DYNARE worksop

Michel Juillard[a]

Bank of France and CEPREMAP

michel.juillard@mjui.fr

ESRI, February 11, 2014

New features:

1. 2nd and 3rd order pruning
2. model_diagnostics
3. endogenous priors
4. differentiate forward variables for deterministic simulation
5. automatic detrending
6. conditional forecasts
7. dynSeries
8. reporting tool

1. differentiate forward variables for deterministic simulations
2. automatic detrending
3. conditional forecasts
4. dynSeries
5. reporting tool
6. example

Diffentiate forward variables for deterministic simulations

We need to solve a system of nonlinear equations for the paths of the endogenous variables (y):

$$f(y_{t-1}, y_t, y_{t+1}) = 0 \qquad \forall t = 1, \dots, T-1$$

with an arbitrary initial condition y_0 (for the states) and the terminal condition $y_T = y^\star$ (for the jumping variables).

- Main approximation: the system goes back to the steady state ($y^\star$) in finite time.
- The system of non linear equations is solved with a standard Newton algorithm exploiting the sparse structure of the Jacobian.

Issue 1. The steady state may be unknown.
Issue 2. The variables may be far from the steady state at time T.

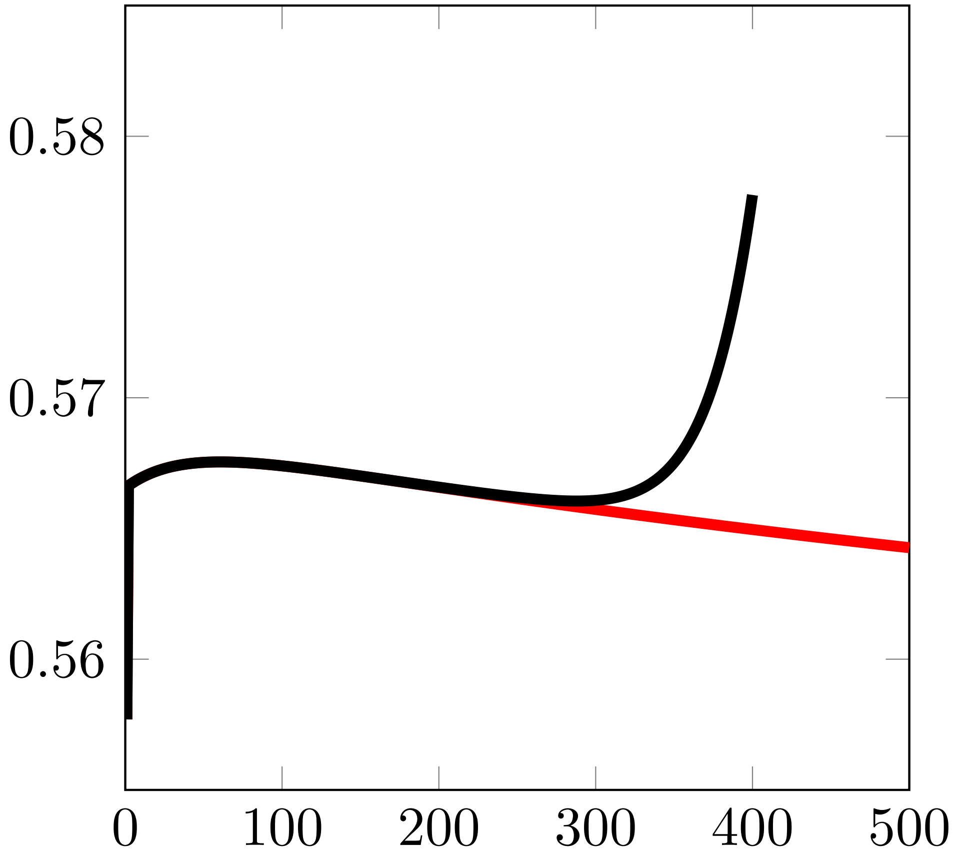
- Instead of imposing that y matches the steady state at time T, we can impose that the variations of y are zero at time T (assuming that there is no long term growth in the model).
- Basically, we just need to replace any occurence of y_{t+1} in the model by $y_t + \Delta y_{t+1}$, so that the leaded variables are variations instead of levels.
- This transformation is triggered by adding an option to the `model` block:

```
model(differentiate_forward_variables);
    ... EQUATIONS ... ;
end;
```

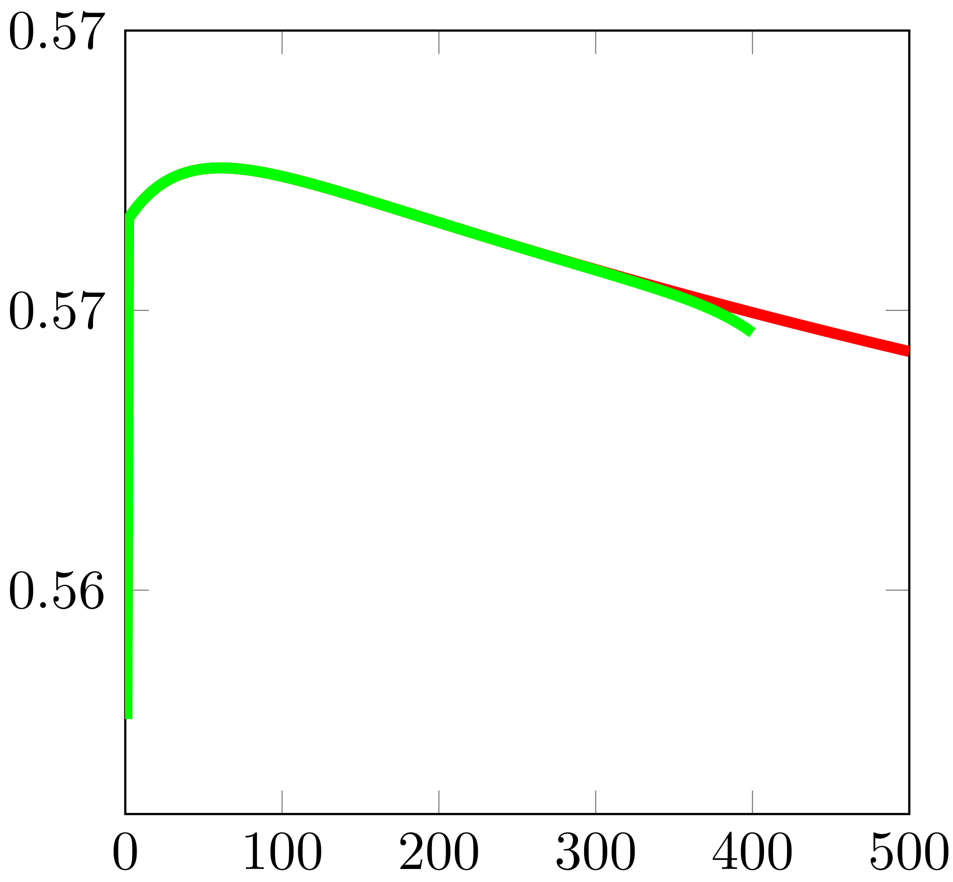
- The preprocessor automatically creates one auxiliary variable for each endogenous variable appearing with a lead in the model.
- If the model contains `x(1)`, then a variable `AUX_DIFF_VAR` will be created such that `AUX_DIFF_VAR=x-x(-1)`.
- Any occurence of `x(1)` will be replaced by `x+AUX_DIFF_VAR(1)`
- By default this transformation is applied to all the endogenous variables appearing at time $t+1$.

- Real Business Cycle model with endogenous labor supply and CES technology.
- We calibrate a very persistent productivity ($\rho = .999$) so that in period 400 the level of productivity, after an initial one percent shock, is still 0,67% above its steady state level (1).
- We consider three scenarii
  1. $T = 8000$ and a terminal condition on the levels.
  2. $T = 400$ and a terminal condition on the levels.
  3. $T = 400$ and a terminal condition on the variations.

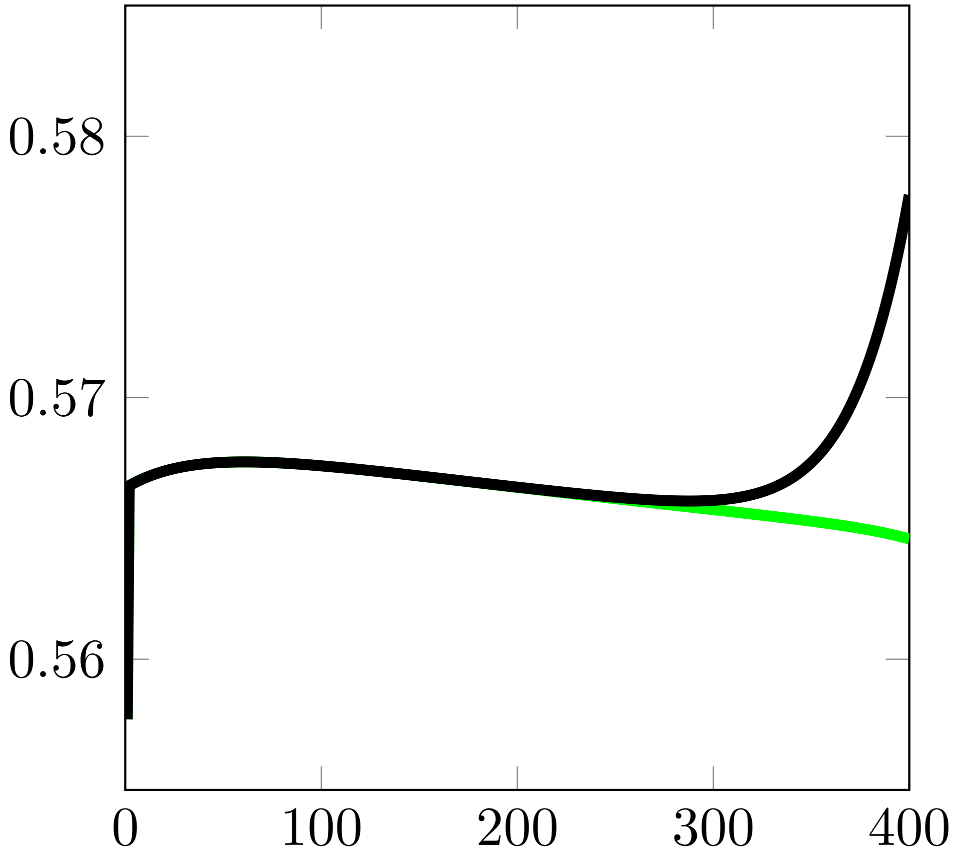
  The paths obtained under the first scenario will be interpreted as the true paths, because it seems reasonable to assume that the economy is back at the steady state after 8000 periods.



Red and black curves are respectively the solutions under scenarii 1 and 2.



Red and green curves are respectively the solutions under scenarii 1 and 3.



Black and green curves are respectively the solutions under scenarii 2 and 3.

Automatic detrending

Consider the following model of an economy.

- Representative agent preferences

$$U = \sum_{t=1}^{\infty} \left(\frac{1}{1+\rho} \right)^{t-1} E_t \left[\log\left(C_t\right) - \frac{L_t^{1+\gamma}}{1+\gamma} \right].$$

  The household supplies labor and rents capital to the corporate sector.

  - L_t is labor services
  - $\rho \in (0, \infty)$ is the rate of time preference
  - $\gamma \in (0, \infty)$ is a labor supply parameter.
  - C_t is consumption,
  - w_t is the real wage,
  - r_t is the real rental rate

- The household faces the sequence of budget constraints

$$K_t = K_{t-1}(1-\delta) + w_t L_t + r_t K_{t-1} - C_t,$$

where

  - K_t is capital at the end of period
  - $\delta \in (0,1)$ is the rate of depreciation

- The production function is given by the expression

$$Y_t = A_t K_{t-1}^{\alpha} \left((1+g)^t L_t \right)^{1-\alpha}$$

where $g \in (0,\infty)$ is the growth rate and α is a parameter.

- A_t is a technology shock that follows the process

$$A_t = A_{t-1}^{\lambda} \exp(e_t),$$

where e_t is an i.i.d. zero mean normally distributed error with standard deviation σ_1 and $\lambda \in (0, 1)$ is a parameter.

Lagrangian

$$L = \max_{C_t, L_t, K_t} \sum_{t=1}^{\infty} \left(\frac{1}{1+\rho}\right)^{t-1} E_t \Bigg[\log\left(C_t\right) - \frac{L_t^{1+\gamma}}{1+\gamma} \\ - \mu_t \left(K_t - K_{t-1}\left(1-\delta\right) - w_t L_t - r_t K_{t-1} + C_t\right) \Bigg]$$

First order conditions

$$\frac{\partial L}{\partial C_t} = \left(\frac{1}{1+\rho}\right)^{t-1} \left(\frac{1}{C_t} - \mu_t\right) = 0$$

$$\frac{\partial L}{\partial L_t} = \left(\frac{1}{1+\rho}\right)^{t-1} \left(L_t^{\gamma} - \mu_t w_t\right) = 0$$

$$\frac{\partial L}{\partial K_t} = -\left(\frac{1}{1+\rho}\right)^{t-1} \mu_t + \left(\frac{1}{1+\rho}\right)^{t} \mathbb{E}_t \left(\mu_{t+1}(1-\delta+r_t)\right) = 0$$

Eliminating the Lagrange multiplier, one obtains

$$
\begin{aligned}
L_t^{\gamma} &= \frac{w_t}{C_t} \\
\frac{1}{C_t} &= \frac{1}{1+\rho}\mathbb{E}_t\left(\frac{1}{C_{t+1}}(r_{t+1}+1-\delta)\right)
\end{aligned}
$$

$$\max_{L_t, K_{t-1}} A_t K_{t-1}^{\alpha} \left((1+g)^t L_t \right)^{1-\alpha} - r_t K_{t-1} - w_t L_t$$

First order conditions:

$$r_t = \alpha A_t K_{t-1}^{\alpha-1} \left((1+g)^t L_t \right)^{1-\alpha}$$

$$w_t = (1-\alpha) A_t K_{t-1}^{\alpha} \left((1+g)^t \right)^{1-\alpha} L_t^{-\alpha}$$

$$K_t + C_t = K_{t-1}(1-\delta) + A_t K_{t-1}^{\alpha}\left((1+g)^t L_t\right)^{1-\alpha}$$

$$A_t = A_{t-1}^{\lambda} \exp\left(e_t\right)$$

or

$$\ln A_t = \lambda \ln A_{t-1} + e_t$$

$$
\begin{aligned}
\frac{1}{C_t} &= \frac{1}{1+\rho}\mathbb{E}_t\left(\frac{1}{C_{t+1}}(r_{t+1}+1-\delta)\right) \\
L_t^{\gamma} &= \frac{w_t}{C_t} \\
r_t &= \alpha A_t K_{t-1}^{\alpha-1}\left((1+g)^t L_t\right)^{1-\alpha} \\
w_t &= (1-\alpha)A_t K_{t-1}^{\alpha}\left((1+g)^t\right)^{1-\alpha} L_t^{-\alpha} \\
K_t + C_t &= K_{t-1}(1-\delta) + A_t K_{t-1}^{\alpha}\left((1+g)^t L_t\right)^{1-\alpha}
\end{aligned}
$$

There must exist a growth rates g_c and g_k so that

$$(1+g_k)^t K_1 + (1+g_c)^t C_1 = \frac{(1+g_k)^t}{1+g_K} K_1 (1-\delta) + A\left(\frac{(1+g_k)^t}{1+g_k} K_1\right)^{\alpha} \left((1+g)^t L_t\right)^{1-\alpha}$$

So,

$$g_c = g_k = g$$

Let's define

$$\widehat{C}_t = C_t/(1+g)^t$$

$$\widehat{K}_t = K_t/(1+g)^t$$

$$\widehat{w}_t = w_t/(1+g)^t$$

$$
\begin{aligned}
\frac{1}{\widehat{C}_t(1+g)^t} &= \frac{1}{1+\rho}\mathbb{E}_t\left(\frac{1}{\widehat{C}_{t+1}(1+g)(1+g)^t}(r_{t+1}+1-\delta)\right)\\
L_t^{\gamma} &= \frac{\widehat{w}_t(1+g)^t}{\widehat{C}_t(1+g)^t}\\
r_t &= \alpha A_t\left(\widehat{K}_{t-1}\frac{(1+g)^t}{1+g}\right)^{\alpha-1}\left((1+g)^tL_t\right)^{1-\alpha}\\
\widehat{w}_t(1+g)^t &= (1-\alpha)A_t\left(\widehat{K}_{t-1}\frac{(1+g)^t}{1+g}\right)^{\alpha}\left((1+g)^t\right)^{1-\alpha}L_t^{-\alpha}\\
\left(\widehat{K}_t+\widehat{C}_t\right)(1+g)^t &= \widehat{K}_{t-1}\frac{(1+g)^t}{1+g}(1-\delta)\\
&\quad + A_t\left(\widehat{K}_{t-1}\frac{(1+g)^t}{1+g}\right)^{\alpha}\left((1+g)^tL_t\right)^{1-\alpha}
\end{aligned}
$$

$$
\begin{aligned}
\frac{1}{\widehat{C}_t} &= \frac{1}{1+\rho}\mathbb{E}_t\left(\frac{1}{\widehat{C}_{t+1}(1+g)}(r_{t+1}+1-\delta)\right) \\
L_t^{\gamma} &= \frac{\widehat{w}_t}{\widehat{C}_t} \\
r_t &= \alpha A_t\left(\frac{\widehat{K}_{t-1}}{1+g}\right)^{\alpha-1} L_t^{1-\alpha} \\
\widehat{w}_t &= (1-\alpha)A_t\left(\frac{\widehat{K}_{t-1}}{1+g}\right)^{\alpha} L_t^{-\alpha} \\
\widehat{K}_t + \widehat{C}_t &= \frac{\widehat{K}_{t-1}}{1+g}(1-\delta) + A_t\left(\frac{\widehat{K}_{t-1}}{1+g}\right)^{\alpha} L_t^{1-\alpha}
\end{aligned}
$$

State variables: K_t, A_{t-1}, and e_t, or K_t and A_t.

$$
\begin{aligned}
A &= 1 \\
r &= \rho(1+g) + \delta - 1 \\
\hat{K}/L &= (1+g)\left(\frac{r}{\alpha A}\right)^{\frac{1}{\alpha-1}} \\
\hat{w} &= (1-\alpha)A\left(\frac{\hat{K}/L}{1+g}\right)^{\alpha} \\
\hat{C}/L &= \hat{w}L^{-(1+\gamma)} \\
\hat{C}/L &= -\frac{\delta+g}{1+g}\hat{K}/L + A\left(\frac{\hat{K}/L}{1+g}\right)^{\alpha}
\end{aligned}
$$

$$L = \left(- \frac{\delta + g}{1+g} \frac{\hat{K}/L}{\hat{w}} + A \left(\frac{\hat{K}/L}{(1+g)\hat{w}^{\frac{1}{\alpha}}} \right)^{\alpha} \right)^{-\frac{1}{1+\gamma}}$$

$$= \left(\frac{r - \alpha(\delta + g)}{r(1-\alpha)} \right)^{-\frac{1}{1+\gamma}}$$

$$\hat{K} = \left(\hat{K}/L \right) L$$

$$\hat{C} = A \left(\frac{\hat{K}}{1+g} \right)^{\alpha} L^{1-\alpha} - \frac{\delta + g}{1+g} \hat{K}$$

```
var C K L w r A;
varexo e;

parameters rho delta gamma alpha lambda g;

alpha = 0.33;
delta = 0.1;
rho = 0.03;
lambda = 0.97;
gamma = 0;
g = 0.015;
```

```
model;
1/C=1/(1+rho)*(1/(C(+1)*(1+g)))*(r(+1)+1-delta);
L^gamma = w/C;
r = alpha*A*(K(-1)/(1+g))^(alpha-1)*L^(1-alpha);
w = (1-alpha)*A*(K(-1)/(1+g))^alpha*L^(-alpha);
K+C = (K(-1)/(1+g))*(1-delta)
     +A*(K(-1)/(1+g))^alpha*L^(1-alpha);
log(A) = lambda*log(A(-1))+e;
end;
```

```
 steady_state_model;
 A = 1;
 r = (1+g)*(1+rho)+delta-1;
 K_L = (1+g)*(r/(alpha A))^(1/(alpha-1);
 w = (1-alpha)*A*(K_L/(1+g))^alpha;
 L = (-((delta+g)/(1+g))*K_L/w
t        +A*(K_L/((1+g)*w^(1/alpha)))^alpha)
     ^(-1/(1+gamma));
 K = K_L*L;
 C = (1-delta)*K/(1+g)+(K_L/(1+g))^alpha*L-K;
 end;
```

```
shocks;
var e; stderr 0.01;
end;

steady;
check;

stoch_simul(order=1);
```

- multiplicative trend: $y_t = (1+g)^t \hat{y}_t$

  ```
  trend_var(growth_factor=MODEL_EXPRESSION) VARIABLE_NAME ...
  ```

- additive case (logarithm of previous case)

  ```
  log_trend_var(log_growth_factor=MODEL_EXPRESSION) VARIABLE_NAME
  ```

- nonstationary variable with multiplicative trend:

  ```
  var(deflator=MODEL_EXPRESSION) VARIABLE_NAME ...
  ```

- nonstationary variable with additive trend:

  ```
  var(log_deflator=MODEL_EXPRESSION) VARIABLE_NAME ...
  ```

- Deterministic, multiplicative case:

```
parameters g;
trend_var(growth_factor=g) tfp;
var(deflator=tfp) c k w;
```

- Stochastic, multiplicative case:

```
var g;
trend_var(growth_factor=g) tfp;
var(deflator=tfp) c k w;
```

- Stochastic, additive case:

```
var pie;
log_trend_var(log_growth_factor=pie) t_log_price_level;
var(log_deflator=t_price_level) lcpi;
```

- The algorithm is performed before introduction of auxiliary variables (because the preprocessor cannot guess the trends of auxiliary variables). Each detrended variable is replaced by itself (representing the deflated variable) multiplied by its deflator (leading or lagging the deflator if the variable is itself leaded or lagged).
- Then each leaded or lagged trend variable (declared with trend_var) is shifted to current period (by multiplying or dividing by the growth factor)

- A test is performed to check that the trends are compatible with balanced growth:
  - For all model equations $F(\ldots) = 0$, all trend variables $A_{i,t}$ and all dynamic endogenous variables $y_{j,t+k}$, check that $\frac{\partial^2 \log F}{\partial A_{i,t} \partial y_{j,t+k}} = 0$ (by evaluating the derivative at some point, typically the values given in initval, but possibly other random points)
  - If any of the cross-derivatives is not null, the equation or the specification of trend for each variable is not compatible with balanced growth. Exit with an error message identifying the equation and the list of nonstationary variables affected by the faulty trend
- If the test passed, replace trend variables (trend_var variables) by the value 1.

- When the user wants to estimate the model in level, the user needs to introduce observed variables in level, but these should not be declared in trend_var nor be declared with a deflator and nonstationary variables must be linked to the stationarized variable via (log-)linear relations. For example, if Pobs is the observed price level, it should be a standard endogenous, with the equation:

  ```
  Pobs/Pobs(-1) = 1+pie;
  ```

- The trend of the observed variable must be declared with the trend keyword used for estimation:

  ```
  observation_trends;
  P_obs (log(1+pie));
  end;
  ```

```
parameters rho delta gamma alpha lambda g;
trend_var(growth_factor=1+g) G;
var(deflator=G) C K w;
var L r A;
varexo e;
```

```
alpha = 0.33;
delta = 0.1;
rho = 0.03;
lambda = 0.97;
gamma = 0;
g = 0.015;
```

```
model;
1/C = 1/(1+rho)*(1/C(+1))*(r(+1)+1-delta);
L^gamma = w/C;
r = alpha*A*K(-1)^(alpha-1)*L^(1-alpha);
w = (1-alpha)*A*K(-1)^alpha*L^(-alpha);
K+C = K(-1)*(1-delta)+A*K(-1)^alpha*L^(1-alpha);
log(A) = lambda*log(A(-1))+e;
end;
```

```
model;
1/C=1/(1+rho)*(1/(C(+1)*(1+g)))*(r(+1)+1-delta);
L^gamma = w/C;
r = alpha*A*(K(-1)/(1+g))^(alpha-1)*L^(1-alpha);
w = (1-alpha)*A*(K(-1)/(1+g))^alpha*L^(-alpha);
K+C = (K(-1)/(1+g))*(1-delta)
     +A*(K(-1)/(1+g))^alpha*L^(1-alpha);
log(A) = lambda*log(A(-1))+e;
end;
```

```
steady_state_model;
A = 1;
r = (1+g)*(1+rho)+delta-1;
K_L = (1+g)*(r/(alpha*A))^(1/(alpha-1));
w = (1-alpha)*A*(K_L/(1+g))^alpha;
L = (-((delta+g)/(1+g))*K_L/w+A*(K_L/((1+g)*w^(1/alpha)))^alpha)
    ^(-1/(1+gamma));
K = K_L*L;
C = (1-delta)*K/(1+g)+(K_L/(1+g))^alpha*L-K;
end;
```

```
shocks;
var e; stderr 0.01;
end;

steady;
check;

stoch_simul(order=1);
```

Conditional forecast

```
conditional_forecast (OPTIONS...) [VARIABLE_NAME...];
```

- This command computes forecasts for a given constrained path of some future endogenous variables. This is done, from the reduced form representation of the DSGE model, by finding the structural shocks that are needed to match the restricted paths.
- Use `conditional_forecast_paths` block to give the list of constrained endogenous, and their constrained future path. Option `controlled_varexo` is used to specify the structural shocks which will be matched to generate the constrained path.
- Use `plot_conditional_forecast` to graph the results.

- Options:
  - `parameter_set = calibration|prior_mode|prior_mean |posterior_mode|posterior_mean|posterior_median`
    Specify the parameter set to use for the forecasting. No default value, mandatory option.
  - `controlled_varexo = (VARIABLE_NAME...)`
    Specify the exogenous variables to use as control variables. No default value, mandatory option.
  - `periods = INTEGER`
    Number of periods of the forecast. Default: 40. periods cannot be less than the number of constrained periods.
  - `replic = INTEGER`
    Number of simulations. Default: 5000.
  - `conf_sig = DOUBLE`
    Level of significance for confidence interval. Default: 0.80

- Output

  The results are not stored in the `oo_` structure but in a separate structure forecasts saved to the harddisk into a file called `<fname>_conditional_forecasts.mat`

```
var y a
varexo e u;
...
conditional_forecast_paths;
var y;
periods 1:3, 4:5;
values 2, 5;
var a;
periods 1:5;
values 3;
end;

conditional_forecast(parameter_set = calibration,
                     controlled_varexo = (e, u),
                     replic = 3000);
```

```
plot_conditional_forecast(periods = 10) a y;
```

dates and dseries classes

- Starting with version 4.4, Dynare provides a class to facilitate the handling of time series (`@dseries`)...
- Which is based on class for handling dates (`@dates`).
- Dynare also provides a new type for dates in mod files, so that one can define dates in a natural manner.
- The `@dseries` class comes with a set of methods which allows to load, manipulate, and save data.

- Matlab/Octave implementation of OOP does not allow in place modifications of instantiated objects.
- ⇒ When a method is applied to an object, a new object is instantiated (and returned).

For instance, suppose that an object `X` returns 1 when displayed:

```
>> X

X =

     1
```

Suppose there exists a method `multiplybytwo`, then:

```
>> X.multiplybytwo()

ans =

     2
```

But, `X` is unchanged:

```
>> X

X =

     1
```

⇒ A new object must be defined:

```
>> Y = X.multiplybytwo()

Y =

     2

>> X

X =

     1
```

- The `@dates` class allows to create and manipulate objects containing collections of dates.
- The `@dates` class has three private members:
  - `freq`: 1 (Annual), 4 (Quaterly), 12 (Monthly), 52 (Weekly).
  - `ndat`: the number of dates.
  - `time`: `ndat`$\times$2 array of integers.

  Each line of the `time` member corresponds to a date. The first column is the year ($\in \mathbb{Z}$), the second colum is the subperiod (a positive integer between 1 and `freq`).
- Members are private: one can read them but not modify them. If `dd` is a `@dates` object, the following statement is illegal:

  ```
  >> dd.freq = 52
  ```

- In a matlab code, a `@dates` object can be instantiated as follows:
  - `dd = dates('1990Q1')`
  - `ee = dates('1990Q1', '1990Q2', '1978Q3')`

  Note that if the instantiated object contains more than one date, it is not possible to mix frequencies. Also, the dates need not to be ordered.

- To create `@dates` object programmatically, it is more efficient to avoid string manipulations.
  - `dd = dates(4, 1990, 1)` or `dd = dates('Q', 1990, 1)`
  - `ee = dates(4, [1990; 1990; 1978], [1; 2; 3])`

  First argument is the frequency, second argument is the first column of `time` (year) and third argument is the second column of `time` (subperiod).

- It is possible to create empty `@dates` objects:
  - `qq = dates('Q')` or `qq = dates(4)`
- An empty `@dates` object can be used as a shortcut to instantiate `@dates` objects programmatically:
  - `dd = qq(1990, 1)`
  - `ee = qq([1990; 1990; 1978], [1; 2; 3])`

  i.e. without specifying the frequency.

- Starting with version 4.4, Dynare understands dates, *i.e.* `1990Q1` is a legal statement in a mod file (not in the `model` block though).
- In the background, Dynare's preprocessor recognizes and translates date tokens into `@dates` instantiations. For instance,

  ```
  initial_period = 1971Q1 ;
  ```

  is translated as:

  ```
  initial_period = dates('1971Q1') ;
  ```

  in the generated `m` file.

- Because of this behavior, using date tokens in a string will cause an error.
- For instance, if the user writes:

  ```
  disp('The first date is 1971Q1');
  ```

  in a mod file, Dynare's preprocessor will write in the generated `m` file:

  ```
  disp('The first date is dates('1971Q1')');
  ```

  which is an illegal statement in Matlab/Octave.
- Dynare preprocessor will not interpret a date token if the date is preceeded by the `$` escape parameter. The following statement:

  ```
  disp('The first date is $1971Q1');
  ```

  will be translated into:

  ```
  disp('The first date is 1971Q1');
  ```

- It is possible to concatenate `@dates` objects as we would do with Matlab/Octave's objects:
  - `a = 1990Q1; b = 1957Q1; c = -52Q1;`
  - `d = [a, b, c];`
- It is possible to create a range of consecutive dates, using the colon notation. For instance, the following statement:

  ```
  a = 1990Q3:1991Q2
  ```

  will create a `@dates` object `a` containing four dates (`1990Q3`, `1990Q4`, `1991Q1` and `1991Q2`)
- It is possible to create a range of regularly spaced dates, using the (double) colon notation. For instance, the following statement:

  ```
  a = 1990Q1:2:1991Q1
  ```

  will create a `@dates` object `a` containing three dates (`1990Q1`, `1990Q3` and `1991Q1`)

- The `@dates` class overloads the relational operators $<$, $>$, $\leq$, $\geq$, $==$ and $\sim=$.
- If `a = 1990Q1; b = 1990Q1; c = 1989Q4;` then:
  - `a<c` returns 0,
  - `a>c` returns 1,
  - `a<=c` returns 0,
  - `a==b` returns 1, ...
- Compared objects must have common frequency and the same number of elements (except if one object is a singleton).
- If one of the compared object has more than one element, say n, then a $n \times 1$ vector of 0 and 1 is returned.

- The `@dates` class overloads the `+` and `-` unary operators.
  - `a = 1990Q4;`
  - `b = +a;` adds one period to `a`, *i.e.* `b==1991Q1` returns 1.
  - `c = -a;` substracts one period to `a`, *i.e.* `b==1990Q3` returns 1.
- The `@dates` class overloads the `+` and `-` binary operators.
  - `1990Q1+4` adds four quarters and returns `1991Q1`
  - `1990M9+5` adds five months and returns `1991M2`
  - `1991Q1-4` substracts four quarters and returns `1990Q1`
  - `1991M2-1990M9` returns 5 (months)
  - `1991Q1-1990Q1` returns 4 (quarters)
- It would be meaningless to add two `@dates` objects
- `@dates` objects must have common frequency and the same number of elements except if one is a singleton.

- The `@dates` class overloads the `union`, `intersect`, `unique`, `setdiff` Matlab/Octave functions
  - `a = [1990Q1:1991Q1 1990Q1]; b = [1990Q3:1991Q3];`
  - `unique(a)` returns a `@dates` object with five elements (`1990Q1`, `1990Q2`, `1990Q3`, `1990Q4` and `1991Q1`)
  - `intersect(a,b)` returns a `@dates` object with three elements (`1990Q3`, `1990Q4` and `1991Q1`)
  - `setdiff(a,b)` returns a `@dates` object with two elements (`1990Q1`, `1990Q2`)
  - `union(a,b)` returns a `@dates` object with seven elements (repetitions are removed).
- `pop`/`append` methods removes/adds an element in a `@dates` object:
  - `b.pop(1991Q4)` removes date `1991Q4` from `b`

- The `@dseries` class allows to create and manipulate objects containing collections of time series.
- The `@dseries` class has eight members, among which:
  - `nobs`: scalar integer, the number of observations.
  - `vobs`: scalar integer, the number of variables.
  - `dates`: `@dates` object, the dates of the sample.
  - `name`: cell of strings, names of the variables.
  - `tex`: cell of strings, TeX names of the variables.
  - `data`: `nobs`×`vobs` array of doubles.
- These members are private.

- The more general approach to instantiate the `@dseries` class is to use the following syntax:

```
ts = dseries(DATA, INITIAL_PERIOD, NAMES, TEX_NAMES)
```

  where `DATA` is a $T \times N$ matrix, `INITIAL_PERIOD` is a singleton `@dates` object, `NAMES` and `TEX_NAMES` are cells of strings.

- For instance:

```
ts = dseries(randn(5,2), 1989Q3,
             {'Output'; 'Consumption'},
             {'\hat{y}'; '\hat{c}'})
```

☞

```
       | Output    | Consumption
1989Q3 | -0.49301  | 0.10932
1989Q4 | -0.18074  | 1.814
1990Q1 | 0.045841  | 0.31202
1990Q2 | -0.063783 | 1.8045
1990Q3 | 0.61134   | -0.72312
```

- It is also possible to instantiate the `@dseries` class with a file containing data (`*.xls`, `*.csv`, `*.m`, `*.mat`)
- For instance:

```
ts = dseries('../data/dataset.csv');
```

- In `*.xls` or `*.csv` files, the first line must contain the variable names and the first column must specify the dates (using the standard format: `199Q1` for quarterly data, `1990Y` for annual data, ...)
- `*.mat` or `*.m` files must contain a vector for each variable, and the following variables:
  - `INIT__` (mandatory) a singleton `@dates` object for the initial date of the sample.
  - `TEX__` (optional) a cell of strings for the TEX names.

- Suppose that `ts` is a `@dseries` object with N variables and 136 observations from `1980Q2` to `2014Q1`.
- To create a subsample with observations from `1990Q1` to `2014Q1`, we can use `@dates` range:

```
us = ts(1990Q1:2014Q1);
```

- We can also use a range of integers (observation numbers):

```
start = find(1980Q2:2014Q1==1990Q1);
us = ts(start:end);
```

- In order to extract *one observation*, a singleton `@dates` object must be used.

- Suppose that `ts` is a `@dseries` object with T observations and the following variables: `GDP_US`, `GDP_FR`, `GDP_BE`, `GDP_UK`, `CPI_US`, `CPI_FR`, `CPI_BE`, `CPI_UK`, `WAG_US`, `WAG_FR`, `WAG_BE`, `WAG_UK`
- We can extract one variable using the following syntax:

```
us = ts.GDP_FR;
```

- To extract all the `GDP` variables:

```
us = ts{'GDP_US','GDP_FR','GDP_BE','GDP_UK'};
```

- A shorter syntax can be obtained using an implicit loop:

```
us = ts{'GDP_@US,FR,BE,UK@'};
```

- Nested implicit loops can be used to select `CPI` and `GDP` data for all countries:

```
us = ts{'@GDP,CPI@_@US,FR,BE,UK@'};
```

- It is also possible to select variables using more general Matlab's regular expressions.
- Regular expressions must be defined between square brackets.
- For instance, to select the `GDP` variables for all countries:

```
us = ts{'GDP_[A-Z]'};
```

- It is not possible to use more than one regular expression.

- Suppose that `ts` and `us` are two `@dseries` objects with the same variables observed on different time ranges. These `@dseries` objects can be merged using the following syntax:

  ```
  vs = [ts; us];
  ```

- Suppose that `ts` and `us` are two `@dseries` objects with different variables observed on different time ranges. These `@dseries` objects can be merged using the following syntax:

  ```
  vs = [ts, us];
  ```

  If `ts` and `us` are not defined over the same time range, the time range of `vs` will be the union of `ts.dates` and `us.dates`, `NaN`s will be added for the missing observations.

- The @dseries class comes with a lot of methods (fully described in the manual).
- Suppose that ts is a @dseries object with variables GDP_US, GDP_FR, GDP_UK, CPI_US, CPI_FR and CPI_UK, observed between 1990Q1 and 2013Q4
- To apply the logarithmic transformation to all the GDP variables:

```
ts{'GDP_[A-Z]'} = ts{'GDP_[A-Z]'}.log();
```

- To apply the logarithmic transformation to all the GDP variables only between 1995Q3 and 2005Q4 (this is kind of weird but we can do it):

```
ts(1995Q3:2005Q4){'GDP_[A-Z]'} =
    ts(1995Q3:2005Q4){'GDP_[A-Z]'}.log();
```

- The `@dseries` class overloads the `+`, `-`, `*`, `\` and `^` which performs element by element operations.
- For instance, for the `plus` (`+`) method:
  - If `ts` and `us` are `@dseries` object with N variables, T observations and common range, then `ts+us` performs the element by element addition.
  - If `us` has one variable and common range with `ts`, `ts+us` will add the variable in `us` to all the variables in `ts` element by element.
  - If `us` has N variables and only one observation with, `ts+us` will add the observation in `us` to all the observations in `ts` element by element.
  - It is possible to add a Matlab/Octave matrix to a `@dseries` object provided that the dimensions are consistent.

- lead and lag methods are available. For instance, if ts = dseries(transpose(1:4)), then ts.lag(1) should output:

```
   | lag(Variable_1,1)
1Y | NaN
2Y | 1
3Y | 2
4Y | 3
```

and ts.lead(1):

```
   | lead(Variable_1,1)
1Y | 2
2Y | 3
3Y | 4
4Y | NaN
```

- Note that `ts.lag(1)` is equal to `ts.lead(-1)`.
- A simpler syntax is available:
  - `ts(-k)` is equivalent to `ts.lag(k)` for any $k \in \mathbb{Z}$
  - `ts(k)` is equivalent to `ts.lead(k)` for any $k \in \mathbb{Z}$
- This shorter syntax allows to instantiate objects by copy/pasting equations from the `model` block.
- For instance, if `C`, `A` `K` are `@dseries` objects, then the residuals of an Euler equation can be computed as:

```
Residuals = 1/C - beta/C(1)*
                  (exp(A(1))*K^(alpha-1)+1-delta) ;
```

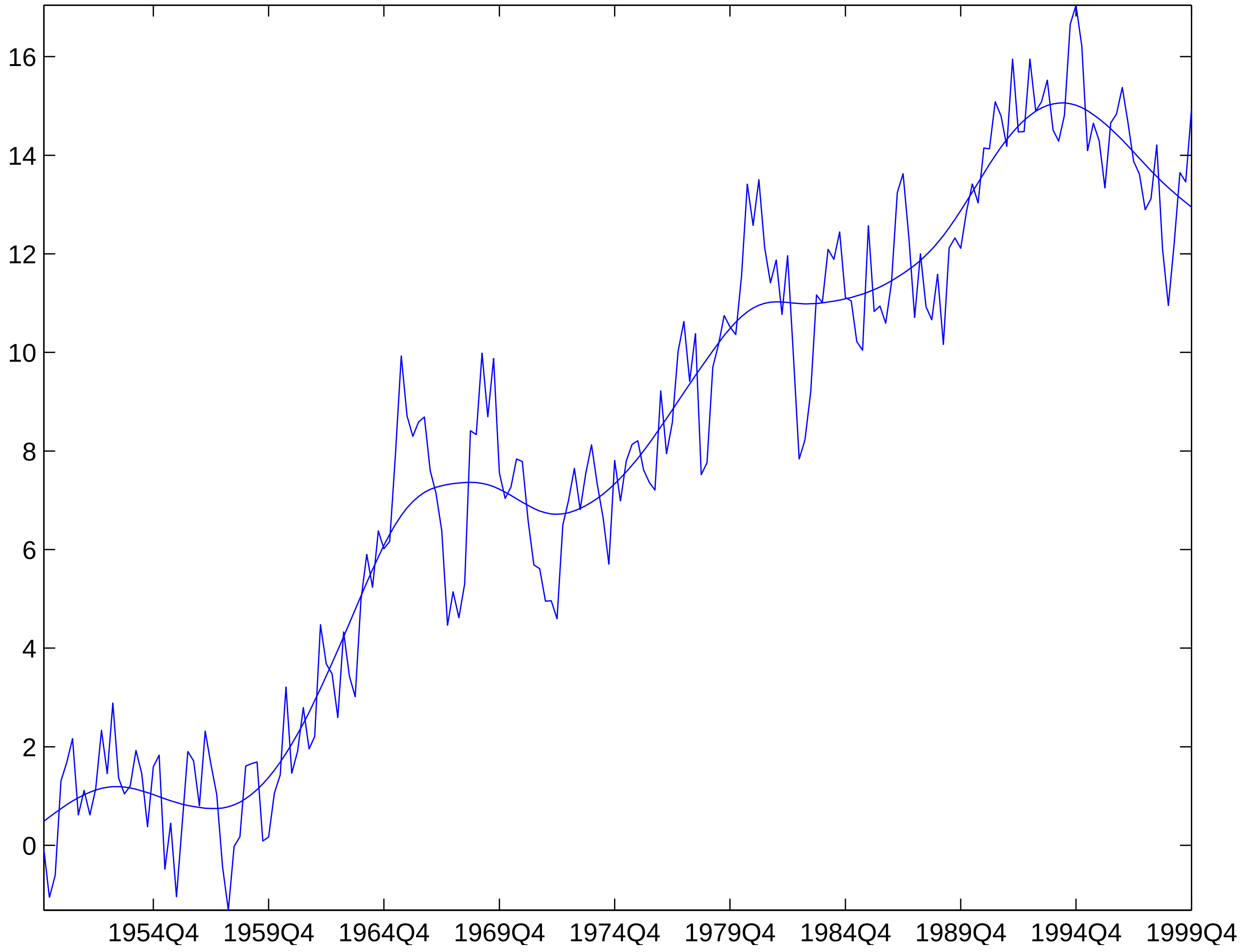
- `@dseries` class overloads the Matlab/Octave's plot function.
- Returns a Matlab/Octave plot handle, that can be used if fine tuning of the figure's properties is needed.
- If the `@dseries` object contains only one variable, additional arguments can be passed to modify the properties of the plot (as one would do with the Matlab/Octave's version of the plot function).
- If the `@dseries` contains more than one variable, it is not possible to pass these additional arguments and the properties of the plotted time series must be modified using the returned plot handle and the Matlab/Octave `set` function

How to plot `@dseries` objects (2, Simulate trended data)

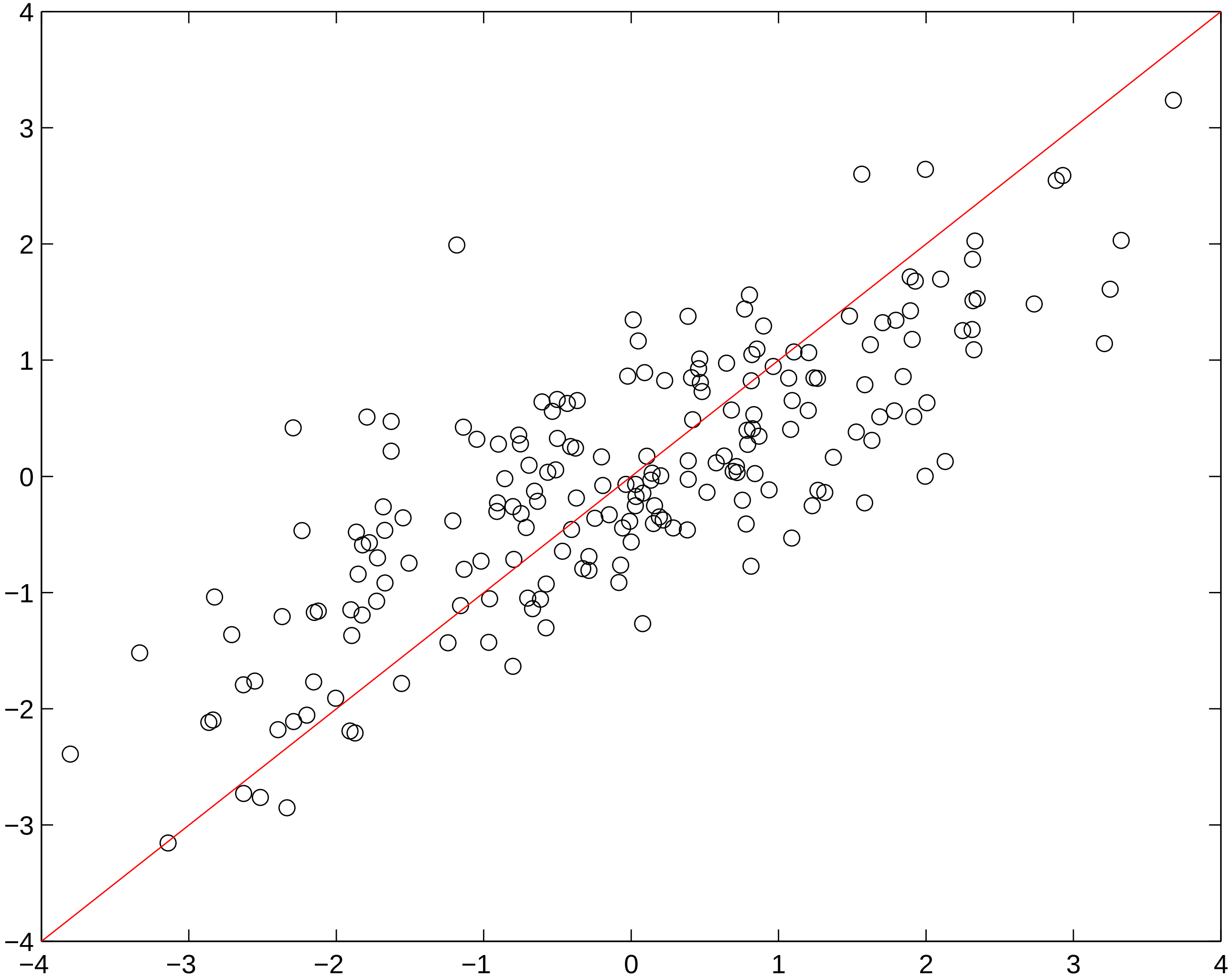
```matlab
% Generate random walk + linear trend + AR(1)
e = .2*randn(200,1);
u = randn(200,1);
stochastic_trend = cumsum(e);
deterministic_trend = .1*transpose(1:200);
x = zeros(200,1);
for t=2:200, x(t) = .75*x(t-1) + u(t); end
y = x + stochastic_trend + deterministic_trend;

% Instantiates time series objects.
ts0 = dseries(y,'1950Q1');
ts1 = dseries(x,'1950Q1'); % stationary component.

% Apply the HP filter.
ts2 = ts0.hpcycle();
ts3 = ts0.hptrend();
```




```
plot(ts0), hold on, plot(ts3), hold off
```




```
plot(ts1,ts2,'ok')
```

Reporting

- Introduce reporting functionality to Dynare
- Support reporting needs of GPM and GIMF
- Maintain Dynare's compatibility with
  - Octave 3.6 or later and Matlab 7.3 (R2006b) or later
  - Windows XP or later, OS X 10.6 or later, and GNU/Linux all flavors

```
rep = report('filename', outfile);
```

```
rep = rep.addTable('title', 'Real GDP Growth', ...
                   'range', larange, ...
                   'vlineAfter', dates('2011y'));
```

```
shortNames = {'US', 'EU', 'JA', 'EA6', 'LA6', 'RC6'};
longNames  = {'United States', 'Euro Area', 'Japan', ...
              'Emerging Asia', 'Latin America', 'Remaining ...
                  Countries'};
for i=1:length(shortNames)
    db_a = db_a.tex_rename(['PCH_GROWTH4_' shortNames{i}], ...
        longNames{i});
    rep = rep.addSeries('data', db_a{['PCH_GROWTH4_' ...
        shortNames{i}]});
    △ = db_a{['PCH_GROWTH4_' ...
        shortNames{i}]}-dc_a{['PCH_GROWTH4_' shortNames{i}]};
    △ = △.tex_rename('$\Delta$');
    rep = rep.addSeries('data', △, 'tableShowMarkers', ...
        true, 'tableAlignRight', true);
end
```

Create Graph

```
rep = rep.addPage('title', {title, 'World Oil and Food ...
    Prices'}, ...
                   'titleFormat', {'\large\bfseries', ...
                       '\large'});
rep = rep.addSection('cols', 2);
rep = rep.addGraph('title', 'World Real Oil Price', ...
                    'xrange', prange, ...
                    'shade', srange, ...
                    'showLegend', true);
```

```
rep.compile();
```

- Reporting released in Dynare 4.4.0
  - Enhanncements and bugfixes are sure to follow
  - Suggestions are welcome