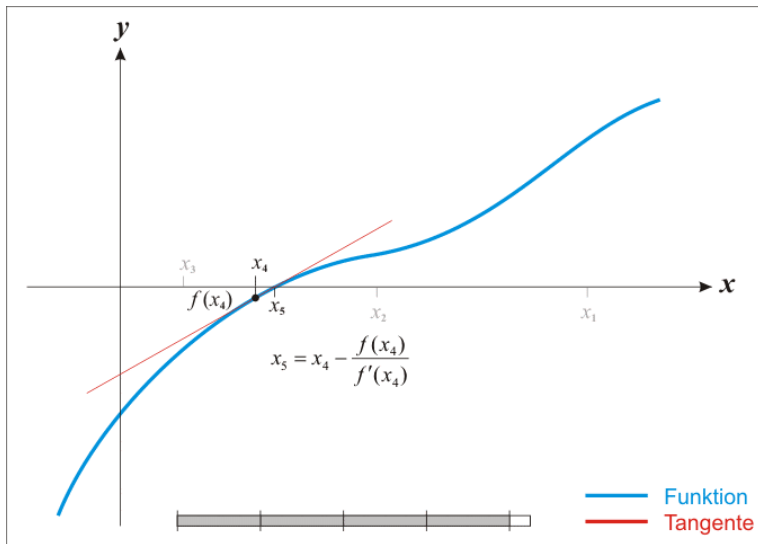
Copyright © 2005 Ralf Pfeifer / Creative Commons Attribution-ShareAlike 3.0

The Newton method (unidimensional)



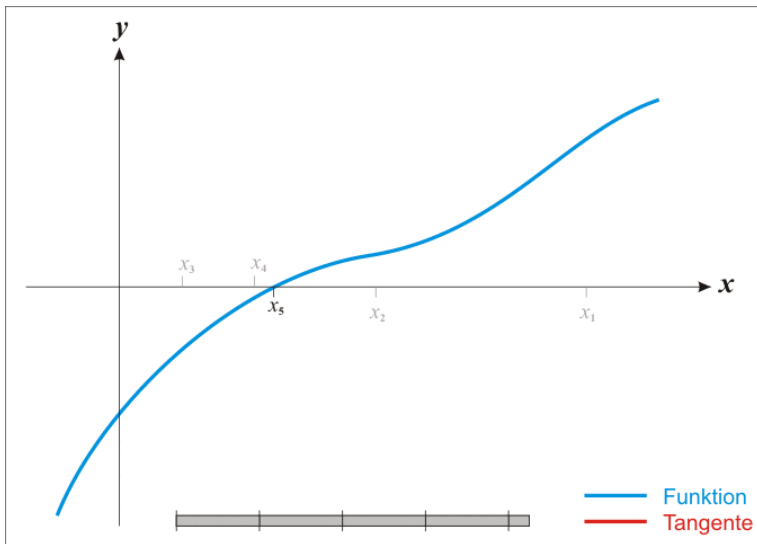
Copyright © 2005 Ralf Pfeifer / Creative Commons Attribution-ShareAlike 3.0

The Newton method (unidimensional)



Copyright © 2005 Ralf Pfeifer / Creative Commons Attribution-ShareAlike 3.0

The Newton method (unidimensional)



Copyright © 2005 Ralf Pfeifer / Creative Commons Attribution-ShareAlike 3.0

The Newton method (multidimensional)

- Start from an initial guess $Y^{(0)}$
- Iterate. Updated solutions $Y^{(k+1)}$ are obtained by solving a linear system:

$$F(Y^{(k)}) + \left[\frac{\partial F}{\partial Y} \right] (Y^{(k+1)} - Y^{(k)}) = 0$$

- Terminal condition:

$$\|Y^{(k+1)} - Y^{(k)}\| < \varepsilon_Y$$

or

$$\|F(Y^{(k)})\| < \varepsilon_F$$

- Convergence may never happen if function is ill-behaved or initial guess $Y^{(0)}$ too far from a solution
⇒ to avoid an infinite loop, abort after a given number of iterations

Controlling the Newton algorithm from Dynare

The following options to the `perfect_foresight_solver` can be used to control the Newton algorithm:

`maxit` Maximum number of iterations before aborting (default: 50)

`tolf` Convergence criterion based on function value (ε_F) (default: 10^{-5})

`tolx` Convergence criterion based on change in the function argument (ε_Y) (default: 10^{-5})

`stack_solve_algo` select between the different flavors of Newton algorithms (see thereafter)

A practical difficulty

The Jacobian can be very large: for a simulation over T periods of a model with n_y endogenous variables, it is a matrix of dimension $n_y T \times n_y T$.

Three alternative ways of dealing with the large problem size:

- Exploit the particular structure of the Jacobian using a technique developed by Laffargue, Boucekkine and Juillard (was the default method in Dynare ≤ 4.2)
- Handle the Jacobian as one large, sparse, matrix (now the default method)
- Block decomposition, which is a divide-and-conquer method (can actually be combined with one of the previous two methods)

Shape of the Jacobian

$$\frac{\partial F}{\partial Y} = \begin{pmatrix} B_1 & C_1 & & & & \\ A_2 & B_2 & C_2 & & & \\ & \ddots & \ddots & \ddots & & \\ & & A_t & B_t & C_t & \\ & & & \ddots & \ddots & \ddots \\ & & & & A_{T-1} & B_{T-1} & C_{T-1} \\ & & & & & A_T & B_T \end{pmatrix}$$

where

$$A_s = \frac{\partial f}{\partial y_{t-1}}(y_{s+1}, y_s, y_{s-1})$$

$$B_s = \frac{\partial f}{\partial y_t}(y_{s+1}, y_s, y_{s-1})$$

$$C_s = \frac{\partial f}{\partial y_{t+1}}(y_{s+1}, y_s, y_{s-1})$$

Laffargue-Boucekkine-Juillard algorithm (2/5)

First period is special:

$$\begin{pmatrix} I & D_1 & & & & & & \\ & B_2 - A_2 D_1 & C_2 & & & & & \\ & A_3 & B_3 & C_3 & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & A_{T-1} & B_{T-1} & C_{T-1} & & \\ & & & & A_T & B_T & & \end{pmatrix} \Delta Y = - \begin{pmatrix} d_1 \\ f(y_3, y_2, y_1, u_2) + A_2 d_1 \\ f(y_4, y_3, y_2, u_3) \\ \vdots \\ f(y_T, y_{T-1}, y_T, u_{T-1}) \\ f(y_{T+1}, y_T, y_{T-1}, u_T) \end{pmatrix}$$

where

- $D_1 = B_1^{-1} C_1$
- $d_1 = B_1^{-1} f(y_2, y_1, y_0, u_1)$

Laffargue-Boucekkine-Juillard algorithm (3/5)

Normal iteration of the triangularization:

$$\begin{pmatrix} I & D_1 & & & & & & \\ & I & D_2 & & & & & \\ & & B_3 - A_3 D_2 & C_3 & & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & A_{T-1} & B_{T-1} & C_{T-1} & \\ & & & & & A_T & B_T & \end{pmatrix} \Delta Y = - \begin{pmatrix} d_1 \\ d_2 \\ f(y_4, y_3, y_2, u_3) + A_3 d_2 \\ \vdots \\ f(y_T, y_{T-1}, y_T, u_{T-1}) \\ f(y_{T+1}, y_T, y_{T-1}, u_T) \end{pmatrix}$$

where

- $D_2 = (B_2 - A_2 D_1)^{-1} C_2$
- $d_2 = (B_2 - A_2 D_1)^{-1} (f(y_3, y_2, y_1, u_2) + A_2 d_1)$

Laffargue-Boucekkine-Juillard algorithm (4/5)

Final iteration:

$$\begin{pmatrix} I & D_1 & & & & & \\ & I & D_2 & & & & \\ & & I & D_3 & & & \\ & & & \ddots & \ddots & & \\ & & & & I & D_{T-1} & \\ & & & & & I & \\ & & & & & & \end{pmatrix} \Delta Y = - \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_{T-1} \\ d_T \end{pmatrix}$$

where

$$d_T = (B_T - A_T D_{T-1})^{-1} (f(y_{T+1}, y_T, y_{T-1}, u_T) + A_T d_{T-1})$$

Laffargue-Boucekkine-Juillard algorithm (5/5)

- The system is then solved by backward iteration:

$$\begin{aligned}y_T^{k+1} &= y_T^k - d_T \\y_{T-1}^{k+1} &= y_{T-1}^k - d_{T-1} - D_{T-1}(y_T^{k+1} - y_T^k) \\&\vdots \\y_1^{k+1} &= y_1^k - d_1 - D_1(y_2^{k+1} - y_2^k)\end{aligned}$$

- No need to ever store the whole Jacobian: only the D_s and d_s have to be stored
- This technique is memory efficient (was the default method in Dynare ≤ 4.2 for this reason)
- Still available as option `stack_solve_algo=1` of `perfect_foresight_solver` command

Sparse matrices (1/3)

- Consider the following matrix with most elements equal to zero:

$$A = \begin{pmatrix} 0 & 0 & 2.5 \\ -3 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

- Dense matrix storage (in column-major order) treats it as a one-dimensional array:

$$[0, -3, 0, 0, 0, 0, 2.5, 0, 0]$$

- Sparse matrix storage:

- ▶ views it as a list of triplets (i, j, v) where (i, j) is a matrix coordinate and v a non-zero value
- ▶ A would be stored as

$$\{(2, 1, -3), (1, 3, 2.5)\}$$

Sparse matrices (2/3)

- In the general case, given an $m \times n$ matrix with k non-zero elements:
 - ▶ dense matrix storage = $8mn$ bytes
 - ▶ sparse matrix storage = $16k$ bytes
 - ▶ sparse storage more memory-efficient as soon as $k < mn/2$(assuming 32-bit integers and 64-bit floating point numbers)
- In practice, sparse storage becomes interesting if $k \ll mn/2$, because linear algebra algorithms are vectorized

Sparse matrices (3/3)

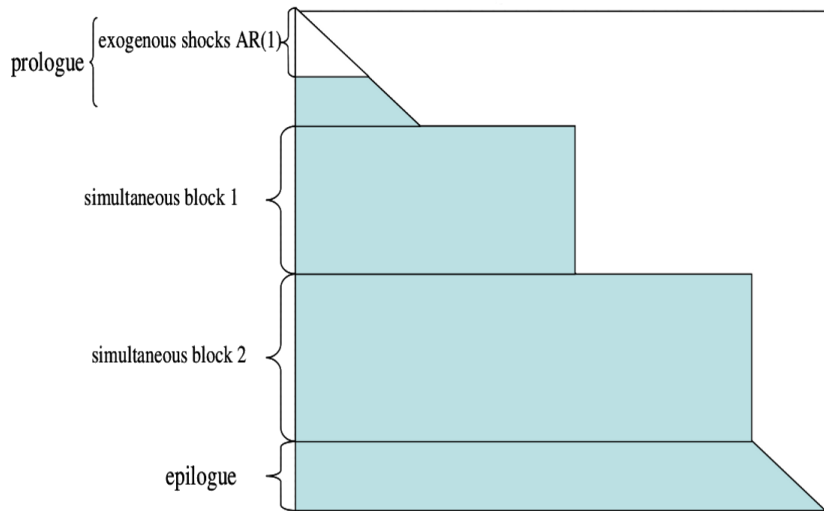
- The Jacobian of the deterministic problem is a sparse matrix:
 - ▶ Lots of zero blocks
 - ▶ The A_s , B_s and C_s usually are themselves sparse
- Family of optimized algorithms for sparse matrices (including matrix inversion for our Newton algorithm)
- Available as native objects in MATLAB/Octave (see the `sparse` command)
- Works well for medium size deterministic models
- Nowadays more efficient than Laffargue-Boucekkine-Juillard, even though it does not exploit the particular structure of the Jacobian
⇒ now the default method in Dynare (`stack_solve_algo=0`)

Block decomposition (1/3)

- Idea: apply a divide-and-conquer technique to model simulation
- Principle: identify recursive and simultaneous blocks in the model structure
- First block (prologue): equations that only involve variables determined by previous equations; example: AR(1) processes
- Last block (epilogue): pure output/reporting equations
- In between: simultaneous blocks, that depend recursively on each other
- The identification of the blocks is performed through a matching between variables and equations (normalization), then a reordering of both

Block decomposition (2/3)

Form of the reordered Jacobian (equations in lines, variables in columns)



Block decomposition (3/3)

- Can provide a significant speed-up on large models
- Implemented in Dynare by Ferhat Mihoubi
- Available as option `block` to the `model` command
- Bigger gains when used in conjunction with `bytecode` or `use_dll` option
- Can be combined with any flavour of the Newton method applied to each individual block

Homotopy

- Another divide-and-conquer method, but in the shocks dimension
- Useful if shocks so large that convergence does not occur
- Idea: achieve convergence on smaller shock size, then use the result as initial guess for bigger shock size
- Algorithm:
 - ① $\lambda \leftarrow 0$: scaling factor of shocks (simulation succeeds when $\lambda = 1$)
 - ② $s \leftarrow 1$: step size
 - ③ Try to compute simulation with shocks scaling factor equal to $\lambda + s$, using last successful computation as initial guess (in the beginning, use initial steady state at all t as initial guess)
 - ★ If success: $\lambda \leftarrow \lambda + s$. Stop if $\lambda = 1$. Otherwise possibly increase s .
 - ★ If failure: diminish s .
 - ④ Go to 3
- Works with both temporary and permanent shocks (*i.e.* shocks and endval)
- Can be combined with any deterministic solver
- Used by default by `perfect_foresight_solver` (disable with option `no_homotopy`)

Outline

- 1 Presentation of the problem
- 2 Solution techniques
- 3 Shocks: temporary/permanent, unexpected/pre-announced**
- 4 Occasionally binding constraints
- 5 More unexpected shocks
- 6 Extended path

Example: neoclassical growth model with investment

The social planner problem is as follows:

$$\max_{\{c_{t+j}, \ell_{t+j}, k_{t+j}\}_{j=0}^{\infty}} \sum_{j=0}^{\infty} \beta^j u(c_{t+j}, \ell_{t+j})$$

s.t.

$$y_t = c_t + i_t$$

$$y_t = A_t f(k_{t-1}, \ell_t)$$

$$k_t = i_t + (1 - \delta)k_{t-1}$$

$$A_t = A^* e^{a_t}$$

$$a_t = \rho a_{t-1} + \varepsilon_t$$

where ε_t is an exogenous shock and k_0 given.

Specifications

- Utility function:

$$u(c_t, l_t) = \frac{(c_t^\theta (1 - l_t)^{1-\theta})^{1-\tau}}{1 - \tau}$$

- Production function:

$$f(k_{t-1}, l_t) = (\alpha k_{t-1}^\psi + (1 - \alpha) l_t^\psi)^{\frac{1}{\psi}}$$

First order conditions

- Euler equation:

$$u_c(c_t, l_t) = \beta \left[u_c(c_{t+1}, l_{t+1}) (A_{t+1} f_k(k_t, l_{t+1}) + 1 - \delta) \right]$$

- Arbitrage between consumption and leisure:

$$\frac{u_\ell(c_t, l_t)}{u_c(c_t, l_t)} + A_t f_\ell(k_{t-1}, l_t) = 0$$

- Resource constraint:

$$c_t + k_t = A_t f(k_{t-1}, l_t) + (1 - \delta)k_{t-1}$$

Dynare code (1/3)

```
var k, y, L, c, A, a;  
varexo epsilon;  
parameters beta, theta, tau, alpha, psi, delta, rho, Astar;  
  
beta = 0.99;  
theta = 0.357;  
tau = 2;  
alpha = 0.45;  
psi = -0.1;  
delta = 0.02;  
rho = 0.8;  
Astar = 1;
```


Dynare code (2/3)

```
model;
  a = rho*a(-1) + epsilon;
  A = Astar*exp(a);
  y = A*(alpha*k(-1)^psi+(1-alpha)*L^psi)^(1/psi);
  k = y-c + (1-delta)*k(-1);
  (1-theta)/theta*c/(1-L) - (1-alpha)*(y/L)^(1-psi);
  (c^theta*(1-L)^(1-theta))^(1-tau)/c =
    beta*(c(+1)^theta*(1-L(+1))^(1-theta))^(1-tau)/c(+1)
    *(alpha*(y(+1)/k)^(1-psi)+1-delta);
end;
```

Dynare code (3/3)

```
steady_state_model;
  a = epsilon/(1-rho);
  A = Astar*exp(a);
  Output_per_unit_of_Capital=((1/beta-1+delta)/alpha)^(1/(1-psi));
  Consumption_per_unit_of_Capital=Output_per_unit_of_Capital-delta;
  Labour_per_unit_of_Capital=(((Output_per_unit_of_Capital/A)^psi-alpha)/(1-alpha))^(1/psi);
  Output_per_unit_of_Labour=Output_per_unit_of_Capital/Labour_per_unit_of_Capital;
  Consumption_per_unit_of_Labour=Consumption_per_unit_of_Capital/Labour_per_unit_of_Capital;

  % Compute steady state of the endogenous variables.
  L=1/(1+Consumption_per_unit_of_Labour/(((1-alpha)*theta/(1-theta)*Output_per_unit_of_Labour^(1-psi))));
  c=Consumption_per_unit_of_Labour*L;
  k=L/Labour_per_unit_of_Capital;
  y=Output_per_unit_of_Capital*k;
end;
```

Scenario 1: Return to equilibrium

Return to equilibrium starting from $k_0 = 0.5\bar{k}$.

Fragment from `rbc_det1.mod`

```
...
steady;

ik = varlist_indices('k',M_.endo_names);
kstar = oo_.steady_state(ik);

histval;
  k(0) = kstar/2;
end;

perfect_foresight_setup(periods=300);
perfect_foresight_solver;
```

Scenario 2: A temporary shock to TFP

- The economy starts from the steady state
- There is an unexpected negative shock at the beginning of period 1: $\varepsilon_1 = -0.1$

Fragment from `rbc_det2.mod`

```
...
steady;

shocks;
  var epsilon;
  periods 1;
  values -0.1;
end;

perfect_foresight_setup(periods=300);
perfect_foresight_solver;
```

Scenario 3: Pre-announced favorable shocks in the future

- The economy starts from the steady state
- There is a sequence of positive shocks to A_t : 4% in period 5 and an additional 1% during the 4 following periods

Fragment from `rbc_det3.mod`

```
...
steady;

shocks;
  var epsilon;
  periods 4, 5:8;
  values 0.04, 0.01;
end;

perfect_foresight_setup(periods=300);
perfect_foresight_solver;
```

Scenario 4: A permanent shock

- The economy starts from the initial steady state ($a_0 = 0$)
- In period 1, TFP increases by 5% permanently (and this was unexpected)

Fragment from `rbc_det4.mod`

```
...
initval;
  epsilon = 0;
end;

steady;

endval;
  epsilon = (1-rho)*log(1.05);
end;

steady;
```

Scenario 5: A pre-announced permanent shock

- The economy starts from the initial steady state ($a_0 = 0$)
- In period 6, TFP increases by 5% permanently
- A shocks block is used to maintain TFP at its initial level during periods 1–5

Fragment from `rbc_det5.mod`

```
...  
// Same initval and endval blocks as in Scenario 4  
...  
  
shocks;  
  var epsilon;  
  periods 1:5;  
  values 0;  
end;
```

Summary of commands

`initval` for the initial steady state (followed by `steady`)

`endval` for the terminal steady state (followed by `steady`)

`histval` for initial or terminal conditions out of steady state

`shocks` for shocks along the simulation path

`perfect_foresight_setup` prepare the simulation

`perfect_foresight_solver` compute the simulation

Under the hood

- The paths for exogenous and endogenous variables are stored in two MATLAB/Octave matrices:

$$\begin{aligned} \text{oo_endo_simul} &= \begin{pmatrix} y_0 & y_1 & \dots & y_T & y_{T+1} \\ \end{pmatrix} \\ \text{oo_exo_simul}' &= \begin{pmatrix} \boxtimes & u_1 & \dots & u_T & \boxtimes \\ \end{pmatrix} \end{aligned}$$

- `perfect_foresight_setup` initializes those matrices, given the shocks, `initval`, `endval` and `histval` blocks
 - ▶ y_0 , y_{T+1} and $u_1 \dots u_T$ are the constraints of the problem
 - ▶ $y_1 \dots y_T$ are the initial guess for the Newton algorithm
- `perfect_foresight_solver` replaces $y_1 \dots y_T$ in `oo_endo_simul` by the solution
- Notes:
 - ▶ for historical reasons, dates are in columns in `oo_endo_simul` and in lines in `oo_exo_simul`, hence the transpose (') above
 - ▶ this is the setup for no lead and no lag on exogenous
 - ▶ if one lead and/or one lag, u_0 and/or u_{T+1} would become relevant
 - ▶ if more than one lead and/or lag, matrices would be larger

Initial guess

The Newton algorithm needs an initial guess $Y^{(0)} = [y_1^{(0)'} \dots y_T^{(0)'}]$.

What is Dynare using for this?

- By default, if there is no `endval` block, it is the steady state as specified by `initval` (repeated for all simulations periods)
- Or, if there is an `endval` block, then it is the final steady state declared within this block
- Possibility of customizing this default by manipulating `oo_.endo_simul` after `perfect_foresight_setup` (but of course before `perfect_foresight_solver!`)
- If homotopy is triggered, the initial guess of subsequent iterations is the result of the previous iteration

Alternative way of specifying terminal conditions

- With the `differentiate_forward_vars` option of the `model` block, Dynare will substitute forward variables using new auxiliary variables:
 - ▶ Substitution: $x_{t+1} \rightarrow x_t + a_{t+1}$
 - ▶ New equation: $a_t = x_{t+1} - x_t$
- If the terminal condition is a steady state, the new auxiliary variables have obvious *zero* terminal condition
- Useful when:
 - ▶ the final steady state is hard to compute (this transformation actually provides a way to find it)
 - ▶ the model is very persistent and takes time to go back to steady state (this transformation avoids a kink at the end of the simulation if T is not large enough)

Outline

- 1 Presentation of the problem
- 2 Solution techniques
- 3 Shocks: temporary/permanent, unexpected/pre-announced
- 4 Occasionally binding constraints**
- 5 More unexpected shocks
- 6 Extended path

Zero nominal interest rate lower bound

- Implemented by writing the law of motion under the following form in Dynare:

$$i_t = \max \left\{ 0, (1 - \rho_i)i^* + \rho_i i_{t-1} + \rho_\pi (\pi_t - \pi^*) + \varepsilon_t^i \right\}$$

- *Warning*: this form will be accepted in a stochastic model, but the constraint will not be enforced in that case!

Irreversible investment

Same model as above, but the social planner is constrained to positive investment paths:

$$\max_{\{c_{t+j}, \ell_{t+j}, k_{t+j}\}_{j=0}^{\infty}} \sum_{j=0}^{\infty} \beta^j u(c_{t+j}, \ell_{t+j})$$

s.t.

$$y_t = c_t + i_t$$

$$y_t = A_t f(k_{t-1}, \ell_t)$$

$$k_t = i_t + (1 - \delta)k_{t-1}$$

$$i_t \geq 0$$

$$A_t = A^* e^{a_t}$$

$$a_t = \rho a_{t-1} + \varepsilon_t$$

where the technology (f) and the preferences (u) are as above.

First order conditions

$$u_c(c_t, l_t) - \mu_t = \beta [u_c(c_{t+1}, l_{t+1}) (A_{t+1} f_k(k_t, l_{t+1}) + 1 - \delta) - \mu_{t+1} (1 - \delta)]$$
$$\frac{u_\ell(c_t, l_t)}{u_c(c_t, l_t)} + A_t f_l(k_{t-1}, l_t) = 0$$
$$c_t + k_t = A_t f(k_{t-1}, l_t) + (1 - \delta) k_{t-1}$$

Complementarity condition:

$$\mu_t = 0 \text{ and } i_t \geq 0$$

or

$$\mu_t > 0 \text{ and } i_t = 0$$

where $\mu_t \geq 0$ is the Lagrange multiplier associated to the non-negativity constraint for investment.

Mixed complementarity problems

- A mixed complementarity problem (MCP) is given by:
 - ▶ function $F(x) : \mathbb{R}^n \rightarrow \mathbb{R}^n$
 - ▶ lower bounds $\ell_i \in \mathbb{R} \cup \{-\infty\}$
 - ▶ upper bounds $u_i \in \mathbb{R} \cup \{+\infty\}$
- A solution of the MCP is a vector $x \in \mathbb{R}^n$ such that for each $i \in \{1 \dots n\}$, one of the following alternatives holds:
 - ▶ $\ell_i < x_i < u_i$ and $F_i(x) = 0$
 - ▶ $x_i = \ell_i$ and $F_i(x) \geq 0$
 - ▶ $x_i = u_i$ and $F_i(x) \leq 0$

- Notation:

$$F(x) \perp \ell \leq x \leq u$$

- Solving a square system of nonlinear equations is a particular case (with $\ell_i = -\infty$ and $u_i = +\infty$ for all i)
- Optimality problems with inequality constraints are naturally expressed as MCPs (finite bounds are imposed on Lagrange multipliers)

The irreversible investment model in Dynare

- MCP solver triggered with option `lmmcp` of `perfect_foresight_solver`
- Complementarity condition described by equation tag `mcp`

Fragment from `rbcii.mod`

```
(c^theta*(1-L)^(1-theta))^(1-tau)/c - mu =  
    beta*((c(+1)^theta*(1-L(+1))^(1-theta))^(1-tau)/c(+1)  
        *(alpha*(y(+1)/k)^(1-psi)+1-delta)-mu(+1)*(1-delta));
```

```
...  
[ mcp = 'i > 0' ]  
mu = 0;  
...  
perfect_foresight_setup(periods=400);  
perfect_foresight_solver(lmmcp, maxit=200);
```

OccBin (1/2)

- Piecewise linear approach of Guerrieri and Iacoviello (JME, 2015)
- Under certainty equivalence; but quite fast, works on large models
- Relies on a solution under perturbation and not a deterministic solver, hence nonlinearities other than regime change are not taken into account

Example

```
model;
  [name='Notional rate Taylor rule']
  i_not=rho*i_not(-1)+rho*(phi_pi*pi+phi_y*y)+zeps_i;
  [name='Observed interest rate', relax='zlb']
  i = i_not;
  [name='Observed interest rate', bind='zlb']
  i = i_elb;
  ...
end;
```

OccBin (2/2)

Example (cont'd)

```
occbin_constraints;  
    name 'zlb'; bind i_not <= i_elb;  
end;  
  
shocks(surprise);  
    var zeps_i;  
    periods 1 2;  
    values -0.01 -0.02;  
end;  
  
occbin_setup;  
occbin_solver(simul_periods=20, simul_check_ahead_periods=50);  
occbin_graph y i i_not pie;
```

Outline

- 1 Presentation of the problem
- 2 Solution techniques
- 3 Shocks: temporary/permanent, unexpected/pre-announced
- 4 Occasionally binding constraints
- 5 More unexpected shocks**
- 6 Extended path

Simulating unexpected shocks

With a perfect foresight solver:

- shocks are unexpected in period 1
- but in subsequent periods they are anticipated

How to simulate an unexpected shock at a period $t > 1$?

- Do a perfect foresight simulation from periods 0 to T *without the unexpected shock in t* (but with other expected shocks)
- Do another perfect foresight simulation from periods t to T
 - ▶ applying the unexpected shock in t (and keeping expected shocks),
 - ▶ and using the results of the first simulation as initial condition
- Combine the two simulations:
 - ▶ use the first one for periods 1 to $t - 1$,
 - ▶ and the second one for t to T

A Dynare example

Simulation of a scenario with:

- Pre-announced (negative) shocks in periods 5 and 15
- Unexpected (positive) shock in period 10

From `rbc_unexpected.mod`:

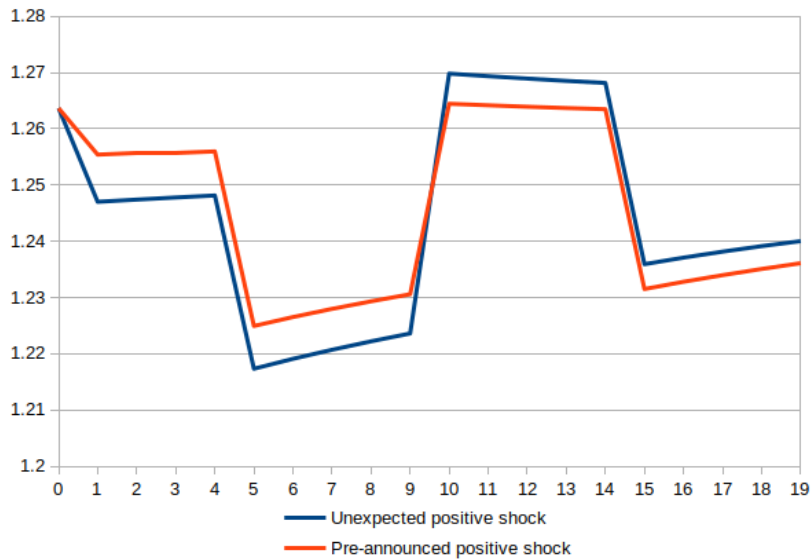
```
...  
// Declare pre-announced shocks  
shocks(learnt_in=1);  
    var epsilon;  
    periods 5, 15;  
    values -0.1, -0.1;  
end;
```

A Dynare example (continued)

```
// Declare shocks learnt in period 10
shocks(learnt_in=10);
    var epsilon;
    periods 10;
    values 0.1;
end;

perfect_foresight_with_expectation_errors_setup(periods=300);
perfect_foresight_with_expectation_errors_solver;
```

Consumption path



Additional details

- The `perfect_foresight_with_expectation_errors_solver` command automatically runs the required deterministic simulations and combines the paths to get the solution
- More complex scenarios are possible, where agents learn in period $t > 1$ about shock(s) in periods $t + s$ for $s > 0$ (*i.e.* pre-announced shocks, but which are learnt in some period > 1)
- In the context of permanent shocks, information can also be learnt about terminal conditions, using the `endval(learnt_in=...)` block

Outline

- 1 Presentation of the problem
- 2 Solution techniques
- 3 Shocks: temporary/permanent, unexpected/pre-announced
- 4 Occasionally binding constraints
- 5 More unexpected shocks
- 6 Extended path**

Extended path (EP) algorithm

- Idea: use the previous method to simulate a *rational expectations* (RE) model (*i.e.* with stochastic shocks that can happen at every period)...
- ... but under the simplifying assumption that agents believe that the economy will not be perturbed in the future (all future shocks will be at their steady state value $\bar{u} = 0$), and do not update their belief when observing shocks (hence we are not solving the true RE model, but an approximation of it)
- Advantage: *deterministic* nonlinearities fully taken into account
- Inconvenient: solution under certainty equivalence (Jensen inequality is violated). For example, no precautionary motive.
- When the goal is to generate a timeseries, this method is strictly superior to first-order perturbation (which is also under certainty equivalence, but does not take into account deterministic nonlinearities)
- Method introduced by Fair and Taylor (1983)

Extended path (EP) algorithm (continued)

- Algorithm

- 1 $H \leftarrow$ Set the horizon of the perfect foresight (PF) model
- 2 $(\bar{y}, \bar{u}) \leftarrow$ Compute steady state of the model
- 3 $y_0 \leftarrow$ Choose an initial condition for the endogenous variables
- 4 for $t = 1$ to T
- 5 $u_t \leftarrow$ Draw random shocks for the current period
- 6 $y_t \leftarrow$ Solve a PF with:
 - ★ Initial condition: y_{t-1} computed in previous iteration
 - ★ Terminal condition: $y_{t+H} = \bar{y}$
 - ★ Shocks: u_t just drawn and $u_{t+s} = \bar{u}$ (for $s > 0$)
- 7 end for

- Implemented under the command `extended_path` (with option `order=0`, which is the default)
- Option `periods` controls T
- Option `solver_periods` controls H (defaults to 200)

Extended path: Dynare example

From `rbc_ep.mod`:

```
...  
// Declare shocks as in a stochastic setup  
shocks;  
    var epsilon;  
    stderr 0.02;  
end;  
  
extended_path(periods=300);  
  
// Plot 20 first periods of consumption  
ic = varlist_indices('c', M_.endo_names);  
plot(oo_.endo_simul(ic, 1:21));
```

Stochastic extended path (SEP)

- Idea: generalize the extended path method to take into account *some* future uncertainty
- Approximation: at date t ,
 - ▶ agents know that there will be future stochastic shocks in periods $t + 1$ to $t + k$
 - ▶ but they assume that there will be no more shocks in periods $> t + k$
- k measures the degree of future uncertainty taken into account
- We are still not solving the true RE model, but the larger k , the closer we are to the true model (which corresponds to $k = +\infty$).
- Note: (plain) extended path as presented in the previous slides corresponds to $k = 0$
- Additional approximation: the probability distribution about future uncertainty is simplified using discrete numerical integration (*a.k.a.* quadrature)

Gauss-Hermite quadrature (univariate)

- Let X be a Gaussian random variable with mean zero and variance $\sigma_x^2 > 0$, and suppose that we need to evaluate $\mathbb{E}[\varphi(X)]$, where φ is a continuous function
- By definition we have:

$$\mathbb{E}[\varphi(X)] = \frac{1}{\sigma_x \sqrt{2\pi}} \int_{-\infty}^{\infty} \varphi(x) e^{-\frac{x^2}{2\sigma_x^2}} dx$$

- This integral can be approximated by a finite sum using the following result (Gauss-Hermite quadrature formula at order n):

$$\int_{-\infty}^{\infty} \varphi(z) e^{-z^2} dz \approx \sum_{i=1}^n \omega_i \varphi(z_i)$$

where z_i ($i = 1, \dots, n$) are the roots of an order n Hermite polynomial, and the weights ω_i are positive and summing up to one (variable change: $x_i = \frac{z_i}{\sigma_x \sqrt{2}}$)

- The higher n , the better the approximation

Gauss-Hermite quadrature (multivariate)

- Let X be a multivariate Gaussian random variable with mean zero and unit variance, and suppose that we need to evaluate

$$\mathbb{E}[\varphi(X)] = (2\pi)^{-\frac{p}{2}} \int_{\mathbb{R}^p} \varphi(\mathbf{x}) e^{-\frac{1}{2}\mathbf{x}'\mathbf{x}} d\mathbf{x}$$

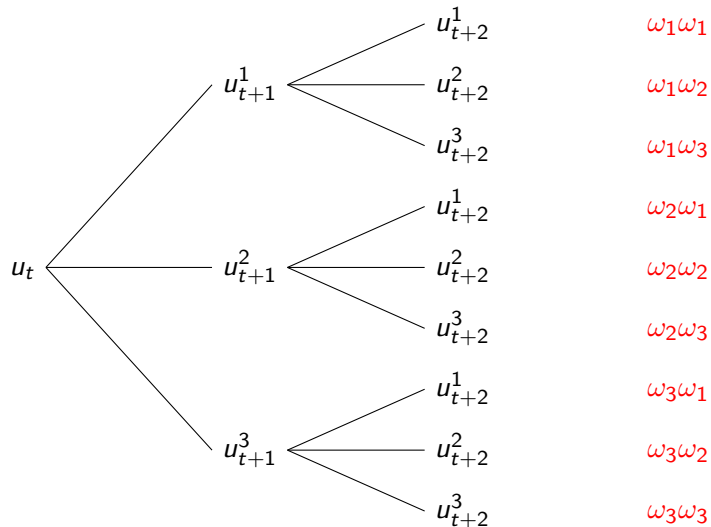
- Let $\{(\omega_i, z_i)\}_{i=1}^n$ be the weights and nodes of an order n univariate Gauss-Hermite quadrature
- This integral can be approximated using a tensor grid:

$$\int_{\mathbb{R}^p} \varphi(\mathbf{z}) e^{-\mathbf{z}'\mathbf{z}} d\mathbf{z} \approx \sum_{i_1, \dots, i_p=1}^n \omega_{i_1} \dots \omega_{i_p} \varphi(z_{i_1}, \dots, z_{i_p})$$

- Curse of dimensionality:** The number of terms in the sum grows exponentially with the number of shocks.

Forward history

One shock, three quadrature nodes, order two SEP ($k = 2$)



Stochastic extended path (SEP) (continued)

- Algorithm similar to extended path (EP), except that instead of solving a perfect foresight (PF) problem for each period, a larger problem is solved:
 - ▶ equations determining future variables are replicated as many times as there are branches on the tree of history;
 - ▶ Gauss-Hermite quadratures are used to compute expectations against those future variables.
- We face two curses of dimensionality (exponential complexity growth):
 - ▶ Stochastic order (k)
 - ▶ Number of shocks
- In practice, only feasible for small k and small number of shocks
- SEP triggered with option `order=k` of `extended_path` command
- NB: currently no interface for controlling the number of nodes for the Gauss-Hermite quadrature. The number of nodes has to be directly set in the `options_global` structure

Thanks for your attention!
Questions?

My email: `sebastien@dynare.org`



Copyright © 2015-2024 Dynare Team
License: Creative Commons Attribution-ShareAlike 4.0