# Deterministic Models

Perfect foresight, nonlinearities and occasionally binding constraints

Sébastien Villemot

CEPREMAP
CENTRE POUR LA RECHERCHE ECONOMIQUE ET SES APPLICATIONS

June 12, 2018

## 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
      $\Rightarrow$ the model behaves as if it was 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

## Outline

## 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:

$$\begin{aligned} c_t^{-\sigma} &= \beta c_{t+1}^{-\sigma}\left(\alpha A_{t+1} k_t^{\alpha-1} + 1 - \delta\right) \\ c_t + k_t &= A_t k_{t-1}^{\alpha} + (1-\delta)k_{t-1} \end{aligned}$$

Steady state:

$$\begin{aligned} \bar{k} &= \left(\frac{1-\beta(1-\delta)}{\beta\alpha\bar{A}}\right)^{\frac{1}{\alpha-1}} \\ \bar{c} &= \bar{A}\bar{k}^{\alpha} - \delta\bar{k} \end{aligned}$$

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

```
// Steady state (analytically solved)
initval;
 A = 1;
 k = ((1-beta*(1-delta))/(beta*alpha*A))^(1/(alpha-1));
 c = A*k^alpha-delta*k;
end;

// Check that this is indeed the steady state
steady;
```

## Dynare code (3/3)

rcb_basic.mod

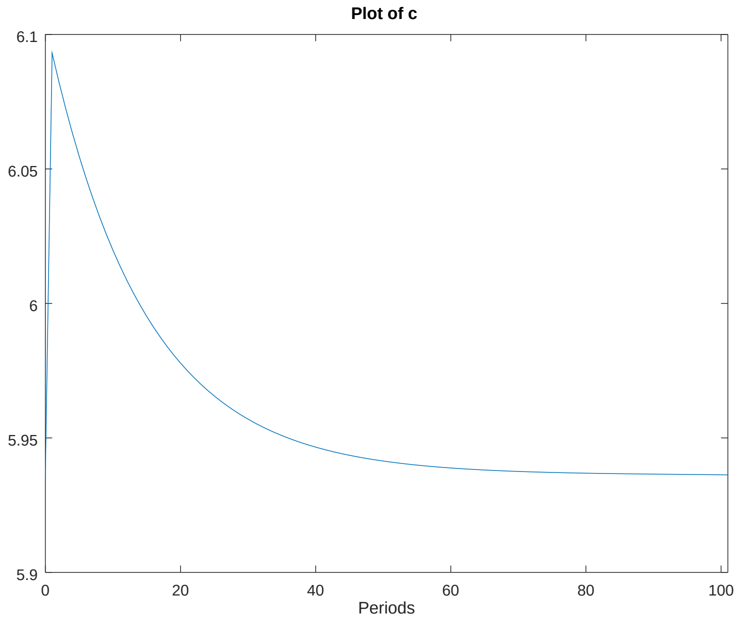
```
// Declare a positive technological shock in period 1
shocks;
 var A;
 periods 1;
 values 1.2;
end;

// 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

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

$$u_t = A_t$$

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

## 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

## 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
- The steady state is computed by Dynare with the `steady` command
- That command internally uses a nonlinear solver

## A two-boundary value problem

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

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

  for $y_0$ and $y_{T+1}$ given.

- Compact representation:

$$F(Y) = 0$$

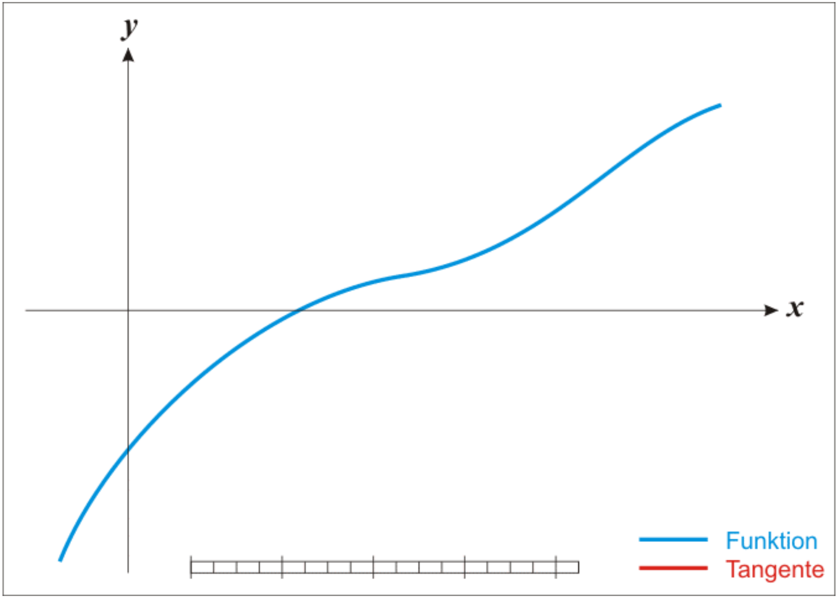
  where $Y = \begin{bmatrix} y_1' & y_2' & \dots & y_T' \end{bmatrix}'$
  and $y_0$, $y_{T+1}$, $u_1 \dots u_T$ are implicit

## Solution of perfect foresight models

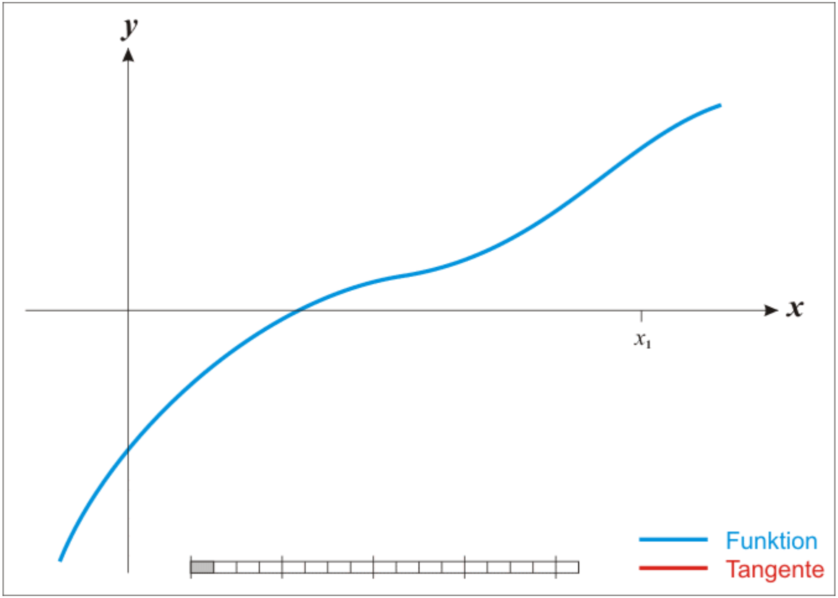
- This technique numerically computes trajectories for given shocks over a finite number of periods
- No possibility of computing a recursive policy function (as with perturbation methods), because future shock paths are state variables, and those are infinite-dimensional objects
- However, it is possible to approximate the asymptotic return to equilibrium with $y_{T+1} = \bar{y}$ and a large enough $T$
- Resolution uses a Newton-type method on the stacked system

## Outline

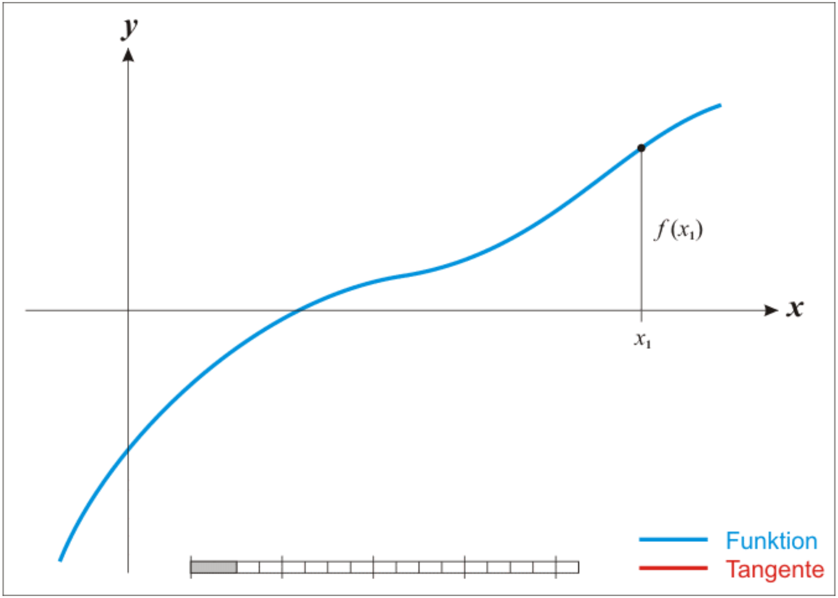
## The Newton method (unidimensional)



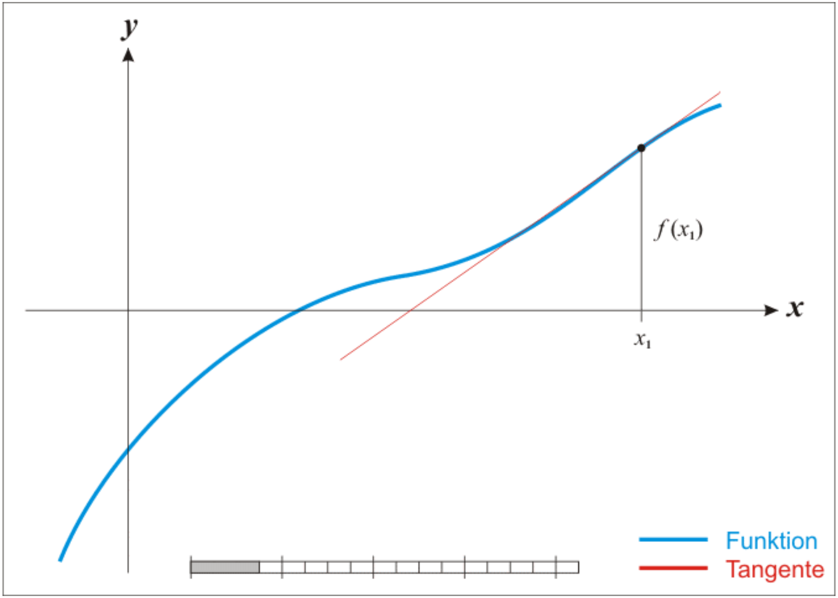
## The Newton method (unidimensional)



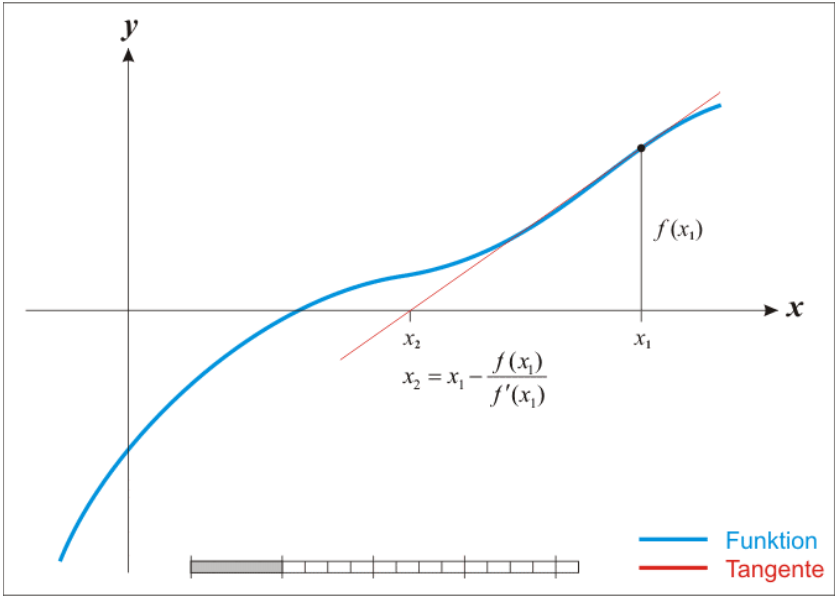
## The Newton method (unidimensional)



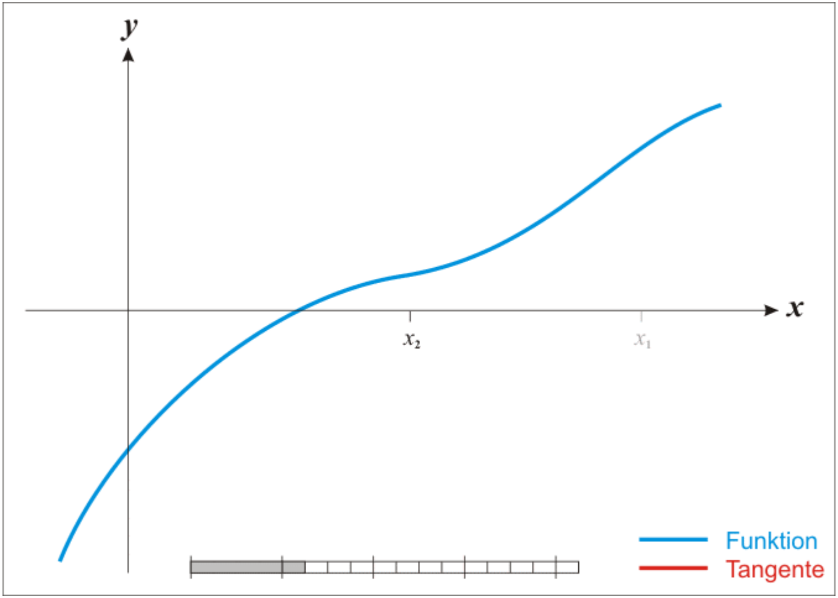
## The Newton method (unidimensional)



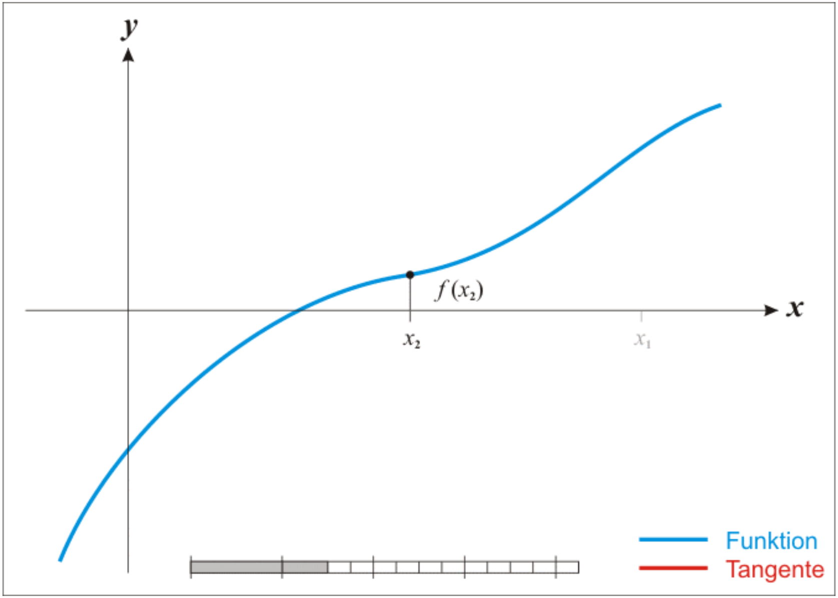
## The Newton method (unidimensional)



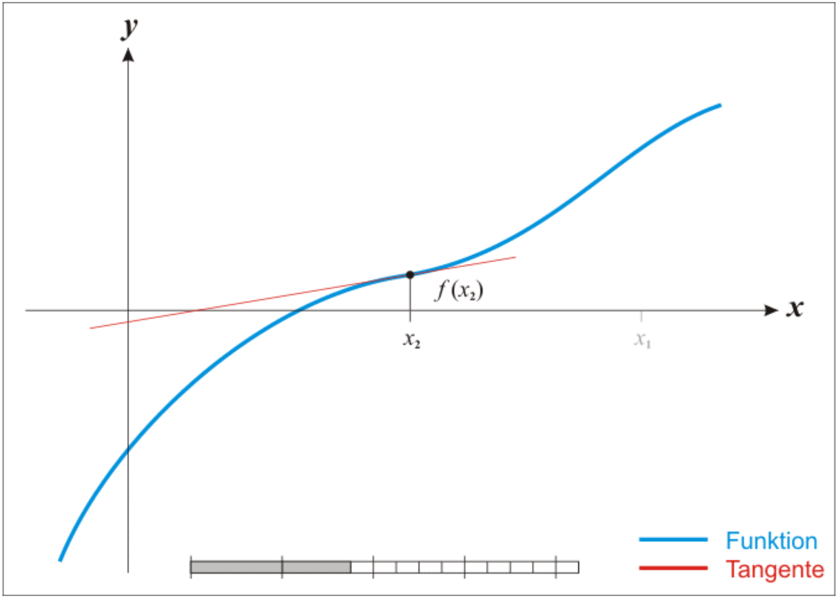
## The Newton method (unidimensional)



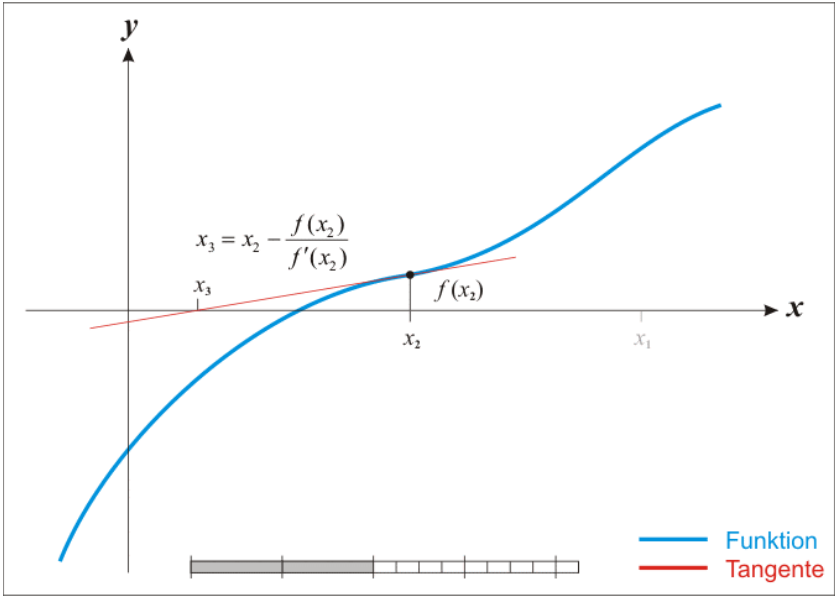
# The Newton method (unidimensional)



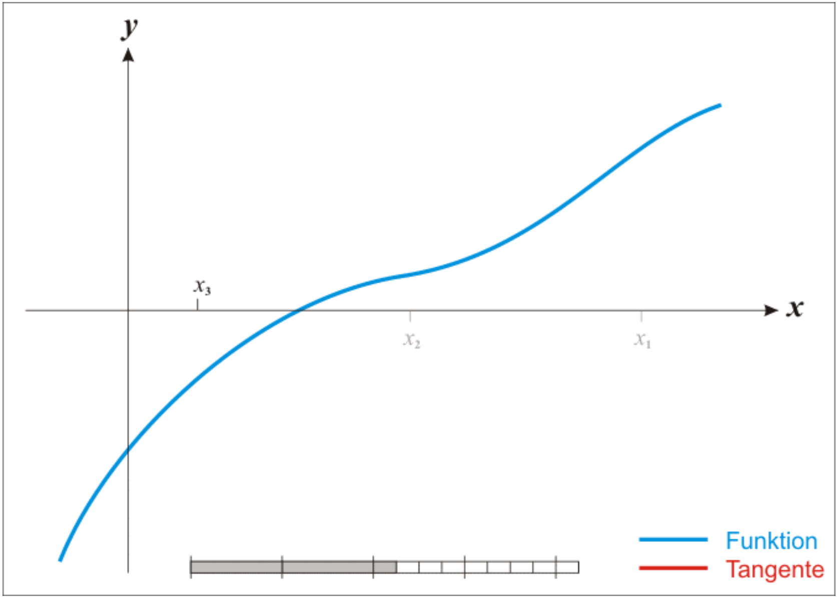
## The Newton method (unidimensional)



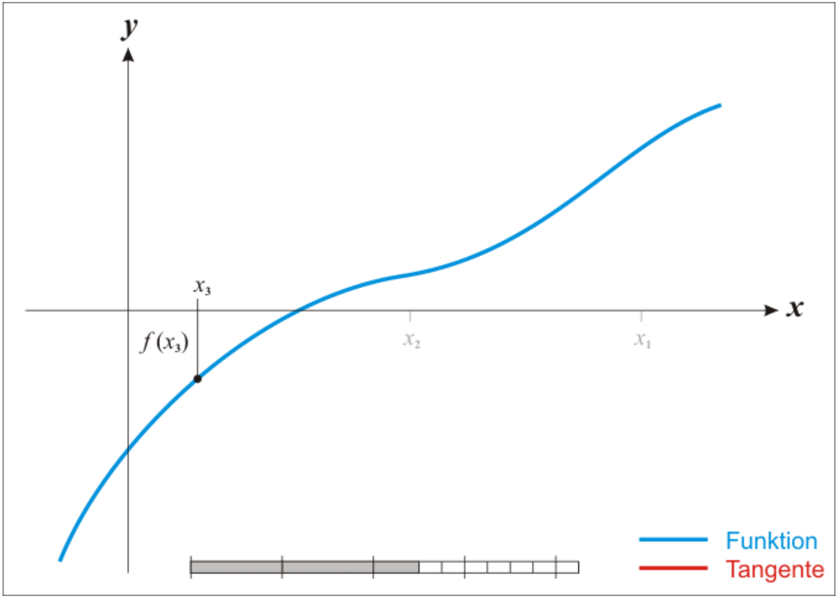
# The Newton method (unidimensional)



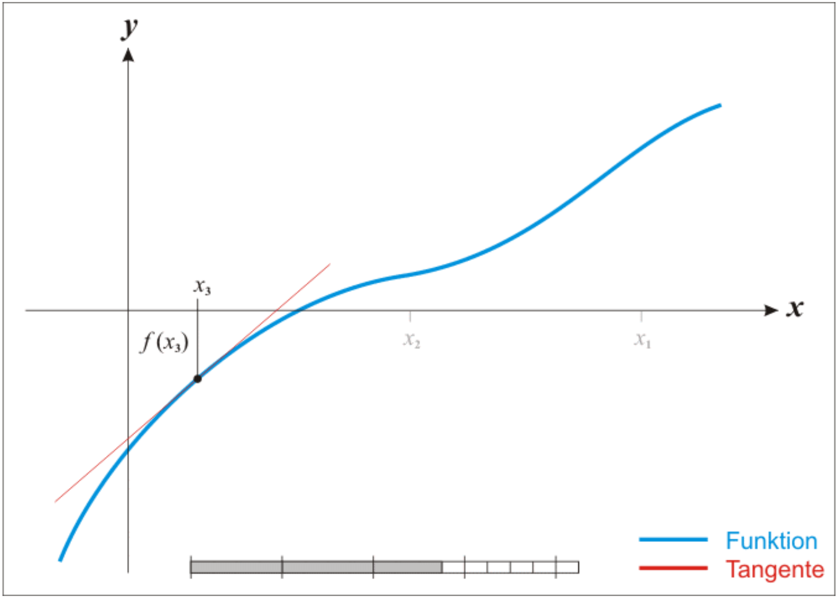
## The Newton method (unidimensional)



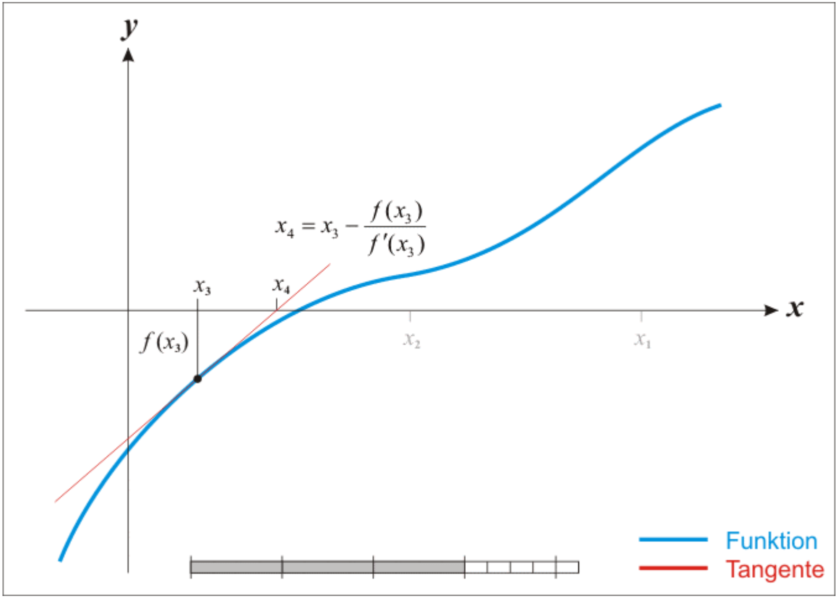
## The Newton method (unidimensional)



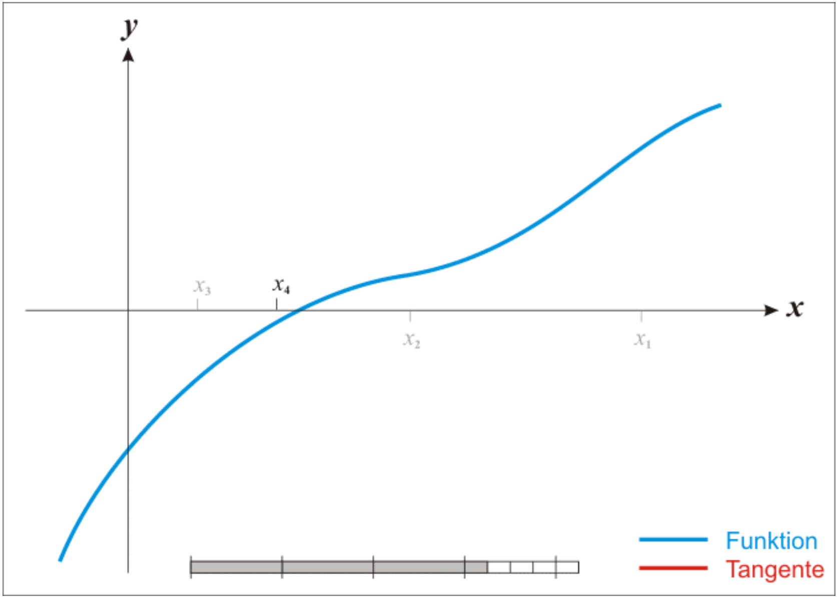
## The Newton method (unidimensional)



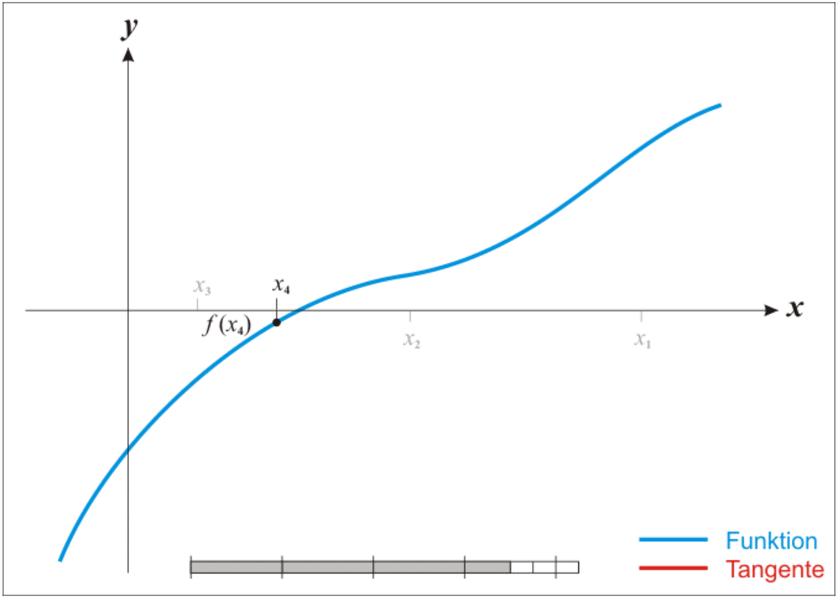
## The Newton method (unidimensional)



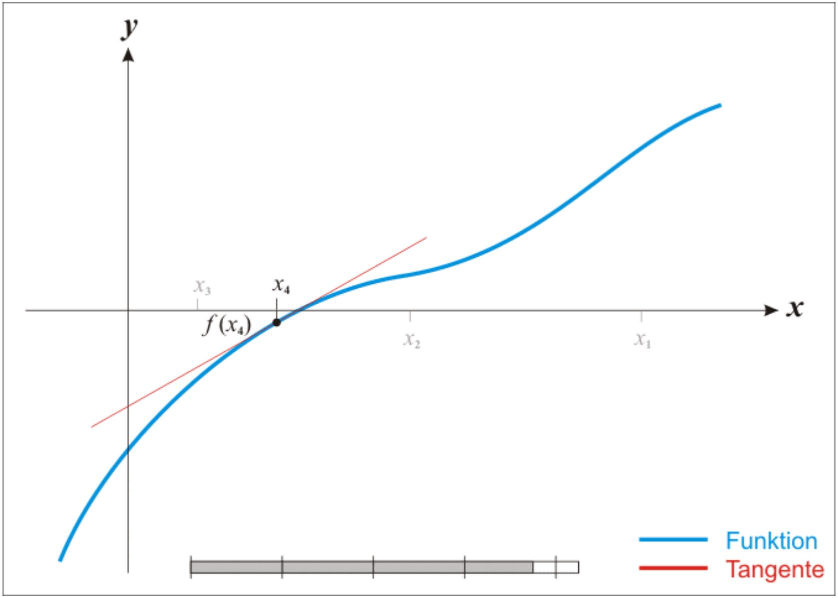
# The Newton method (unidimensional)



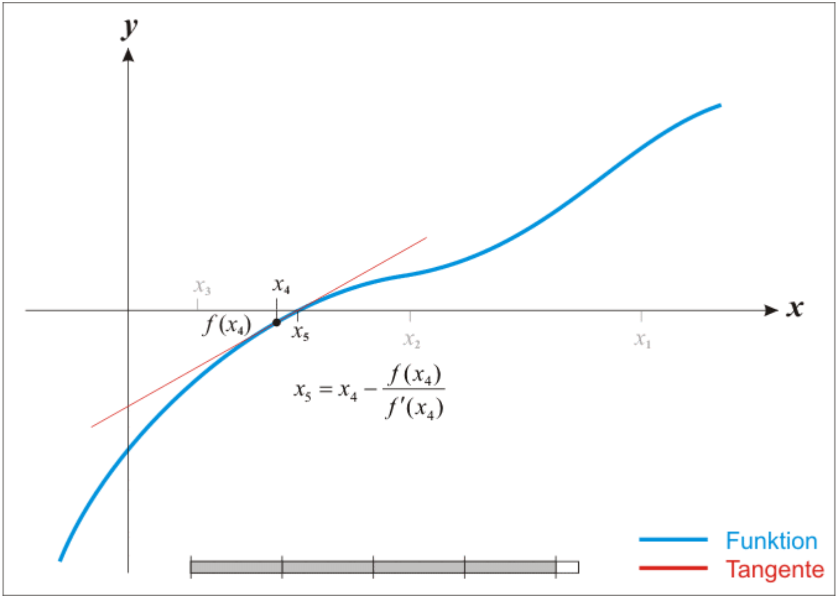
## The Newton method (unidimensional)



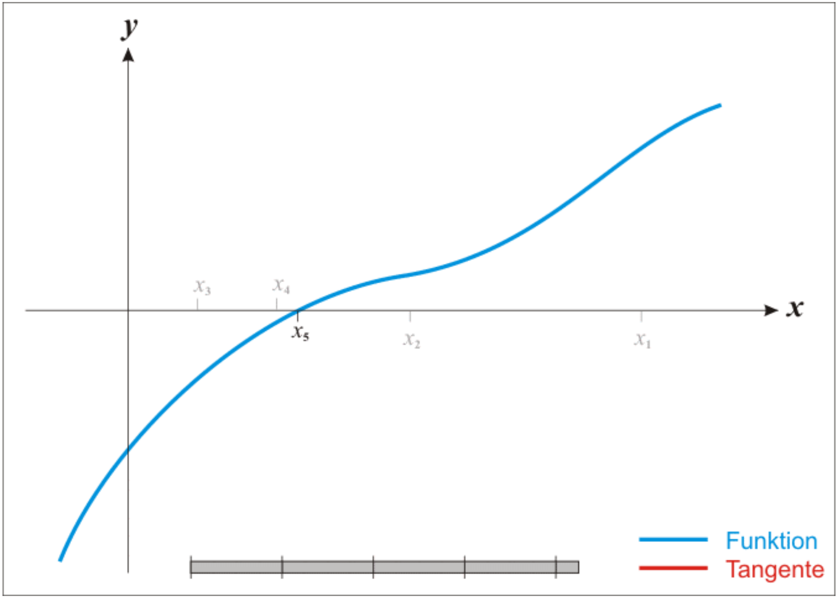
## The Newton method (unidimensional)



## The Newton method (unidimensional)



## The Newton method (unidimensional)



## The Newton method (multidimensional)

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

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

- 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
  $\Rightarrow$ 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 ($\varepsilon_F$) (default: $10^{-5}$)
- `tolx` Convergence criterion based on change in the function argument ($\varepsilon_Y$) (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 $n$ endogenous variables, it is a matrix of dimension $nT \times nT$.

Three alternative ways of dealing with the large problem size:

- Exploit the particular structure of the Jacobian using a relaxation technique developped by Laffargue, Boucekkine and Juillard (was the default method in Dynare $\leq$ 4.2)
- Handle the Jacobian as one large, sparse, matrix (now the default method)
- Block decomposition, which is a divide-and-conquer method, implemented by Mihoubi

## 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

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

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

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

## Relaxation (1/5)

The idea is to triangularize the stacked system:

$$\begin{pmatrix} B_1 & C_1 & & & & \\ A_2 & B_2 & C_2 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & A_{T-1} & B_{T-1} & C_{T-1} \\ & & & & A_T & B_T \end{pmatrix} \Delta Y = - \begin{pmatrix} f(y_2, y_1, y_0, u_1) \\ f(y_3, y_2, y_1, u_2) \\ \vdots \\ \vdots \\ f(y_T, y_{T-1}, y_T, u_{T-1}) \\ f(y_{T+1}, y_T, y_{T-1}, u_T) \end{pmatrix}$$

## Relaxation (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

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

## Relaxation (3/5)

Normal iteration:

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

where

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

## Relaxation (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

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

## Relaxation (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 $\leq$ 4.2 for this reason)
- Still available as option `stack_solve_algo=6` 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)

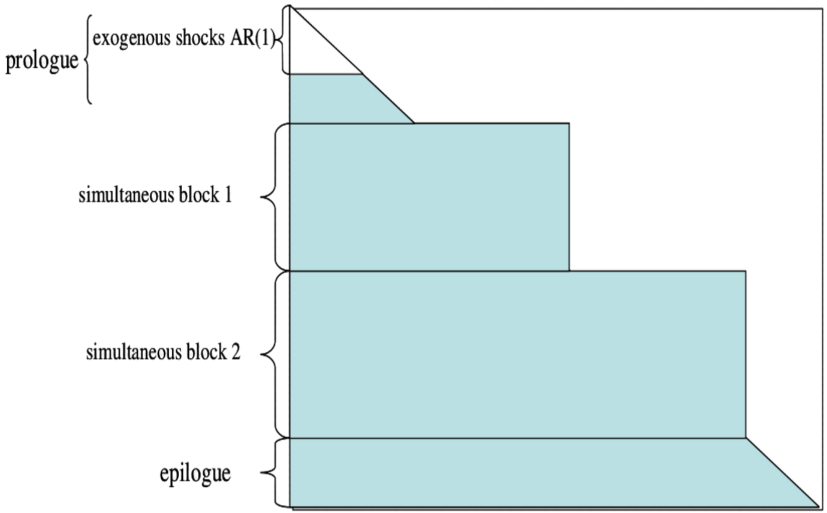
- The Jacobian of the deterministic problem is a sparse matrix:
  - Lots of zero blocks
  - The $A_s$, $B_s$ and $C_s$ usually are themselves sparse
- Family of optimized algorithms for sparse matrices (including matrix inversion for our Newton algorithm)
- Available as native objects in MATLAB/Octave (see the `sparse` command)
- Works well for medium size deterministic models
- Nowadays more efficient than relaxation, even though it does not exploit the particular structure of the Jacobian
  $\Rightarrow$ 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` option

## 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 starting point for bigger shock size
- Algorithm:
  1. Starting point for simulation path: steady state at all $t$
  2. $\lambda \leftarrow 0$: scaling factor of shocks (simulation succeeds when $\lambda = 1$)
  3. $s \leftarrow 1$: step size
  4. Try to compute simulation with shocks scaling factor equal to $\lambda + s$ (using last successful computation as starting point)
     - ★ If success: $\lambda \leftarrow \lambda + s$. Stop if $\lambda = 1$. Otherwise possibly increase $s$.
     - ★ If failure: diminish $s$.
  5. Go to 4
- Can be combined with any deterministic solver
- Used by default by `perfect_foresight_solver` (can be disabled with option `no_homotopy`)

## Outline

## Example: neoclassical growth model with investment

The social planner problem is as follows:

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

s.t.

$$\begin{aligned} y_t &= c_t + i_t \\ y_t &= A_t f(k_{t-1}, \ell_t) \\ k_t &= i_t + (1-\delta)k_{t-1} \\ A_t &= A^{\star} e^{a_t} \\ a_t &= \rho\, a_{t-1} + \varepsilon_t \end{aligned}$$

where $\varepsilon_t$ is an exogenous shock.

## Specifications

- Utility function:

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

- Production function:

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

## First order conditions

- Euler equation:

$$u_c(c_t, \ell_t) = \beta\, \mathbb{E}_t \left[ u_c(c_{t+1}, \ell_{t+1}) \Big( A_{t+1} f_k(k_t, \ell_{t+1}) + 1 - \delta \Big) \right]$$

- Arbitrage between consumption and leisure:

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

- Resource constraint:

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

## Dynare code (1/3)

```
var k, y, L, c, A, a;
varexo epsilon;
parameters beta, theta, tau, alpha, psi, delta, rho, Astar;

beta    =   0.9900;
theta   =   0.3570;
tau     =   2.0000;
alpha   =   0.4500;
psi     =  -0.1000;
delta   =   0.0200;
rho     =   0.8000;
Astar   =   1.0000;
```

## 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 |
| `simul` | do both operations at the same time (deprecated syntax, alias for `perfect_foresight_setup` + `perfect_foresight_solver`) |

## Under the hood

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

$$\begin{array}{lcl} \texttt{oo\_.endo\_simul} & = & \begin{pmatrix} y_0 & y_1 & \dots & y_T & y_{T+1} \end{pmatrix} \\ \texttt{oo\_.exo\_simul'} & = & \begin{pmatrix} \boxtimes & u_1 & \dots & u_T & \boxtimes \end{pmatrix} \end{array}$$

- `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 $Y^{(0)} = [y_1^{(0)'} \; \ldots \; y_T^{(0)'}]$.

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

## 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.

$$\begin{gathered} y_t = c_t + i_t \\ y_t = A_t f(k_{t-1}, \ell_t) \\ k_t = i_t + (1-\delta)k_{t-1} \\ i_t \geq 0 \\ A_t = A^\star e^{a_t} \\ a_t = \rho\, a_{t-1} + \varepsilon_t \end{gathered}$$

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

## First order conditions

$$u_c(c_t,\ell_t) - \mu_t = \beta\,\mathbb{E}_t\big[u_c(c_{t+1},\ell_{t+1})\,(A_{t+1}f_k(k_t,\ell_{t+1}) + 1 - \delta) \\ - \mu_{t+1}(1-\delta)\big]$$

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

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

Slackness condition:

$$\mu_t = 0 \;\; \texttt{and} \;\; i_t \geq 0$$

or

$$\mu_t > 0 \;\; \texttt{and} \;\; i_t = 0$$

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 $F(x) : \mathbb{R}^n \to \mathbb{R}^n$
  - 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:

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

- 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`
- Slackness 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);
```

## Outline

## 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 shock*
- Do another perfect foresight simulation from periods $t$ to $T$
  - applying the shock in $t$,
  - 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 (1/2)

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;
 var epsilon;
 periods 5, 15;
 values -0.1, -0.1;
end;

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

## A Dynare example (2/2)

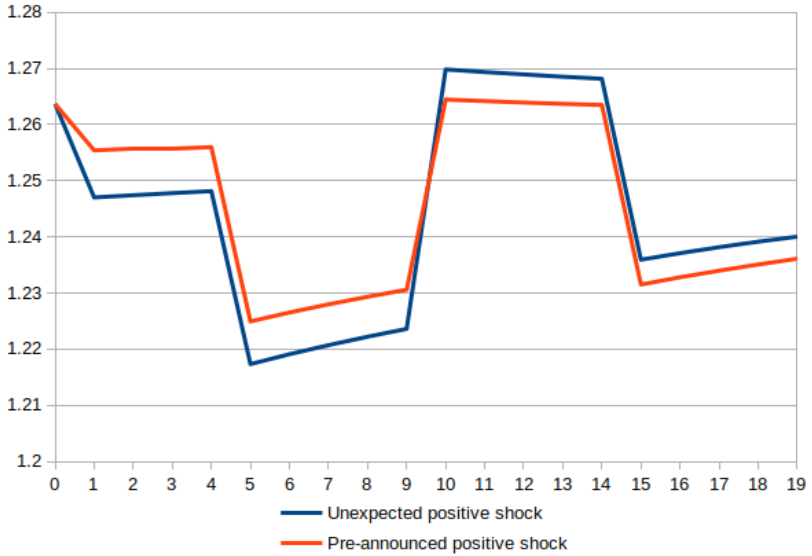
```
// Declare unexpected shock (after first simulation!)
oo_.exo_simul(11, 1) = 0.1; // Period 10 has index 11!

// Strip first 9 periods and save them
saved_endo = oo_.endo_simul(:, 1:9);  // Save periods 0 to 8
saved_exo = oo_.exo_simul(1:9, :);
oo_.endo_simul = oo_.endo_simul(:, 10:end); // Keep periods 9
oo_.exo_simul = oo_.exo_simul(10:end, :);

periods 291;
perfect_foresight_solver;

// Combine the two simulations
oo_.endo_simul = [ saved_endo oo_.endo_simul ];
oo_.exo_simul = [ saved_exo; oo_.exo_simul ];
```

## Consumption path



## Extended path (EP) algorithm

- Generalization of the previous method, where unexpected shocks may happen in all periods
- At every period, compute endogenous variables by running a deterministic simulation with:
  - the previous period as initial condition
  - the steady state as terminal condition
  - a random shock drawn for the current period
  - but no shock in the future
- Advantages:
  - shocks are unexpected *at every period*
  - nonlinearities fully taken into account
- Inconvenient: solution under certainty equivalence (Jensen inequality is violated)
- Method introduced by Fair and Taylor (1983)
- Implemented under the command `extended_path` (with option `order = 0`, which is the default)

## Extended path in Dynare

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));
```

## $k$-step ahead extended path

- Accuracy can be improved by computing conditional expectation by quadrature, computing next period endogenous variables with the previous algorithm
- Approximation: at date $t$, agents assume that there will be no more shocks after period $t+k$ (hence $k$ measures the degree of future uncertainty taken into account)
- If $k=1$: one-step ahead EP; no more certainty equivalence
- By recurrence, one can compute a $k$-step ahead EP: even more uncertainty taken into account
- Difficulty: computing complexity grows exponentially with $k$
- $k$-step ahead EP triggered with option `order = k` of `extended_path` command

## Outline

## Local approximation of stochastic models

The general problem:

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

$y$ : vector of endogenous variables

$u$ : vector of exogenous shocks

with:

$$\begin{aligned}
\mathbb{E}(u_t) &= 0 \\
\mathbb{E}(u_t u_t') &= \Sigma_u \\
\mathbb{E}(u_t u_s') &= 0 \text{ for } t \neq s
\end{aligned}$$

## What is a solution to this problem?

- A solution is a policy function of the form:

$$y_t = g\left(y_{t-1}, u_t, \sigma\right)$$

where $\sigma$ is the *stochastic scale* of the problem and:

$$u_{t+1} = \sigma\,\varepsilon_{t+1}$$

- The policy function must satisfy:

$$\mathbb{E}_t f\left(g\left(g\left(y_{t-1}, u_t, \sigma\right), u_{t+1}, \sigma\right), g\left(y_{t-1}, u_t, \sigma\right), y_{t-1}, u_t\right) = 0$$

## Local approximations

$$
\begin{aligned}
\hat{g}^{(1)}\left(y_{t+1}, u_t, \sigma\right) &= \bar{y} + g_y \hat{y}_{t-1} + g_u u_t \\
\hat{g}^{(2)}\left(y_{t+1}, u_t, \sigma\right) &= \bar{y} + \frac{1}{2} g_{\sigma\sigma} + g_y \hat{y}_{t-1} + g_u u_t \\
&\quad + \frac{1}{2}\left(g_{yy}\left(\hat{y}_{t-1} \otimes \hat{y}_{t-1}\right) + g_{uu}\left(u_t \otimes u_t\right)\right) \\
&\quad + g_{yu}\left(\hat{y}_{t-1} \otimes u_t\right) \\
\hat{g}^{(3)}\left(y_{t+1}, u_t, \sigma\right) &= \bar{y} + \frac{1}{2} g_{\sigma\sigma} + \frac{1}{6} g_{\sigma\sigma\sigma} + \frac{1}{2} g_{\sigma\sigma y} \hat{y}_{t-1} + \frac{1}{2} g_{\sigma\sigma u} u_t \\
&\quad + g_y \hat{y}_{t-1} + g_u u_t + \ldots
\end{aligned}
$$

## Breaking certainty equivalence (1/2)

The combination of future uncertainty (future shocks) and nonlinear relationships makes for precautionary motives or risk premia.

- 1st order: certainty equivalence; today's decisions don't depend on future uncertainty
- 2nd order:

$$\begin{aligned}\hat{g}^{(2)}\left(y_{t+1}, u_t, \sigma\right) = \bar{y} &+ \frac{1}{2} g_{\sigma\sigma} + g_y \hat{y}_{t-1} + g_u u_t \\ &+ \frac{1}{2}\left(g_{yy}\left(\hat{y}_{t-1} \otimes \hat{y}_{t-1}\right) + g_{uu}\left(u_t \otimes u_t\right)\right) \\ &+ g_{yu}\left(\hat{y}_{t-1} \otimes u_t\right)\end{aligned}$$

Risk premium is a constant: $\frac{1}{2} g_{\sigma\sigma}$

## Breaking certainty equivalence (2/2)

- 3rd order:

$$\hat{g}^{(3)}\left(y_{t+1}, u_t, \sigma\right) = \bar{y} + \frac{1}{2}g_{\sigma\sigma} + \frac{1}{6}g_{\sigma\sigma\sigma} + \frac{1}{2}g_{\sigma\sigma y}\hat{y}_{t-1} + \frac{1}{2}g_{\sigma\sigma u}u_t + g_y\hat{y}_{t-1} + g_u u_t + \dots$$

Risk premium is linear in the state variables:

$$\frac{1}{2}g_{\sigma\sigma} + \frac{1}{6}g_{\sigma\sigma\sigma} + \frac{1}{2}g_{\sigma\sigma y}\hat{y}_{t-1} + \frac{1}{2}g_{\sigma\sigma u}u_t$$

## The cost of local approximations

1. High order approximations are accurate around the steady state, and more so than lower order approximations
2. But can be totally wrong far from the steady state (and may be more so than lower order approximations)
3. Error of approximation of a solution $\hat{g}$, at a given point of the state space $(y_{t-1}, u_t)$:

$$\mathcal{E}\left(y_{t-1}, u_t\right) = \mathbb{E}_t f\left(\hat{g}\left(\hat{g}\left(y_{t-1}, u_t, \sigma\right), u_{t+1}, \sigma\right), \hat{g}\left(y_{t-1}, u_t, \sigma\right), y_{t-1}, u_t\right)$$

4. Necessity for pruning

## Approximation of occasionally binding constraints with penalty functions

The investment positivity constraint is translated into a penalty on the welfare:

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

s.t.

$$\begin{aligned} y_t &= c_t + i_t \\ y_t &= A_t f(k_{t-1},\ell_t) \\ k_t &= i_t + (1-\delta)k_{t-1} \\ A_t &= A^\star e^{a_t} \\ a_t &= \rho\, a_{t-1} + \varepsilon_t \end{aligned}$$

where the technology ($f$) and the preferences ($u$) are as before, and $h$ governs the strength of the penalty (*barrier parameter*)

Thanks for your attention!

Questions?