Package: dsge (via r-universe)

October 25, 2026

Title Dynamic Stochastic General Equilibrium Models

Version 1.2.0

Description Specify, solve, and estimate dynamic stochastic general equilibrium (DSGE) models by maximum likelihood and Bayesian methods. Supports both linear models via an equation-based formula interface and nonlinear models via string-based equations with perturbation up to third order (Schmitt-Grohe and Uribe, 2004 <doi:10.1016/S0165-1889(03)00043-5>). Solution uses the method of undetermined coefficients (Klein, 2000 <doi:10.1016/S0165-1889(99)00045-7>). Likelihood evaluated via the Kalman filter or a bootstrap particle filter (Gordon et al., 1993). Bayesian estimation uses adaptive Random-Walk Metropolis-Hastings or Particle Marginal Metropolis-Hastings (Andrieu et al., 2010 <doi:10.1111/j.1467-9868.2009.00736.x>) with parallel chain support. Additional tools include Bayes factor model comparison with Kass-Raftery evidence scales, Ramsey optimal policy via linear-quadratic regulator, nonlinear perfect foresight via stacked-time Newton (Juillard et al., 1998), Kalman smoothing, historical shock decomposition, local identification diagnostics, parameter sensitivity analysis, occasionally binding constraints, impulse-response functions, forecasting, and robust standard errors.

License MIT + file LICENSE

Depends R (>= 3.5.0)

Imports grDevices, graphics, stats, numDeriv

Suggests coda, Matrix, R.matlab, readxl, testthat (>= 3.0.0), knitr, rmarkdown

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.3

VignetteBuilder knitr

NeedsCompilation no

Author Mustapha Wasseja Mohammed [aut, cre]

Maintainer Mustapha Wasseja Mohammed <muswaseja@gmail.com>

Repository https://muswaseja92.r-universe.dev

Date/Publication 2026-09-25 14:10:02 UTC

RemoteUrl https://github.com/cran/dsge

RemoteRef HEAD

RemoteSha 5d8675d1f464b5888a0dc0375736ec85527fd782

Contents

bayes_dsge . . . 4
bayes_dsge_var . . . 6
bayes_dsge_var_mh . . . 8
bayes_factor . . . 10
bayes_particle . . . 12
bayes_smc . . . 14
calibrated_smoother . . . 15
check_identification . . . 16
conditional_forecast . . . 18
conditional_forecast.dsge_dsgevar . . . 20
discretionary_policy . . . 20
dsge_model . . . 22
dsgenl_model . . . 23
E . . . 25
endogenous_prior . . . 25
estimate . . . 27
extended_path . . . 28
fitted.dsge_fit . . . 30
forecast . . . 30
forecast.dsge_dsgevar . . . 31
forecast.dsge_fit . . . 32
geweke_test . . . 32
global_sensitivity . . . 33
gmm_estimate . . . 35
irf.dsge_bayes . . . 37
irf_2nd_order . . . 38
irf_match . . . 39
kalman_filter_skewed . . . 41
lead . . . 43
linearize . . . 43
marginal_likelihood . . . 44
mcmc_diagnostics . . . 45
model_covariance . . . 45
model_latex . . . 46
ms_filter . . . 48
obc_constraint . . . 49
obs . . . 50

osr . . . 51
pac_simulate . . . 53
pac_target_loading . . . 54
pac_weights . . . 55
parameter_sensitivity . . . 56
particle_filter . . . 58
particle_filter_loglik . . . 59
perfect_foresight . . . 60
perfect_foresight_expect_err . . . 62
perfect_foresight_nonlinear . . . 64
plot.dsge_bayes . . . 66
plot.dsge_decomposition . . . 68
plot.dsge_forecast . . . 68
plot.dsge_irf . . . 69
plot.dsge_occbin . . . 69
plot.dsge_perfect_foresight . . . 70
plot.dsge_smoothed . . . 71
plot.dsge_variance_decomposition . . . 71
policy_matrix . . . 72
posterior_predictive . . . 72
predetermined . . . 73
predict.dsge_fit . . . 74
prediction_accuracy . . . 75
prediction_interval . . . 76
prior . . . 76
prior_posterior_update . . . 77
ramsey_policy . . . 79
read_dynare . . . 81
residuals.dsge_fit . . . 85
robust_vcov . . . 85
shock_decomposition.dsge_solution . . . 86
simulate_2nd_order . . . 88
simulate_3rd_order . . . 88
simulate_occbin . . . 89
simulate_perfect_foresight . . . 91
smm_estimate . . . 92
smooth_shocks.dsge_solution . . . 94
smooth_states.dsge_solution . . . 95
solve_dsge . . . 96
stability . . . 97
state . . . 98
steady_state . . . 99
summary.dsge_perfect_foresight . . . 100
transition_matrix . . . 100
unobs . . . 101
variance_decomposition . . . 101
vcov.dsge_fit . . . 103
welfare_loss . . . 103

Index **105**

bayes_dsge *Estimate a DSGE Model by Bayesian Methods*

Description

Estimates the parameters of a DSGE model using Random-Walk Metropolis-Hastings (RWMH) with adaptive proposal covariance. Supports both linear models (dsge_model) and nonlinear models (dsgenl_model). For nonlinear models, the steady state is re-solved and the model re-linearized at each candidate parameter vector.

Usage

```
bayes_dsge(
  model,
  data,
  priors,
  chains = 2L,
  iter = 5000L,
  warmup = floor(iter/2),
  thin = 1L,
  proposal_scale = 0.1,
  demean = TRUE,
  seed = NULL,
  n_cores = 1L,
  endogenous_prior = NULL,
  presample = 0L,
  shock_start = NULL
)
```

Arguments

model	A `dsge_model` or `dsgenl_model` object, or a Dynare model imported with `read_dynare()`.
data	A data frame, matrix, or `ts` object containing the observed variables.
priors	Named list of `dsge_prior` objects (one per free parameter). For a model imported with `read_dynare()`, defaults to the priors translated from its `estimated_params` block. Shock standard deviations get default `inv_gamma(0.01, 0.01)` priors unless overridden (names: `"sd_e.shock_name"`).
chains	Integer. Number of MCMC chains. Default is 2.
iter	Integer. Total iterations per chain (warmup + sampling). Default is 5000.
warmup	Integer. Number of warmup iterations. Default is `floor(iter / 2)`.
thin	Integer. Thinning interval. Default is 1.
proposal_scale	Numeric. Initial proposal standard deviation scale. Default is 0.1.

`demean` Logical. If `TRUE` (default), observed variables are demeaned before estimation. For nonlinear models, demeaning is not applied because the Kalman filter operates around the parameter-specific steady state.

`seed` Integer. Random seed for reproducibility. If `NULL`, no seed is set.

`n_cores` Integer. Number of CPU cores for parallel chain execution. Set to `1` (default) for sequential execution. Values greater than `chains` are silently clamped to `chains`. On Windows a PSOCK cluster is used; on POSIX systems (`mclapply` is used instead. Parallel execution requires the `dsge` package to be installed (not just loaded via `devtools::load_all()`).

`endogenous_prior` Optional `dsge_endog_prior` object returned by `endogenous_prior()`. If supplied, its log density is added to the parameter log-prior at every MH evaluation. Default `NULL` (no endogenous prior, identical to the original behaviour).

`presample` Integer. Number of initial observations used only to initialise the Kalman filter and excluded from the likelihood (as Dynare's `presample` option). Default 0; for a model imported with `read_dynare()`, the value in the file's `estimation` command.

`shock_start` Optional named numeric vector of starting values for the shock standard deviations (names as the shocks). Defaults to the standard deviation of the data; for a model imported with `read_dynare()`, the shock standard deviations from the file.

Details

The sampler operates in unconstrained space with appropriate transformations (log for positive parameters, logit for unit-interval parameters) and Jacobian corrections. The proposal covariance is adapted during warmup to target approximately 25\

Chains are initialized by drawing from the prior. If any draw yields a non-finite log-posterior, the starting values are jittered until a valid point is found.

For nonlinear models (`dsgenl_model`), each posterior evaluation involves: (1) solving the deterministic steady state at the candidate parameters, (2) computing a first-order linearization, (3) solving the resulting linear system, and (4) evaluating the Kalman filter likelihood. If any stage fails (e.g., steady-state non-convergence, Blanchard-Kahn violation), the proposal is safely rejected. This is computationally more expensive than linear Bayesian estimation.

Parallel chains:

When `n_cores` > `1` each chain runs in its own R worker process, so wall-clock time scales roughly as `chains / min(n_cores, chains)`. Results are numerically identical to sequential execution given the same per-chain seeds. The `parallel` package (part of base R) is used; no additional installation is required.

Value

An object of class `"dsge_bayes"` containing posterior draws and diagnostics. For nonlinear models, the result also includes `solve_failures`, the number of parameter draws where steady-state, linearization, or solution failed.

Examples

```
m <- dsge_model(
  obs(y ~ z),
  state(z ~ rho * z),
  start = list(rho = 0.5)
)

set.seed(42)
z <- numeric(200); for (i in 2:200) z[i] <- 0.8 * z[i-1] + rnorm(1)
dat <- data.frame(y = z)

fit <- bayes_dsge(m, data = dat,
                  priors = list(rho = prior("beta", shape1 = 2, shape2 = 2)),
                  chains = 2, iter = 2000, seed = 1)
summary(fit)
```

---

bayes_dsge_var *Bayesian VAR with DSGE-Implied Prior (DSGE-VAR)*

Description

Estimates a Bayesian VAR whose prior on the autoregressive coefficients and innovation covariance is centred on the second-moment implications of a structural DSGE model. The scalar `lambda` controls the tightness of the DSGE prior relative to the sample data: `lambda = 0` gives essentially an unrestricted Bayesian VAR, while large `lambda` pulls the VAR toward DSGE implied dynamics.

Usage

```
bayes_dsge_var(
  model,
  data,
  params = NULL,
  shock_sd = NULL,
  p = 4L,
  lambda = 1,
  n_draws = 1000L,
  include_intercept = TRUE,
  seed = NULL
)
```

Arguments

`model`	A `dsge_solution` or `dsge_model` object. If a model is supplied, `params` and `shock_sd` must also be given so the model can be solved.
`data`	Numeric matrix or data frame of observed variables (rows = time, columns = observables). Column names must match the model's observed-variable names.

params — Named numeric vector of structural parameters, used only when `model` is a `dsge_model`.

shock_sd — Named numeric vector of shock standard deviations, used only when `model` is a `dsge_model`.

p — Integer. VAR lag order. Default 4.

lambda — Numeric scalar (>= 0). Weight on the DSGE prior in units of effective sample size; the prior is worth `lambda * T` observations. Default 1.0.

n_draws — Integer. Number of posterior draws to return. Default 1000.

include_intercept — Logical. Add a constant term to the VAR. Default `TRUE`.

seed — Optional integer seed for reproducibility.

Details

The combined posterior moments are

$$\bar{M}_{XX} = X'X + \lambda T \Gamma_{XX}(\theta), \quad \bar{M}_{XY} = X'Y + \lambda T \Gamma_{XY}(\theta), \quad \bar{M}_{YY} = Y'Y + \lambda T \Gamma_{YY}(\theta)$$

where the DSGE-implied moment matrices are constructed from the unconditional autocovariances $\Gamma_{yy}(0), \ldots, \Gamma_{yy}(p)$. The unconditional state covariance solves $\Sigma_x = H \Sigma_x H' + M M'$; observable autocovariances are $\Gamma_{yy}(k) = Z H^{|k|} \Sigma_x Z'$ (with appropriate transpose for negative lags).

The approximate log marginal likelihood follows the Del Negro– Schorfheide formula

$$\log p(Y \mid \lambda) = -\frac{T n_y}{2} \log \pi + \frac{n_y}{2} \log |\Gamma_{XX} \cdot \lambda T| - \frac{n_y}{2} \log |\bar{M}_{XX}| + \frac{\lambda T}{2} \log |\lambda T \, \Gamma_{YY|X}| - \frac{\bar{T}}{2} \log |\bar{S}| + \log \Gamma_{n_y}(\bar{T}/2) - \log \Gamma_{n_y}$$

where Γ_{n_y} is the multivariate gamma function. This value is useful for comparing different choices of `lambda` (higher = better fit).

Value

An object of class `"dsge_dsgevar"` with elements:

`Phi_post` Array (k x n_y x n_draws) of posterior draws of the VAR coefficient matrix.

`Sigma_post` Array (n_y x n_y x n_draws) of posterior draws of the innovation covariance matrix.

`Phi_mean` Posterior mean of `Phi`.

`Sigma_mean` Posterior mean of `Sigma`.

`log_marg_lik` Approximate log marginal likelihood of the DSGE-VAR(lambda) model – useful for choosing the optimal `lambda` by maximisation.

`lambda` The prior weight used.

`p` The lag order.

`T` Effective sample size used (rows of data minus p).

`var_names` Observable / VAR variable names.

`solution` The DSGE solution used to build the prior.

References

Del Negro, M. and Schorfheide, F. (2004). Priors from general equilibrium models for VARs. *International Economic Review*, 45(2), 643-673.

Examples

```
nk <- dsge_model(
  obs(p   ~ beta * lead(p) + kappa * x),
  unobs(x ~ lead(x) - (r - lead(p) - g)),
  obs(r   ~ psi * p + u),
  state(u ~ rhou * u),
  state(g ~ rhog * g),
  fixed = list(beta = 0.99),
  start = list(kappa = 0.1, psi = 1.5, rhou = 0.7, rhog = 0.9)
)
sol <- solve_dsge(nk,
  params   = c(kappa = 0.1, psi = 1.5, rhou = 0.7, rhog = 0.9),
  shock_sd = c(e.u = 1.0, e.g = 0.5))
# Simulate data from the solution
set.seed(1)
TT <- 120
xst <- matrix(0, TT, nrow(sol$H))
y   <- matrix(0, TT, nrow(sol$G))
for (t in 2:TT) {
  e <- rnorm(ncol(sol$M)) * c(1, 0.5)
  xst[t, ] <- as.numeric(sol$H %*% xst[t-1, ] + sol$M %*% e)
  y[t, ]   <- as.numeric(sol$G %*% xst[t, ])
}
colnames(y) <- rownames(sol$G)
dat <- as.data.frame(y[, nk$variables$observed, drop = FALSE])
fit <- bayes_dsge_var(sol, data = dat, p = 2, lambda = 1.0,
                      n_draws = 100, seed = 1)
print(fit)
```

`bayes_dsge_var_mh` *Joint Bayesian Estimation of DSGE-VAR(lambda)*

Description

Estimates the structural DSGE parameters, shock standard deviations and the DSGE-prior weight λ jointly by random-walk Metropolis-Hastings. The VAR coefficients are analytically marginalised out at every iteration via the Normal-inverse-Wishart conjugate posterior, so only $(\theta_{\mathrm{DSGE}}, \sigma, \lambda)$ are explicitly sampled.

Usage

```
bayes_dsge_var_mh(
  model,
  data,
  priors,
  lambda_prior = NULL,
  p = 4L,
  chains = 2L,
  iter = 5000L,
  warmup = NULL,
  thin = 1L,
  proposal_scale = 0.1,
  include_intercept = TRUE,
  seed = NULL
)
```

Arguments

`model`	A `dsge_model` object.
`data`	Matrix or data frame of observable variables. Column names must match a subset of `model$variables$observed`.
`priors`	Named list of `dsge_prior` objects for the free structural parameters. Shock standard deviations receive default `inv_gamma(0.1, 2)` priors unless overridden under names `"sd_e.<shockname>"`.
`lambda_prior`	A `dsge_prior` object for the DSGE-VAR weight λ. Default: `prior("uniform", lower = 0, upper = 10)`, matching the Del Negro–Schorfheide / Dynare default.
`p`	Integer. VAR lag order. Default 4.
`chains`	Integer. Number of MH chains. Default 2.
`iter`	Integer. Total iterations per chain (warmup + sampling). Default 5000.
`warmup`	Integer. Warmup iterations. Default `floor(iter/2)`.
`thin`	Integer. Thinning interval. Default 1.
`proposal_scale`	Numeric. Initial RW proposal scale. Default 0.1.
`include_intercept`	Logical. Include a constant term in the VAR. Default `TRUE`.
`seed`	Optional integer seed.

Details

The likelihood kernel is the DSGE-VAR closed-form log marginal likelihood (Del Negro & Schorfheide 2004, eqs. 19–21):

$$\log p(Y \mid \theta, \lambda) = -\frac{T n_y}{2} \log \pi + \frac{n_y}{2} \log \left|\lambda T\, \Gamma_{XX}(\theta)\right| - \frac{n_y}{2} \log |\bar{M}_{XX}| + \frac{\lambda T}{2} \log \left|\lambda T\, \Gamma_{YY|X}(\theta)\right| - \frac{\bar{T}}{2} \log |\bar{S}| + \log \Gamma_{n_y}(\bar{T}/2)$$

This is added to log priors on the structural parameters, shock standard deviations and λ to form the log-posterior.

The sampler is the same adaptive random-walk Metropolis-Hastings used by bayes_dsge; it operates in unconstrained parameter space with appropriate Jacobian corrections (log for positive parameters, logit for bounded parameters).

Value

An object of class "dsge_dsgevar_mh" containing:

posterior Array (iter-warmup) x n_params x chains of posterior draws (in natural parameter space). Last column is lambda.

param_names Names of all sampled parameters.

acceptance_rate Per-chain MH acceptance rate.

lambda_posterior_mean Posterior mean of λ.

lambda_posterior_q 2.5\ of λ.

model, data, p, include_intercept Inputs needed by downstream methods such as forecast.dsge_dsgevar_mh.

free_parameters Names of free DSGE structural parameters.

shock_names Names of shocks whose SDs are estimated.

n_iter, n_warmup, chains MH settings.

References

Del Negro, M. and Schorfheide, F. (2004). Priors from general equilibrium models for VARs. *International Economic Review*, 45(2), 643-673.

See Also

bayes_dsge_var for VAR estimation with a fixed DSGE solution; forecast.dsge_dsgevar_mh and conditional_forecast.dsge_dsgevar_mh for forecasts integrating over the joint posterior.

---

bayes_factor *Bayes factor and posterior odds ratio for DSGE model comparison*

Description

Computes Bayes factors (ratios of marginal likelihoods) between pairs of estimated DSGE models and converts them to posterior odds ratios when prior model probabilities are supplied. Evidence strength follows the Kass-Raftery (1995) scale.

Usage

```
bayes_factor(..., prior_odds = NULL, method = "harmonic_mean", tau = 0.5)
```

Arguments

`...`	Two or more objects of class `"dsge_bayes"` or `"dsge_marginal_likelihood"` (results of `marginal_likelihood()`). Named arguments are used as model labels; unnamed arguments are labelled `Model 1`, `Model 2`, etc.
`prior_odds`	A numeric vector of prior model probabilities (or unnormalized weights) in the same order as `...`. Defaults to `NULL` (equal prior probabilities).
`method`	Passed to `marginal_likelihood()` when the inputs are `"dsge_bayes"` objects and marginal likelihoods have not yet been computed. Default: `"harmonic_mean"`.
`tau`	Truncation parameter passed to `marginal_likelihood()`. Default: `0.5`.

Details

The Bayes factor $BF_{12} = p(y|M_1)/p(y|M_2)$ measures how much the data prefer model M_1 over M_2. Following Kass and Raftery (1995), evidence strength is assessed via $2 \log BF_{12}$:

2 log BF	Evidence for M1
< 0	Negative (favours M2)
0 to 2	Not worth more than a bare mention
2 to 6	Positive
6 to 10	Strong
10	Very strong

When `prior_odds` is supplied the posterior model probability of model i is

$$P(M_i|y) \propto p(y|M_i) \cdot \pi_i$$

where π_i is the (possibly unnormalized) prior weight.

The marginal likelihood estimates are based on the modified harmonic mean estimator and inherit its limitations (see `marginal_likelihood`). Use consistent MCMC settings across all models being compared.

Value

An object of class `"dsge_bayes_factor"` containing:

`log_ml` Named numeric vector of log marginal likelihoods.

`nse` Numerical standard errors for each log-ML estimate.

`bf_matrix` Matrix of log Bayes factors $\log BF_{ij} = \log ML_i - \log ML_j$.

`posterior_probs` Posterior model probabilities.

`prior_odds` Prior model probabilities used.

`model_names` Character vector of model labels.

References

Kass, R. E. and Raftery, A. E. (1995). Bayes factors. *Journal of the American Statistical Association*, 90(430), 773-795.

See Also

marginal_likelihood, bayes_dsge

Examples

```
m1 <- dsge_model(
  obs(y ~ lead(y) + u),
  state(u ~ rho * u),
  start = list(rho = 0.5)
)
m2 <- dsge_model(
  obs(y ~ u),
  state(u ~ rho * u),
  start = list(rho = 0.5)
)
set.seed(1)
y <- cumsum(rnorm(60, sd = 0.1))
dat <- data.frame(y = y)
pr <- list(rho = prior("beta", shape1 = 5, shape2 = 5))
fit1 <- bayes_dsge(m1, dat, pr, chains = 1L, iter = 500L, warmup = 200L)
fit2 <- bayes_dsge(m2, dat, pr, chains = 1L, iter = 500L, warmup = 200L)
bf <- bayes_factor(M1 = fit1, M2 = fit2)
print(bf)
```

---

bayes_particle — *Bayesian DSGE Estimation Using Particle Marginal Metropolis-Hastings*

Description

Estimates DSGE model parameters via the Particle Marginal Metropolis-Hastings (PMMH) algorithm (Andrieu, Doucet & Holenstein, 2010). The particle filter replaces the Kalman filter, enabling fully nonlinear likelihood evaluation without any linearization.

Usage

```
bayes_particle(
  model,
  data,
  priors,
  chains = 1L,
  iter = 2000L,
  warmup = floor(iter/2),
  thin = 1L,
  n_particles = 500L,
  meas_sd = 0.001,
```

```
  proposal_scale = 0.1,
  demean = TRUE,
  seed = NULL
)
```

Arguments

`model`	A `dsge_model` or `dsgenl_model` object.
`data`	A data frame or matrix of observed variables.
`priors`	Named list of `dsge_prior` objects.
`chains`	Integer. Number of MCMC chains. Default 1.
`iter`	Integer. Total iterations per chain. Default 2000.
`warmup`	Integer. Warmup iterations. Default `floor(iter/2)`.
`thin`	Integer. Thinning interval. Default 1.
`n_particles`	Integer. Particles per likelihood evaluation. Default 500. Higher values give more accurate but slower estimates.
`meas_sd`	Numeric. Measurement error SD. Default 0.001.
`proposal_scale`	Numeric. Initial RWMH proposal scale. Default 0.1.
`demean`	Logical. Demean observed data before estimation. Default TRUE.
`seed`	Integer. Random seed.

Details

The PMMH algorithm is an exact Bayesian method: the particle filter provides an unbiased estimator of the likelihood, and the resulting Markov chain targets the exact posterior distribution. A larger `n_particles` gives a less noisy likelihood estimate and better mixing, at the cost of more computation per iteration.

As a rough guide, `n_particles = 500` is adequate for models with up to 5-6 states; larger models may require 1000-2000.

Value

An object of class `c("dsge_particle", "dsge_bayes")` with the same structure as `bayes_dsge` plus:

`n_particles` Number of particles used.

`meas_sd` Measurement error SD.

`estimator` Character string `"pmmh"`.

References

Andrieu, C., Doucet, A. & Holenstein, R. (2010). Particle Markov chain Monte Carlo methods. Journal of the Royal Statistical Society: Series B, 72(3), 269-342.

Fernandez-Villaverde, J. & Rubio-Ramirez, J. F. (2007). Estimating macroeconomic models: A likelihood approach. Review of Economic Studies, 74(4), 1059-1087.

See Also

particle_filter, particle_filter_loglik, bayes_dsge

Examples

```
m <- dsge_model(
  obs(y ~ z), state(z ~ rho * z), start = list(rho = 0.8)
)
set.seed(2)
z <- numeric(80); for (i in 2:80) z[i] <- 0.8 * z[i-1] + rnorm(1)
dat <- data.frame(y = z)
pr  <- list(rho = prior("beta", shape1 = 2, shape2 = 2))
fit <- bayes_particle(m, dat, pr,
                      chains = 1L, iter = 300L, warmup = 150L,
                      n_particles = 200L, seed = 2L)
coef(fit)
```

---

bayes_smc *Tempered Sequential Monte Carlo Sampler for DSGE Estimation*

Description

Runs a tempered SMC sampler over the DSGE parameter space. At each tempering stage particles are reweighted, optionally resampled, and mutated by random-walk Metropolis kernels. More robust than vanilla RWMH for posteriors with multiple modes or strong banana-shaped nonlinearities; the standard alternative to bayes_dsge.

Usage

```
bayes_smc(
  model,
  data,
  priors,
  n_particles = 500L,
  n_phi = 30L,
  phi_schedule = c("linear", "quadratic"),
  n_mh = 1L,
  ess_threshold = 0.5,
  scale = NULL,
  seed = NULL
)
```

Arguments

`model`	A `dsge_model` object.
`data`	Matrix or data frame of observed variables.
`priors`	Named list of `dsge_prior` objects.
`n_particles`	Integer. Number of particles. Default 500.
`n_phi`	Integer. Number of tempering stages. Default 30.
`phi_schedule`	Character. Tempering schedule shape. `"linear"` = uniform on the unit interval (default). `"quadratic"` = tighter spacing near phi = 1 (recommended for sharp posteriors).
`n_mh`	Integer. Number of MH mutation steps per tempering stage. Default 1.
`ess_threshold`	Numeric. Resample when effective sample size falls below `ess_threshold * n_particles`. Default 0.5.
`scale`	Numeric. Mutation proposal scale multiplier on the particle-cloud Cholesky factor. Default `2.38 / sqrt(d)` (Roberts-Gelman-Gilks scaling).
`seed`	Optional integer seed.

Value

An object of class `c("dsge_smc","dsge_bayes")` with the same field layout as `bayes_dsge()`. Key fields:

`posterior` Array (n_particles x n_par x 1) of final particles, treated as posterior draws.

`log_marg_lik` Log-marginal-likelihood estimate from the tempering recursion.

`ess_path` Effective sample size at each stage.

`acceptance_path` MH acceptance rate at each stage.

References

Chopin, N. (2002). A sequential particle filter method for static models. *Biometrika*, 89(3), 539-552.

Herbst, E. and Schorfheide, F. (2014). Sequential Monte Carlo sampling for DSGE models. *Journal of Applied Econometrics*, 29(7), 1073-1098.

`calibrated_smoother` *Run the Kalman Smoother on a Calibrated Model*

Description

Convenience wrapper that runs the Kalman smoother, smoothed-shock recovery, and historical shock decomposition on a calibrated `dsge_solution` object together with observed data. Analogous to Dynare's `calibrated_smoother` command.

Usage

```
calibrated_smoother(sol, data, what = c("states", "shocks", "decomposition"))
```

Arguments

sol	A dsge_solution object (from solve_dsge).
data	Matrix or data frame of observed variables. Column names must match rownames(sol$D) (the observable names).
what	Character vector of outputs to compute. Any subset of c("states", "shocks", "decomposition"). Default is all three.

Value

A named list containing the requested smoothed-output objects. Elements are themselves objects of the same classes returned by smooth_states, smooth_shocks, and shock_decomposition, so all existing print and plot methods apply.

See Also

smooth_states.dsge_solution, smooth_shocks.dsge_solution, shock_decomposition.dsge_solution.

Examples

```
m <- dsge_model(
  obs(y ~ z),
  state(z ~ rho * z),
  fixed = list(rho = 0.8))
sol <- solve_dsge(m, params = c(rho = 0.8), shock_sd = c(z = 1))
set.seed(1)
e <- rnorm(100)
z_path <- numeric(100)
for (i in 2:100) z_path[i] <- 0.8 * z_path[i - 1] + e[i]
out <- calibrated_smoother(sol, data = data.frame(y = z_path))
plot(out$states)
```

---

check_identification *Check Local Identification of DSGE Parameters*

Description

Assesses local identification by computing the Jacobian of the mapping from structural parameters to model-implied autocovariance moments. Uses an SVD decomposition to detect rank deficiency (non-identification) and near-collinearity (weak identification).

Usage

```
check_identification(x, ...)

## S3 method for class 'dsge_fit'
check_identification(x, n_lags = 4L, tol = 1e-06, ...)

## S3 method for class 'dsge_bayes'
check_identification(x, n_lags = 4L, tol = 1e-06, ...)
```

Arguments

`x`	A `dsge_fit` or `dsge_bayes` object.
`...`	Additional arguments (currently unused).
`n_lags`	Integer. Number of autocovariance lags to include in the moment vector. Default is 4.
`tol`	Numeric. Singular values below `tol` times the largest singular value are considered zero (rank deficiency). Default is 1e-6.

Details

The identification check constructs the moment vector $m(\theta) = \mathrm{vec}(\Gamma(0), \Gamma(1), \ldots, \Gamma(K))$ where $\Gamma(k)$ is the autocovariance of observables at lag k, implied by the state-space solution. The Jacobian $J = \partial m / \partial \theta$ is computed numerically.

A parameter is locally identified if the Jacobian has full column rank. If the rank is deficient, some linear combination of parameters cannot be distinguished from the data.

Per-parameter identification strength is measured by the norm of the corresponding Jacobian column: parameters with small column norms have little influence on the moments and may be weakly identified.

The condition number of J flags near-collinearity: a large condition number indicates that some parameter combinations are hard to distinguish.

Value

An object of class `"dsge_identification"` containing:

jacobian The Jacobian matrix (n_moments x n_params).

svd SVD decomposition of the Jacobian.

rank Numerical rank of the Jacobian.

identified Logical: are all parameters locally identified?

singular_values Vector of singular values.

strength Per-parameter identification strength (norm of corresponding Jacobian column).

condition_number Condition number of the Jacobian.

param_names Character vector of parameter names.

summary Data frame with per-parameter diagnostics.

Examples

```
m <- dsge_model(
  obs(y ~ z),
  state(z ~ rho * z),
  start = list(rho = 0.5)
)
set.seed(1)
z <- numeric(100); for (i in 2:100) z[i] <- 0.8*z[i-1]+rnorm(1)
fit <- estimate(m, data = data.frame(y = z))
id <- check_identification(fit)
print(id)
```

---

conditional_forecast *Conditional Forecast*

Description

Produces a dynamic forecast of a fitted DSGE model conditional on a user-specified path for one or more observable variables. Useful for policy scenario analysis (e.g. holding the policy rate fixed for k periods, or imposing an inflation path implied by a survey).

Usage

```
conditional_forecast(object, horizon = 12L, condition, ...)

## S3 method for class 'dsge_fit'
conditional_forecast(object, horizon = 12L, condition, ...)
```

Arguments

`object`	A `dsge_fit` object (returned by `estimate`).
`horizon`	Integer. Number of periods to forecast. Default 12.
`condition`	A named list. Each element corresponds to one observable variable and supplies the conditioning path as a numeric vector of length up to `horizon`. Use `NA` to leave a particular period unconditioned (the forecast value at that period is then determined endogenously).
`...`	Additional arguments (currently unused).

Details

The minimum-norm shock sequence is computed by solving

$$e^* = R'(RR')^{-1}(c - b)$$

where R is the stacked impulse-response matrix from each shock at each period to the conditioned variables, c is the vector of conditioning targets (de-meaned) and b is the unconditional forecast of those variables. A small Tikhonov regularisation is added to RR' for numerical stability when constraints are linearly dependent.

Value

An object of class `c("dsge_conditional_forecast", "dsge_forecast")` containing the same fields as `forecast.dsge_fit` plus the implied structural shock sequence (`shocks`), the conditioning input (`condition`), and a logical flag `conditioned` in the `forecasts` data frame indicating which (period, variable) pairs were constrained.

Because the result inherits from `dsge_forecast`, the existing `plot.dsge_forecast` method will display it with history and (point) forecast. Confidence bands are not currently computed for the conditional case.

References

Waggoner, D.F. and Zha, T. (1999). Conditional forecasts in dynamic multivariate models. *Review of Economics and Statistics*, 81(4), 639-651.

See Also

`forecast.dsge_fit` for unconditional forecasts.

Examples

```
nk <- dsge_model(
  obs(p   ~ beta * lead(p) + kappa * x),
  unobs(x ~ lead(x) - (r - lead(p) - g)),
  obs(r   ~ psi * p + u),
  state(u ~ rhou * u),
  state(g ~ rhog * g),
  fixed = list(beta = 0.99),
  start = list(kappa = 0.1, psi = 1.5, rhou = 0.7, rhog = 0.9)
)
sol <- solve_dsge(nk,
  params   = c(kappa = 0.1, psi = 1.5, rhou = 0.7, rhog = 0.9),
  shock_sd = c(e.u = 1.0, e.g = 0.5))
# Simulate data and fit
set.seed(1)
TT <- 100
xst <- matrix(0, TT, nrow(sol$H))
y   <- matrix(0, TT, nrow(sol$G))
for (t in 2:TT) {
  e <- rnorm(ncol(sol$M)) * c(1, 0.5)
  xst[t, ] <- as.numeric(sol$H %*% xst[t-1, ] + sol$M %*% e)
  y[t, ]   <- as.numeric(sol$G %*% xst[t, ])
}
colnames(y) <- rownames(sol$G)
dat <- as.data.frame(y[, nk$variables$observed, drop = FALSE])
fit <- estimate(nk, data = dat)

# Conditional forecast: hold r at 0 for the next 4 periods
cf <- conditional_forecast(fit, horizon = 12,
  condition = list(r = c(0, 0, 0, 0, rep(NA, 8))))
plot(cf)
```

conditional_forecast.dsge_dsgevar *Conditional Forecast for a DSGE-VAR Posterior*

Description

Produces forecasts from a DSGE-VAR posterior conditional on a user-specified path for a subset of variables. For each posterior draw of the VAR coefficients, the function applies an algorithm analogous to Waggoner-Zha (1999) at the VAR level: at each period the conditioning constraints pin down a minimum-norm sequence of VAR innovations.

Usage

```
## S3 method for class 'dsge_dsgevar'
conditional_forecast(object, horizon = 12L, condition, ...)

## S3 method for class 'dsge_dsgevar_mh'
conditional_forecast(object, horizon = 12L, condition, ...)
```

Arguments

`object`	A `dsge_dsgevar` or `dsge_dsgevar_mh` object.
`horizon`	Integer. Forecast horizon.
`condition`	A named list of numeric vectors (use `NA` for unconditioned periods). Names must match VAR variable names.
`...`	Additional arguments (currently unused).

Value

An object inheriting from `dsge_forecast` with posterior summary statistics of the conditional forecast.

discretionary_policy *Discretionary (Time-Consistent) Optimal Policy*

Description

Solves for the welfare-maximising policy under *discretion*: the planner has no commitment power and at every period re-optimises subject to private-sector expectations of the policy from then on. Implements the Soederlind (1999) / Dennis (2007) value-iteration algorithm.

Usage

```
discretionary_policy(
  model,
  params = NULL,
  shock_sd = NULL,
  instruments,
  welfare_weights,
  beta = 0.99,
  tol = 1e-10,
  max_iter = 2000L
)
```

Arguments

`model`	A `dsge_model` or `dsgenl_model` object.
`params`	Named numeric vector of model parameters.
`shock_sd`	Named numeric vector of shock standard deviations.
`instruments`	Character vector. Names of the policy instrument variables (subset of the model's control variables).
`welfare_weights`	A named list specifying the welfare loss weights. Elements: `Q_xx` Square matrix (n_s x n_s) on state variables. If a named vector is provided a diagonal matrix is constructed. `Q_yy` Square matrix (n_c x n_c) on control variables. If a named vector is provided a diagonal matrix is constructed. `Q_xy` Cross-weight matrix (n_s x n_c). Default is zero. At least one of `Q_xx` or `Q_yy` must be supplied.
`beta`	Numeric. Discount factor (0 < beta < 1). Default 0.99.
`tol`	Numeric. Convergence tolerance on $\|\|P_{k+1} - P_k\|\|$. Default 1e-10.
`max_iter`	Integer. Maximum fixed-point iterations. Default 2000.

Details

The discretionary solution always satisfies the time-consistency constraint: the planner cannot make credible promises that condition on past states beyond what is in the current state vector. For linear-quadratic problems whose forward-looking elements have been absorbed into the reduced-form transition matrix `H` by `solve_dsge()`, the discretionary fixed point coincides with the commitment fix point (Ramsey). The two solutions differ when the policy has direct access to forward-looking auxiliary variables not yet integrated out. Use `ramsey_policy()` for the commitment problem; the two functions take identical arguments.

Value

An object of class `c("dsge_discretionary","dsge_ramsey")` with the same fields as `ramsey_policy` – in particular `F` (feedback rule), `H_ram` (closed-loop transition under the optimal rule), `welfare_loss`, `P` (value matrix), `converged`, `n_iter`.

References

Soederlind, P. (1999). Solution and estimation of RE macromodels with optimal policy. *European Economic Review*, 43(4-6), 813-823.

Dennis, R. (2007). Optimal policy in rational expectations models: New solution algorithms. *Macroeconomic Dynamics*, 11(1), 31-55.

See Also

`ramsey_policy` (commitment), `osr` (restricted simple rules), `welfare_loss`.

`dsge_model` *Define a Linear DSGE Model*

Description

Constructs a linear DSGE model object from a set of equations. Each equation is wrapped in `obs()`, `unobs()`, or `state()` to indicate the role of the left-hand-side variable.

Usage

```
dsge_model(..., fixed = list(), start = list(), derived = NULL)
```

Arguments

`...`	Equation specifications created by `obs()`, `unobs()`, and `state()`.
`fixed`	Named list of parameter values to hold fixed (constrained) during estimation.
`start`	Named list of starting values for free parameters.
`derived`	Optional function mapping a named list of primitive parameter values to a named list of derived parameter values (e.g. `beta_bar = beta * gamma^(-sigma_c)`). Called by `solve_dsge()` at every solve so derived parameters track the current primitives during estimation. The returned list is concatenated with the primitives before equation coefficients are evaluated. Names returned by `derived()` may then appear directly inside equation coefficients alongside primitive parameters.

Details

A linear DSGE model is specified as a system of equations in which variables enter linearly but parameters may enter nonlinearly.

Use `lead(x)` or `E(x)` within formulas to denote the one-period-ahead model-consistent expectation of variable `x`.

The number of exogenous state variables (those with shocks) must equal the number of observed control variables.

Value

An object of class "dsge_model" containing:

equations List of parsed equation objects.

variables List with elements observed, unobserved, exo_state, endo_state.

parameters Character vector of all parameter names.

free_parameters Character vector of free (estimable) parameter names.

fixed Named list of fixed parameter values.

start Named list of starting values.

Examples

```
# Simple New Keynesian model
nk <- dsge_model(
  obs(p   ~ beta * lead(p) + kappa * x),
  unobs(x ~ lead(x) - (r - lead(p) - g)),
  obs(r   ~ psi * p + u),
  state(u ~ rhou * u),
  state(g ~ rhog * g),
  fixed = list(beta = 0.96),
  start = list(kappa = 0.1, psi = 1.5, rhou = 0.7, rhog = 0.9)
)
```

---

dsgenl_model *Define a Nonlinear DSGE Model*

Description

Constructs a nonlinear DSGE model from string-based equations. Each equation is a character string of the form "LHS = RHS". State equations are detected automatically when the left-hand side is of the form VAR(+1).

Usage

```
dsgenl_model(
  ...,
  observed = character(0),
  unobserved = character(0),
  exo_state,
  endo_state = character(0),
  fixed = list(),
  start = list(),
  ss_guess = NULL,
  ss_function = NULL
)
```

Arguments

`...`	Character strings, each defining one model equation. Use `VAR(+1)` to denote the one-period-ahead value of variable `VAR`.
`observed`	Character vector of observed control variable names.
`unobserved`	Character vector of unobserved control variable names. Default is `character(0)`.
`exo_state`	Character vector of exogenous state variable names. These have shocks attached. There must be at least as many exogenous states as observed controls (more observables than shocks would make the likelihood singular).
`endo_state`	Character vector of endogenous (predetermined) state variable names. These have no shocks. Default is `character(0)`.
`fixed`	Named list of parameter values to hold fixed during estimation.
`start`	Named list of starting values for free parameters.
`ss_guess`	Named numeric vector of initial guesses for steady-state solving. If `NULL`, defaults to 1 for all variables.
`ss_function`	Optional function that computes the steady state analytically. Must accept a named parameter vector and return a named numeric vector of steady-state variable values.

Details

Equations are written in standard mathematical notation with `=` as the equality sign and `VAR(+1)` for leads. For example:

```
"1/C = beta / C(+1) * (alpha * K^(alpha-1) + 1 - delta)"
```

State equations must have exactly one lead variable on the left-hand side (e.g., `"K(+1) = K^alpha - C + (1 - delta) * K"`). Control equations are all remaining equations.

Control equations must be provided in the order matching the controls vector (observed first, then unobserved).

Value

An object of class `"dsgenl_model"`.

Examples

```
# Simple RBC model
rbc <- dsgenl_model(
  "1/C = beta / C(+1) * (alpha * exp(Z) * K^(alpha-1) + 1 - delta)",
  "K(+1) = exp(Z) * K^alpha - C + (1 - delta) * K",
  "Z(+1) = rho * Z",
  observed = "C",
  endo_state = "K",
  exo_state = "Z",
  fixed = list(alpha = 0.33, beta = 0.99, delta = 0.025),
  start = list(rho = 0.9)
)
```

E — *Expectation Operator (Alias for lead)*

Description

A user-friendly alias for `lead(x, 1)`. Represents the one-period-ahead model-consistent expectation of variable `x`.

Usage

```
E(x)
```

Arguments

`x`	A variable name (unquoted) within a DSGE equation formula.

Details

`E(x)` is equivalent to `lead(x, 1)`. The parser translates `E(x)` to `lead(x, 1)` internally.

Value

This function is not meant to be called directly; it always throws an error. It is recognized as a syntactic marker by the equation parser inside `dsge_model()`.

See Also

`lead()`, `dsge_model()`

endogenous_prior — *Endogenous Prior on Model-Implied Moments*

Description

Constructs an endogenous prior (Christiano, Trabandt & Walentin 2011) that penalises parameter draws whose model-implied second moments differ from user-specified empirical targets. The endogenous prior adds a multivariate-Gaussian log-density on the moment vector

$$\log p_{endog}(\theta) = -\frac{1}{2}\big(m(\theta) - m^*\big)^\top V^{-1}\big(m(\theta) - m^*\big)$$

to the existing log-prior, where $m(\theta)$ is the model-implied moment vector and m^* the empirical target.

Usage

```
endogenous_prior(target, weight = NULL)
```

Arguments

`target` Named numeric vector of target moments to match. Names should look like "sd_<var>" (standard deviation of an observable) or "ac1_<var>" (lag-1 autocorrelation). See Details.

`weight` Optional positive-definite weighting matrix (V^{-1} above). If omitted, the inverse of `diag(0.1^2 * target^2)` is used (a 10\ moment).

Details

Supported moment specifications via name conventions:

- "sd_<var>": standard deviation of observable <var>.
- "ac1_<var>": lag-1 autocorrelation of observable <var>.
- "cor_<v1>_<v2>": contemporaneous correlation between observables <v1> and <v2>.

The closure builds the model-implied moments from the unconditional state covariance (solved via the doubling Lyapunov algorithm) and the observation equation.

Value

A list of class "dsge_endog_prior" containing:

`target` The supplied target moment vector.

`weight` The precision matrix used.

`moments_fn` A closure that takes a `dsge_solution` and returns the model-implied moment vector (same names as `target`).

`log_density` A closure that takes a `dsge_solution` and returns the endogenous-prior log density (a scalar; `-Inf` on failure).

Pass this object to `bayes_dsge(..., endogenous_prior = ...)`.

References

Christiano, L.J., Trabandt, M. and Walentin, K. (2011). Introducing financial frictions and unemployment into a small open economy model. *Journal of Economic Dynamics and Control*, 35(12), 1999-2041.

Examples

```
# Empirical SD and AR(1) of inflation and the policy rate
tgt <- c(sd_p = 0.5, sd_r = 0.6, ac1_p = 0.85, ac1_r = 0.95)
ep  <- endogenous_prior(tgt)
# Then in estimation:
# fit <- bayes_dsge(model, data, priors = ...,
#                   endogenous_prior = ep)
```

estimate *Estimate a Linear DSGE Model by Maximum Likelihood*

Description

Estimates the parameters of a linear DSGE model by maximizing the log-likelihood computed via the Kalman filter.

Usage

```
estimate(
  model,
  data,
  start = NULL,
  fixed = NULL,
  method = "BFGS",
  control = list(),
  shock_start = NULL,
  demean = TRUE,
  hessian = TRUE,
  presample = 0L
)
```

Arguments

`model`	A `dsge_model` object created by `dsge_model()`, a `dsgenl_model`, or a Dynare model imported with `read_dynare()`.
`data`	A data frame, matrix, or `ts` object containing the observed variables. Column names must match the observed control variable names in the model.
`start`	Named list of starting values for free parameters. Overrides any starting values specified in the model.
`fixed`	Named list of fixed parameter values. Overrides any fixed values specified in the model.
`method`	Optimization method passed to `stats::optim()`. Default is `"BFGS"`.
`control`	Control list passed to `stats::optim()`.
`shock_start`	Named numeric vector of starting values for shock standard deviations. If `NULL`, defaults are set based on data variability.
`demean`	Logical. If `TRUE` (default), observed variables are demeaned before estimation.
`hessian`	Logical. If `TRUE` (default), the Hessian is computed at the solution for standard errors.
`presample`	Integer. Number of initial observations used only to initialise the Kalman filter and excluded from the likelihood (as Dynare's `presample` option). Default 0; for a model imported with `read_dynare()`, the value in the file's `estimation` command.

Details

The estimator optimizes over the structural parameters and the log standard deviations of the shocks. Shock standard deviations are parameterized in log-space to ensure positivity.

If the optimizer encounters parameter values for which the model is not saddle-path stable, the log-likelihood is set to `-Inf`.

Value

An object of class `"dsge_fit"`.

Examples

```
# Define a simple AR(1) model
m <- dsge_model(
  obs(y ~ z),
  state(z ~ rho * z),
  start = list(rho = 0.5)
)

# Simulate some data
set.seed(42)
e <- rnorm(200)
z <- numeric(200)
for (i in 2:200) z[i] <- 0.8 * z[i-1] + e[i]
dat <- data.frame(y = z)

fit <- estimate(m, data = dat)
summary(fit)
```

`extended_path` *Stochastic Simulation via the Extended Path*

Description

Simulates a DSGE model under stochastic shocks using the Fair-Taylor / Adjemian-Juillard extended path: at every period the model is solved under perfect foresight conditional on the realised current shock and zero expected future shocks, the period's value is recorded, and the system advances. Used heavily when nonlinearities matter and pruning is unsatisfactory.

Usage

```
extended_path(
  x,
  periods = 200L,
  shock_sd = NULL,
  params = NULL,
```

```
  pf_horizon = 30L,
  initial = NULL,
  burn = 0L,
  seed = NULL
)
```

Arguments

x	A dsge_solution (linear) or dsgenl_model (nonlinear) object.
periods	Integer. Number of simulation periods. Default 200.
shock_sd	Named numeric vector of shock standard deviations (required for dsgenl_model; taken from the solution for dsge_solution).
params	Named numeric vector of parameter values (required for dsgenl_model).
pf_horizon	Integer. Inner perfect-foresight horizon used at every step. Larger values increase accuracy at the cost of speed. Default 30.
initial	Optional named numeric vector of initial state deviations from steady state.
burn	Integer. Number of initial periods to discard before returning the simulation (warm-up). Default 0.
seed	Optional integer seed.

Value

An object of class "dsge_extended_path" with elements:

states (periods x n_states) matrix of state deviations.

controls (periods x n_controls) matrix of control deviations.

shocks (periods x n_shocks) matrix of structural shock realisations.

periods, pf_horizon, model, solution Inputs.

Examples

```
rbc <- dsgenl_model(
  "1/C = beta / C(+1) * (alpha * exp(Z) * K^(alpha-1) + 1 - delta)",
  "K(+1) = exp(Z) * K^alpha - C + (1 - delta) * K",
  "Z(+1) = rho * Z",
  observed = "C", endo_state = "K", exo_state = "Z",
  fixed = list(alpha = 0.33, beta = 0.99, delta = 0.025),
  start = list(rho = 0.9))
sim <- extended_path(rbc, periods = 100,
                     params = c(rho = 0.9), shock_sd = c(Z = 0.01),
                     pf_horizon = 30, seed = 1)
head(sim$controls)
```

fitted.dsge_fit — *Fitted values from a DSGE model*

Description

Returns filtered fitted values of observed variables (using all information up to and including time t).

Usage

```
## S3 method for class 'dsge_fit'
fitted(object, ...)
```

Arguments

`object`	A `"dsge_fit"` object.
`...`	Additional arguments (currently unused).

Value

A matrix of fitted values with columns named by observed variables.

forecast — *Forecast from a DSGE Model*

Description

Generic function for forecasting from estimated DSGE models.

Usage

```
forecast(object, ...)
```

Arguments

`object`	A fitted model object.
`...`	Additional arguments passed to methods.

Value

A forecast object.

See Also

`forecast.dsge_fit()`

forecast.dsge_dsgevar *Forecasts from a DSGE-VAR Posterior*

Description

Produces unconditional fan-chart forecasts from a DSGE-VAR posterior returned by `bayes_dsge_var` or `bayes_dsge_var_mh`. For each posterior draw of the VAR coefficients and innovation covariance, the function iterates the VAR forward for `horizon` periods, drawing innovations from the posterior $N(0, \Sigma)$; quantiles across draws give the fan chart.

Usage

```
## S3 method for class 'dsge_dsgevar'
forecast(object, horizon = 12L, n_paths = 1L, ...)

## S3 method for class 'dsge_dsgevar_mh'
forecast(object, horizon = 12L, n_paths = 1L, ...)
```

Arguments

`object`	A `dsge_dsgevar` or `dsge_dsgevar_mh` object.
`horizon`	Integer. Forecast horizon. Default 12.
`n_paths`	Integer. Number of forecast paths to simulate per posterior draw (each draw uses fresh innovation shocks). Default 1.
`...`	Additional arguments (currently unused).

Value

An object of class `c("dsge_dsgevar_forecast", "dsge_forecast")` containing:

forecasts Tidy data frame (period, variable, value, sd) reporting the posterior mean and standard deviation of the forecast at each (variable, horizon).

forecast_paths Array (horizon x n_var x total_paths) of individual simulated forecast paths.

history In-sample data (un-demeaned) for plotting.

horizon, var_names Inputs.

forecast.dsge_fit *Forecast from a Fitted DSGE Model*

Description

Produces dynamic multi-step forecasts from a fitted DSGE model. Forecasts are generated by iterating the state-space solution forward from the last filtered state, with forecast-error variance computed analytically by iterating the state covariance.

Usage

```
## S3 method for class 'dsge_fit'
forecast(object, horizon = 12L, ...)
```

Arguments

`object`	A `dsge_fit` object.
`horizon`	Integer. Number of periods to forecast ahead. Default is 12.
`...`	Additional arguments (currently unused).

Value

An object of class `"dsge_forecast"` containing:

forecasts Data frame with columns `period`, `variable`, `value`, and `sd` (one-step forecast standard deviation at each horizon). Use `value +/- qnorm(0.5+level/2)*sd` to construct a confidence band at any level.

horizon The forecast horizon.

states Matrix of forecasted state vectors.

obs_matrix Forecast point estimates in matrix form.

obs_sd Matrix of forecast standard deviations (same shape as `obs_matrix`).

history Matrix of in-sample observed data (un-demeaned) for use in plotting forecasts alongside history.

geweke_test *Geweke convergence diagnostic*

Description

Computes Geweke's (1992) convergence diagnostic, which compares the means of the first and last portions of each chain using a z-test.

Usage

```
geweke_test(object, ...)
```

Arguments

`object`	A `"dsge_bayes"` object.
`...`	Additional arguments passed to methods (e.g., `frac1`, `frac2` for chain window fractions).

Value

An object of class `"dsge_geweke"` with z-scores and p-values for each parameter and chain.

`global_sensitivity` *Global Sensitivity Analysis of a DSGE Model*

Description

Quantifies how much each free parameter contributes to variation in a chosen scalar model output, sampling from the prior space. Implements two complementary methods:

- **Sobol' (Saltelli 2010)**: estimates first-order and total-effect variance-based sensitivity indices. More expensive but quantitatively interpretable as "share of output variance attributable to parameter i".
- **Morris (1991) elementary effects**: cheaper "screening" method that produces μ^* (mean absolute elementary effect, ranking importance) and σ (spread, measuring nonlinearity / interactions).

Usage

```
global_sensitivity(
  model,
  priors,
  target,
  method = c("sobol", "morris"),
  n_samples = 200L,
  horizon = 20L,
  data = NULL,
  seed = NULL
)
```

Arguments

model	A dsge_model object.
priors	Named list of dsge_prior objects, one per free structural parameter (use the same priors you would for Bayesian estimation). Shock SDs default to inv_gamma(0.1, 2) if not supplied; pass them under names "sd_e.<shock>" to override.
target	A character string naming the scalar output to analyse: • "sd:<var>" – model-implied unconditional standard deviation of observable <var>. • "cor:<v1>:<v2>" – model-implied unconditional correlation between <v1> and <v2>. • "irf_max:<shock>:<response>" – maximum absolute impulse response of <response> to <shock> over horizon periods. • "loglik" – Kalman-filter log likelihood (requires data).
method	"sobol" (default) or "morris".
n_samples	Integer. Number of base samples (Sobol uses n_samples * (k + 2) model solves total; Morris uses n_samples * (k + 1)). Default 200.
horizon	Integer. IRF horizon when target starts with "irf_max:". Default 20.
data	Required when target = "loglik".
seed	Optional integer seed.

Value

An object of class "dsge_global_sensitivity" with the computed indices and metadata. For Sobol the key fields are S_first (first-order) and S_total (total-effect); for Morris, mu_star and sigma. All indices are named by parameter.

References

Saltelli, A. et al. (2010). Variance based sensitivity analysis of model output. *Computer Physics Communications*, 181: 259-270.

Morris, M.D. (1991). Factorial sampling plans for preliminary computational experiments. *Technometrics*, 33(2): 161-174.

Examples

```
nk <- dsge_model(
  obs(p ~ beta * lead(p) + kappa * x),
  unobs(x ~ lead(x) - (r - lead(p) - g)),
  obs(r ~ psi * p + u),
  state(u ~ rhou * u),
  state(g ~ rhog * g),
  fixed = list(beta = 0.99),
  start = list(kappa = 0.1, psi = 1.5, rhou = 0.7, rhog = 0.9))
priors <- list(
  kappa = prior("beta",   shape1 = 2,   shape2 = 8),
  psi   = prior("normal", mean   = 1.5, sd = 0.25),
```

```
  rhou  = prior("beta",   shape1 = 5,   shape2 = 2),
  rhog  = prior("beta",   shape1 = 5,   shape2 = 2))
gs <- global_sensitivity(nk, priors, target = "sd:p",
                         method = "sobol", n_samples = 100, seed = 1)
print(gs)
```

`gmm_estimate` *Generalised Method of Moments (GMM) Estimation*

Description

Estimates the free parameters of a linear DSGE model by matching *model-implied* moments to *empirical* moments from observed data. Model-implied moments are computed analytically via the Lyapunov solution of the state covariance.

Usage

```
gmm_estimate(
  model,
  data,
  moments,
  params_start,
  shock_sd_start,
  weight = NULL,
  lower = NULL,
  upper = NULL,
  method = "Nelder-Mead",
  control = list()
)
```

Arguments

`model`	A `dsge_model` object.
`data`	Matrix or data frame of observed variables; column names must match a subset of the model's observables.
`moments`	Character vector naming the moments to match. Each element uses the same naming convention as `endogenous_prior`: • `"sd:<var>"` – standard deviation of `<var>`. • `"var:<var>"` – variance of `<var>`. • `"cov:<v1>:<v2>"` – covariance between `<v1>` and `<v2>`. • `"ac1:<var>"` – lag-1 autocorrelation of `<var>`. • `"cor:<v1>:<v2>"` – contemporaneous correlation.
`params_start`	Named numeric vector of starting values for the structural parameters.
`shock_sd_start`	Named numeric vector of starting values for shock standard deviations.

weight	Optional positive-definite weighting matrix. Default identity (one-step GMM). Pass "optimal" to run a two-step estimator where the second-step weight is the inverse Newey-West covariance of the empirical moments.
lower, upper	Bounds (-Inf, Inf by default).
method	Optimiser method for optim. Default "Nelder-Mead".
control	optim control list.

Value

An object of class "dsge_gmm".

References

Hansen, L.P. (1982). Large sample properties of generalized method of moments estimators. *Econometrica*, 50(4), 1029-1054.

Examples

```
nk <- dsge_model(
  obs(p ~ beta * lead(p) + kappa * x),
  unobs(x ~ lead(x) - (r - lead(p) - g)),
  obs(r ~ psi * p + u),
  state(u ~ rhou * u),
  state(g ~ rhog * g),
  fixed = list(beta = 0.99),
  start = list(kappa = 0.1, psi = 1.5, rhou = 0.7, rhog = 0.9))
sol <- solve_dsge(nk,
  params   = c(kappa = 0.1, psi = 1.5, rhou = 0.7, rhog = 0.9),
  shock_sd = c(e.u = 1, e.g = 0.5))
set.seed(1)
H <- sol$H; G <- sol$G; M <- sol$M
TT <- 200
xst <- matrix(0, TT, nrow(H))
y_obs <- matrix(0, TT, nrow(G))
for (t in 2:TT) {
  e <- rnorm(ncol(M)) * c(1, 0.5)
  xst[t, ] <- as.numeric(H %*% xst[t-1, ] + M %*% e)
  y_obs[t, ] <- as.numeric(G %*% xst[t, ])
}
colnames(y_obs) <- rownames(G)
y_obs <- as.data.frame(y_obs[, nk$variables$observed])
est <- gmm_estimate(nk, y_obs,
  moments = c("sd:p","sd:r","ac1:p","ac1:r"),
  params_start   = c(kappa = 0.1, psi = 1.5, rhou = 0.7, rhog = 0.9),
  shock_sd_start = c(e.u = 1.0, e.g = 0.5),
  control = list(maxit = 100))
print(est)
```

irf.dsge_bayes *Compute Impulse-Response Functions*

Description

Computes the impulse-response functions (IRFs) from a fitted or solved DSGE model. An IRF traces the dynamic response of control and state variables to a one-standard-deviation shock.

Usage

```
## S3 method for class 'dsge_bayes'
irf(
  x,
  periods = 20L,
  impulse = NULL,
  response = NULL,
  se = TRUE,
  level = 0.95,
  n_draws = 200L,
  ...
)

irf(
  x,
  periods = 20L,
  impulse = NULL,
  response = NULL,
  se = TRUE,
  level = 0.95,
  ...
)
```

Arguments

`x`	A `dsge_fit` or `dsge_solution` object.
`periods`	Integer. Number of periods to compute. Default is 20.
`impulse`	Character vector of shock names. If `NULL` (default), computes IRFs for all shocks.
`response`	Character vector of variable names. If `NULL` (default), computes responses for all variables.
`se`	Logical. If `TRUE` (default) and `x` is a `dsge_fit`, compute delta-method standard errors for IRFs.
`level`	Confidence level for bands. Default is 0.95.
`n_draws`	Integer. Number of posterior draws to use for IRF computation. Default is 200. Set to `NULL` to use all draws.
`...`	Additional arguments passed to methods.

Value

An object of class "dsge_irf" containing a data frame with columns: period, impulse, response, value, and optionally se, lower, upper.

Methods (by class)

- irf(dsge_bayes): Compute posterior IRFs from a Bayesian DSGE fit. Returns pointwise posterior median and credible bands.

Examples

```
m <- dsge_model(
  obs(y ~ z),
  state(z ~ rho * z),
  start = list(rho = 0.5)
)
sol <- solve_dsge(m, params = c(rho = 0.8))
irfs <- irf(sol, periods = 10)
```

---

irf_2nd_order *Generalized IRFs Using Second-Order Approximation*

Description

Computes impulse-response functions using the second-order solution. These differ from first-order IRFs because responses depend on the initial state and shock sign.

Usage

```
irf_2nd_order(sol, shock, size = 1, periods = 40L, initial = NULL)
```

Arguments

sol	A dsge_solution object with order = 2.
shock	Character. Name of the shock.
size	Numeric. Shock size in standard deviations. Default 1.
periods	Integer. Number of IRF periods. Default 40.
initial	Named numeric vector of initial state deviations. Default is zero (ergodic mean under second-order).

Value

A data frame with columns: period, variable, response, type.

irf_match *Impulse-Response Matching Estimation*

Description

Estimates the free parameters of a linear DSGE model by minimising the weighted squared distance between the model's impulse responses and a user-supplied target (e.g. impulse responses estimated from a VAR or local projections).

Usage

```
irf_match(
  model,
  params_start,
  shock_sd_start,
  target,
  shock_sd_fixed = NULL,
  weight = NULL,
  lower = NULL,
  upper = NULL,
  method = "Nelder-Mead",
  control = list(),
  penalty = 1e+10
)
```

Arguments

`model`	A `dsge_model` object.
`params_start`	Named numeric vector of starting values for the structural parameters to estimate.
`shock_sd_start`	Named numeric vector of starting values for the shock standard deviations to estimate. Pass an empty vector (`c()`) to fix all shock SDs at the values given in `shock_sd_fixed`.
`target`	A data frame with columns `impulse`, `response`, `period` and `value` giving the empirical / target impulse responses to match. Each row is a single (impulse, response, period) datum. Use the same names that the model uses for shocks and variables. The format matches what `irf()$data` returns, so a reference solution's IRF can be passed directly.
`shock_sd_fixed`	Named numeric vector of shock SDs to keep fixed while estimating. Their values come from this argument. The full vector passed to `solve_dsge()` is the union of `shock_sd_start` (estimated) and `shock_sd_fixed`.
`weight`	Optional $N \times N$ positive-definite weighting matrix where N is the number of rows in `target`. If `NULL` (default), the identity is used (equally-weighted distance).

`lower`, `upper`	Numeric vectors of bounds (length = `length(params_start)` + `length(shock_sd_start)`). Default `-Inf` / `Inf`.
`method`	Optimisation method for `optim`. Default `"Nelder-Mead"`. Use `"L-BFGS-B"` for box constraints.
`control`	List of control arguments for `optim`.
`penalty`	Penalty value returned when the candidate parameters yield an unstable / non-existent solution. Default `1e10`.

Details

The objective stacks all (impulse, response, period) rows of `target` into a vector and computes

$$(\mathrm{irf}_{\mathrm{model}}(\theta) - \mathrm{irf}_{\mathrm{target}})^\top W (\mathrm{irf}_{\mathrm{model}}(\theta) - \mathrm{irf}_{\mathrm{target}}).$$

If the candidate parameters make the model unstable or fail to solve, `penalty` is returned (effectively rejecting that vector).

The asymptotic variance of the IRF-matching estimator is

$$(J'WJ)^{-1} J'W\Omega W'J(J'WJ)^{-1}$$

where J is the Jacobian of model IRFs at the optimum and Ω is the variance of the target IRFs. Computing this efficiently requires user knowledge of Ω; this function returns the point estimates only.

Value

An object of class `"dsge_irf_match"` containing:

`params` Estimated structural parameter values.

`shock_sd` Estimated shock standard deviations.

`objective` Achieved minimum objective value.

`converged` Logical (`optim` convergence == 0).

`n_iter` Iteration counts returned by `optim`.

`target` The supplied target IRF data frame, augmented with the model-implied fitted values at the optimum (column `fitted`).

`solution` The `dsge_solution` at the optimum.

References

Christiano, L.J., Eichenbaum, M. and Evans, C.L. (1999). Monetary policy shocks: What have we learned and to what end? In *Handbook of Macroeconomics, Volume 1A*, ch. 2.

Examples

```
nk <- dsge_model(
  obs(p   ~ beta * lead(p) + kappa * x),
  unobs(x ~ lead(x) - (r - lead(p) - g)),
  obs(r   ~ psi * p + u),
  state(u ~ rhou * u),
  state(g ~ rhog * g),
  fixed = list(beta = 0.99),
  start = list(kappa = 0.1, psi = 1.5, rhou = 0.7, rhog = 0.9)
)

# Build a target IRF from a "true" parameterisation
sol_true <- solve_dsge(nk,
  params   = c(kappa = 0.1, psi = 1.5, rhou = 0.7, rhog = 0.9),
  shock_sd = c(e.u = 1.0, e.g = 0.5))
target_df <- irf(sol_true, periods = 12)$data

# Estimate, starting away from the truth
est <- irf_match(nk,
  params_start   = c(kappa = 0.2, psi = 2.0, rhou = 0.5, rhog = 0.5),
  shock_sd_start = c(e.u = 1.0, e.g = 1.0),
  target         = target_df)
print(est)
```

---

kalman_filter_skewed *Skewed Kalman Filter for Skew-Normal Structural Shocks*

Description

Evaluates the log-likelihood of a linear state-space DSGE model whose structural shocks are skew-normal rather than Gaussian. The mean and covariance recursions are the usual Kalman ones (which remain the optimal linear filter under any shock distribution); in addition the filter propagates the third cumulant of the state exactly, and uses it to build a skewed one-step-ahead predictive density.

Usage

```
kalman_filter_skewed(y, G, H, M, D, shock_skew)
```

Arguments

y	Matrix of observed data (T x n_obs).
G	Policy matrix mapping states to controls.
H	State transition matrix.
M	Shock impact matrix (already scaled by the shock standard deviations, as returned by solve_dsge).

D — Observation selection matrix; the observation equation is $y_t = DGx_t$.

shock_skew — Numeric vector of shock skewness coefficients, one per column of M, each in $(-0.99, 0.99)$. A value of 0 means that shock is Gaussian. A scalar is recycled across shocks.

Details

What is exact and what is approximate: The filtered and predicted means, covariances, and third cumulants are computed *exactly* for the linear system: the Kalman gain is the exact minimum-MSE linear filter, and the third cumulant obeys the exact linear-map rule. The only approximation is in the shape of the predictive density used for the likelihood, which is taken to be a product of univariate skew-normals in the Cholesky-decorrelated space, each matched to the exact marginal skewness. This is exact when the shocks are Gaussian and is a third-order-accurate approximation otherwise.

Skew-normal parameterisation: Each shock is standardised to zero mean and unit variance, so shock_skew is the coefficient of skewness. The largest magnitude attainable by a skew-normal is about 0.995; values are clamped to ± 0.99.

Value

A list with the same components as the Gaussian filter – loglik, filtered_states, predicted_states, prediction_errors, filtered_P, innovation_var, predicted_obs – plus:

innovation_skew (T x n_obs) matrix of model-implied skewness coefficients of the decorrelated one-step-ahead forecast errors.

shock_skew The skewness vector used.

See Also

particle_filter for a fully nonlinear / non-Gaussian alternative.

Examples

```
m <- dsge_model(
  obs(y ~ z),
  state(z ~ rho * z),
  fixed = list(rho = 0.8))
sol <- solve_dsge(m, params = c(rho = 0.8), shock_sd = c(z = 1))
set.seed(1)
dat <- matrix(rnorm(100), 100, 1, dimnames = list(NULL, "y"))
# Gaussian shocks: identical to the standard filter
kalman_filter_skewed(dat, sol$G, sol$H, sol$M, sol$D,
                     shock_skew = 0)$loglik
# Left-skewed shocks
kalman_filter_skewed(dat, sol$G, sol$H, sol$M, sol$D,
                     shock_skew = -0.7)$loglik
```

lead — *Forward Lead Operator for DSGE Equations*

Description

Marks a variable as a one-period-ahead model-consistent expectation in a DSGE equation formula. This is the core primitive for forward-looking variables.

Usage

```
lead(x, k = 1L)
```

Arguments

x	A variable name (unquoted) within a DSGE equation formula.
k	Integer lead horizon. Currently only k = 1 is supported. Default is 1.

Details

`lead(x)` in a DSGE equation represents the expectation of variable `x` one period ahead, conditional on the model. In the literature, this corresponds to $E_t[x_{t+1}]$.

This function is not meant to be called directly. It is recognized by the equation parser inside `dsge_model()`.

Value

This function is not meant to be called directly; it always throws an error. It is recognized as a syntactic marker by the equation parser inside `dsge_model()`.

See Also

`E()` for a user-friendly alias, `dsge_model()`

linearize — *Linearize a Nonlinear DSGE Model*

Description

Computes a first-order Taylor expansion of the nonlinear model around its deterministic steady state and returns the structural matrices in the canonical linear form.

Usage

```
linearize(model, steady_state, params = NULL)
```

Arguments

`model`	A `dsgenl_model` object.
`steady_state`	A `dsgenl_steady_state` object or a named numeric vector of steady-state values.
`params`	Named numeric vector of parameter values. If `NULL` and `steady_state` is a `dsgenl_steady_state` object, uses the parameters stored there.

Value

A list of structural matrices (A0, A1, A2, A3, A4, B0, B1, B2, B3, C, D) plus the steady-state values. A4 captures lead-state coefficients in control equations (often zero).

`marginal_likelihood` *Marginal likelihood estimation*

Description

Estimates the log marginal likelihood using the modified harmonic mean estimator (Geweke, 1999). This provides a practical Bayesian model comparison tool via Bayes factors: BF12 = exp(logML1 - logML2).

Usage

```
marginal_likelihood(object, ...)
```

Arguments

`object`	A `"dsge_bayes"` object.
`...`	Additional arguments passed to methods (e.g., `method`, `tau`).

Details

The harmonic mean estimator is known to be numerically unstable in some cases. The modified version (with truncation parameter tau) reduces this instability. Results should be interpreted with caution and compared across models only when both use similar MCMC settings.

Value

An object of class `"dsge_marginal_likelihood"`.

mcmc_diagnostics *MCMC diagnostic summary*

Description

Comprehensive MCMC diagnostic summary combining ESS, R-hat, Geweke, and acceptance rate information.

Usage

```
mcmc_diagnostics(object, ...)
```

Arguments

object	A "dsge_bayes" object.
...	Additional arguments passed to geweke_test().

Value

An object of class "dsge_mcmc_summary".

model_covariance *Model-implied covariance and correlation matrices*

Description

Computes the unconditional (model-implied) covariance and correlation matrices of observable variables from a solved or estimated DSGE model. These are the theoretical second moments implied by the model at the given parameter values.

Usage

```
model_covariance(x, variables = NULL, n_lags = 0L, ...)
```

Arguments

x	A fitted model ("dsge_fit", "dsge_bayes") or a solved model ("dsge_solution").
variables	Character vector of variable names to include. Default NULL returns all observable variables.
n_lags	Integer. If positive, also compute autocovariances at lags 1, ..., n_lags. Default 0 (contemporaneous only).
...	Additional arguments (currently unused).

Value

An object of class "dsge_covariance" containing:

covariance Covariance matrix of selected variables.

correlation Correlation matrix of selected variables.

std_dev Standard deviations (square root of diagonal).

autocovariances List of lagged autocovariance matrices (empty if n_lags = 0).

variables Variable names.

n_lags Number of autocovariance lags computed.

Examples

```
mod <- dsge_model(
  obs(pi ~ beta * lead(pi) + kappa * x),
  unobs(x ~ lead(x) - (r - lead(pi) - g)),
  obs(r ~ psi * pi + u),
  state(u ~ rhou * u),
  state(g ~ rhog * g),
  fixed = list(beta = 0.99),
  start = list(kappa = 0.1, psi = 1.5, rhou = 0.5, rhog = 0.5)
)
p <- list(kappa = 0.1, psi = 1.5, rhou = 0.5, rhog = 0.5)
s <- c(u = 0.5, g = 0.5)
sol <- solve_dsge(mod, params = p, shock_sd = s)
model_covariance(sol)
```

---

model_latex *Export DSGE Model Equations to LaTeX*

Description

Renders the equations of a dsge_model or dsgenl_model as LaTeX source, suitable for pasting into a paper or writing straight to a .tex file. Parameter names that match Greek letters are converted automatically (beta becomes \beta), variables receive time subscripts, and leads are wrapped in a conditional expectation operator.

Usage

```
model_latex(
  model,
  file = NULL,
  env = c("align", "gather", "none"),
  numbered = TRUE,
  greek = TRUE,
  standalone = FALSE,
  labels = FALSE
)
```

Arguments

model	A dsge_model or dsgenl_model object.
file	Optional path to write the LaTeX source to. If NULL (default) the source is returned invisibly and printed to the console.
env	Character. LaTeX environment to wrap the system in. One of "align" (default), "gather", or "none" (bare equation lines with no wrapper).
numbered	Logical. Use the numbered environment (align) rather than the starred one (align*). Default TRUE.
greek	Logical. Substitute Greek-letter parameter names with their LaTeX commands. Default TRUE.
standalone	Logical. Wrap the output in a minimal complete LaTeX document (with \documentclass and amsmath) so the file compiles on its own. Default FALSE.
labels	Logical. Emit a \label{eq:<lhs>} on each equation so they can be cross-referenced. Only used when numbered = TRUE. Default FALSE.

Details

Rendering conventions:

- Variables carry a time subscript: y renders as y_t.
- Leads render inside a conditional expectation: lead(y) becomes $\mathbb{E}_t y_{t+1}$.
- Parameter names matching a Greek letter are converted (sigma to \sigma); a trailing _bar becomes an overbar and a trailing _ss a steady-state superscript.
- Divisions render as \frac, powers as superscripts, and multiplication as juxtaposition.

Value

A character vector of LaTeX source lines, returned invisibly. When file is supplied the same lines are written there.

Examples

```
nk <- dsge_model(
  obs(pi ~ beta * lead(pi) + kappa * x),
  unobs(x ~ lead(x) - (r - lead(pi) - g)),
  obs(r ~ psi * pi + u),
  state(u ~ rhou * u),
  state(g ~ rhog * g),
  fixed = list(beta = 0.99),
  start = list(kappa = 0.1, psi = 1.5, rhou = 0.7, rhog = 0.9))
tex <- model_latex(nk)
cat(head(tex, 4), sep = "\n")
```

`ms_filter` *Markov-Switching Volatility Filter (Kim 1994)*

Description

Evaluates the log-likelihood of a linear DSGE model whose structural shock volatilities switch between a small number of regimes governed by a first-order Markov chain, and returns filtered and smoothed regime probabilities.

Usage

```
ms_filter(
  y,
  G,
  H,
  M,
  D,
  regime_scale,
  P_trans,
  initial_probs = NULL,
  smooth = TRUE
)
```

Arguments

`y`	Matrix of observed data (T x n_obs).
`G`, `H`, `M`, `D`	Solution matrices from `solve_dsge`. At first order these are regime-invariant, so a single solve suffices.
`regime_scale`	Volatility multipliers applied to the shock loading `M` in each regime. Either a length-K numeric vector (the same multiplier for every shock) or a $K \times n_{shocks}$ matrix (a separate multiplier per shock). Regime 1 is conventionally the low-volatility regime, but no ordering is imposed.
`P_trans`	$K \times K$ Markov transition matrix, where `P_trans[i, j]` is the probability of moving from regime i to regime j. Rows must sum to one.
`initial_probs`	Optional length-K vector of initial regime probabilities. Defaults to the ergodic distribution implied by `P_trans`.
`smooth`	Logical. Also compute Kim-smoothed regime probabilities using the full sample. Default `TRUE`.

Details

Why the solution does not switch: A first-order perturbation solution is certainty-equivalent: the decision rules depend on the model's structural parameters but not on the variances of the shocks. Switching *volatility* therefore leaves G and H unchanged and rescales only the shock loading, which is what makes this specification exactly (rather than approximately) solvable. Switching *structural* parameters is a different and much harder problem, because agents' expectations must then account for the possibility of future regime changes.

Value

An object of class "dsge_ms_filter" with elements:

loglik Log-likelihood of the data.

filtered_probs (T x K) matrix of $P(s_t = j \mid y_{1:t})$.

predicted_probs (T x K) matrix of $P(s_t = j \mid y_{1:t-1})$.

smoothed_probs (T x K) matrix of $P(s_t = j \mid y_{1:T})$, or NULL when smooth = FALSE.

filtered_states (T x n_states) regime-averaged filtered state means.

regime_scale, P_trans, ergodic Inputs and the implied ergodic distribution.

See Also

kalman_filter_skewed for non-Gaussian shocks.

Examples

```
m <- dsge_model(
  obs(y ~ z),
  state(z ~ rho * z),
  fixed = list(rho = 0.7))
sol <- solve_dsge(m, params = c(rho = 0.7), shock_sd = c(z = 1))
set.seed(1)
dat <- matrix(rnorm(150), 150, 1, dimnames = list(NULL, "y"))
P <- matrix(c(0.95, 0.05, 0.10, 0.90), 2, 2, byrow = TRUE)
out <- ms_filter(dat, sol$G, sol$H, sol$M, sol$D,
                 regime_scale = c(1, 3), P_trans = P)
out$loglik
head(out$filtered_probs)
```

---

obc_constraint *Create an Occasionally Binding Constraint*

Description

Specifies an inequality constraint for use with simulate_occbin.

Usage

```
obc_constraint(variable, type = ">=", bound = 0, shock = NULL)
```

Arguments

`variable`	Character. Name of the constrained variable (must be a control variable in the model).
`type`	Character. Either `">="` or `"<="`.
`bound`	Numeric. The constraint bound. For models with steady state, this is in levels; for linear models, in deviations.
`shock`	Character or `NULL`. Name of the shock channel used to enforce the constraint. If `NULL` (default), auto-detected as the shock with the largest impact on the constrained variable.

Value

An `obc_constraint` object.

Examples

```
# Zero lower bound on nominal interest rate
obc_constraint("r", ">=", 0)

# Upper bound on debt ratio
obc_constraint("b", "<=", 0.6)
```

---

obs — *Define an Observed Control Variable Equation*

Description

Wraps a formula to mark it as an equation for an observed control variable in a linear DSGE model.

Usage

```
obs(formula)
```

Arguments

`formula`	A formula of the form `variable ~ expression`.

Value

A list with class `"dsge_equation"` containing the parsed equation and its type.

See Also

`unobs()`, `state()`, `dsge_model()`

osr *Optimal Simple (Restricted) Policy Rules*

Description

Optimises the coefficients of a user-specified policy rule to minimise an unconditional quadratic welfare loss, subject to the model's rational expectations equilibrium.

Usage

```
osr(
  model,
  params,
  shock_sd,
  osr_params,
  welfare_weights,
  lower = NULL,
  upper = NULL,
  method = "L-BFGS-B",
  control = list(),
  penalty = 1e+10
)
```

Arguments

`model` A `dsge_model` or `dsgenl_model` object, or a Dynare model imported with `read_dynare` that declares `osr_params` and `optim_weights` (then `params`, `shock_sd`, `osr_params`, `welfare_weights` and the bounds default to the values in the file).

`params` Named numeric vector of model parameter values. Must contain entries for every free parameter and structural parameter that `solve_dsge` requires. Values for any parameters listed in `osr_params` will be overwritten during the search.

`shock_sd` Named numeric vector of shock standard deviations.

`osr_params` Named numeric vector. Names are the parameters to optimise (must be in `names(params)`), values are starting guesses for the optimiser.

`welfare_weights` A named list with elements `Q_xx`, `Q_yy` and optionally `Q_xy`, defining the loss

$$L = \mathrm{tr}(Q_{xx}\Sigma_x) + \mathrm{tr}(Q_{yy}\Sigma_y) + 2\,\mathrm{tr}(Q_{xy}\Sigma'_{xy}).$$

Each weight may be supplied either as a square matrix (rows/columns in state/control order) or as a named numeric vector (interpreted as the diagonal of the corresponding matrix).

`lower, upper` Numeric vectors of lower/upper bounds for the parameters being optimised. Default `NULL` uses `c(-Inf, Inf)` for each, in which case `method` should not be `"L-BFGS-B"`.

`method` Optimisation method passed to `optim`. Default `"L-BFGS-B"`.

control	List of control options passed to optim.
penalty	Numeric. Penalty value returned when the candidate parameter vector yields a non-existent or unstable equilibrium. Default 1e10.

Details

Comparison to ramsey_policy(): ramsey_policy() computes the fully flexible commitment-optimal policy: it is the lower bound on welfare loss subject only to the model's equilibrium conditions. osr() restricts the policy to a parametric family (e.g. a Taylor rule $r_t = \phi_\pi \pi_t + \phi_y x_t$) and optimises within that family. The gap between Ramsey and OSR loss is the welfare cost of restricting to simple rules.

Value

An object of class "dsge_osr" containing:

optimal Named numeric vector of optimal parameter values.

loss Achieved welfare loss at the optimum.

loss_at_start Loss at the starting values, for comparison.

converged Logical (TRUE if optim returned convergence == 0).

n_iter Iteration counts reported by optim.

params Full parameter vector at the optimum.

weights The welfare weight matrices used (with rows/cols named).

solution The dsge_solution object at the optimum.

message Optional diagnostic message from optim.

See Also

ramsey_policy, welfare_loss

Examples

```
nk <- dsge_model(
  obs(p   ~ beta * lead(p) + kappa * x),
  unobs(x ~ lead(x) - (r - lead(p) - g)),
  obs(r   ~ psi * p + u),
  state(u ~ rhou * u),
  state(g ~ rhog * g),
  fixed = list(beta = 0.99),
  start = list(kappa = 0.1, psi = 1.5, rhou = 0.7, rhog = 0.9)
)
res <- osr(nk,
  params      = c(kappa = 0.1, psi = 1.5, rhou = 0.7, rhog = 0.9),
  shock_sd    = c(e.u = 1.0, e.g = 0.5),
  osr_params  = c(psi = 1.5),
  welfare_weights = list(Q_xx = c(u = 0, g = 0),
                         Q_yy = c(p = 1, x = 0.5, r = 0.1)),
  lower = 1.01, upper = 5.0)
```

```
print(res)
```

---

pac_simulate *Simulate a PAC Equation Along a Target Path*

Description

Simulates $y_t = \sum_k a_k y_{t-k} + \sum_j cw_j E_t y^*_{t+j}$ given a deterministic path for the target under perfect foresight (the target path is known at $t = 1$). Values of the target beyond the end of the supplied path are held at its final value, so a path that ends at a constant is treated as settling there permanently.

Usage

```
pac_simulate(pac, ystar, y_init = NULL)
```

Arguments

pac	A "dsge_pac" object from pac_weights.
ystar	Numeric vector giving the target path.
y_init	Optional numeric vector of m initial lags of y, ordered $y_0, y_{-1}, \ldots$. Defaults to zeros.

Value

A numeric vector of the simulated y path, the same length as ystar.

See Also

pac_weights

Examples

```
pac <- pac_weights(beta = 0.99, d = 3)
# Permanent unit step in the target
y <- pac_simulate(pac, ystar = c(rep(0, 5), rep(1, 60)))
round(head(y, 12), 3)
# Adjustment is gradual but eventually complete
round(tail(y, 1), 6)
```

`pac_target_loading` *Closed-Form Loading of a PAC Forward Sum on a Linear State Process*

Description

Collapses the infinite forward sum $\sum_j w_j E_t y^*_{t+j}$ of a PAC equation into a single loading vector, for the common case where the target is a linear function of a state vector that evolves as $s_{t+1} = \Psi s_t$. Because $E_t y^*_{t+j} = e' \Psi^j s_t$, the sum has the exact closed form

$$\sum_j w_j E_t y^*_{t+j} = c\, e' \prod_k (I - \beta \phi_k \Psi)^{-1} s_t,$$

which avoids any truncation.

Usage

```
pac_target_loading(pac, Psi, e)
```

Arguments

`pac`	A `"dsge_pac"` object from `pac_weights`.
`Psi`	Square transition matrix Ψ of the target's state process. Must be stable (spectral radius below $1/(\beta\phi)$ for every root, which in practice means below one).
`e`	Numeric selector vector with `length(e) == nrow(Psi)` such that $y^*_t = e' s_t$.

Value

A numeric vector ℓ of the same length as `e`, such that the PAC forward sum equals $\ell' s_t$.

See Also

`pac_weights`

Examples

```
pac <- pac_weights(beta = 0.99, d = 1.5)
# AR(1) target with persistence 0.8
Psi <- matrix(0.8, 1, 1)
pac_target_loading(pac, Psi, e = 1)
```

`pac_weights` *Solve a Polynomial Adjustment Cost (PAC) Equation*

Description

Computes the reduced-form lag coefficients and forward-looking weights implied by a polynomial adjustment-cost problem, as used in the FRB/US model. Given a discount factor and the adjustment-cost parameters on the first m differences of the choice variable, the function factors the Euler equation's characteristic polynomial and returns the resulting PAC representation.

Usage

```
pac_weights(beta, d, horizon = 60L)
```

Arguments

`beta`	Numeric discount factor in $(0, 1]$.
`d`	Numeric vector of adjustment-cost parameters $(d_1, \ldots, d_m)$ on the first m differences. All entries must be non-negative and at least one must be positive. `length(d)` sets the order m.
`horizon`	Integer. Number of forward weights w_j to return. Default 60.

Details

Special cases worth knowing:

- $m = 1$ has the closed form ϕ = the stable root of $d\beta z^2 - (1 + d + d\beta)z + d = 0$, giving $y_t = \phi y_{t-1} + (1 - \phi)(1 - \beta\phi) \sum_j (\beta\phi)^j E_t y^*_{t+j}$.
- As $d \to 0$ adjustment is costless, $\phi \to 0$ and the solution collapses to $y_t = y^*_t$.
- As $d \to \infty$ adjustment becomes prohibitively costly and $\phi \to 1$.

Value

An object of class `"dsge_pac"` with elements:

`roots` The m stable roots ϕ_k.

`lag_coef` Length-m vector a_k of coefficients on y_{t-k}.

`fwd_weight` Length-`horizon+1` vector of the *scaled* forward weights $c\,w_j$ on $E_t y^*_{t+j}$, starting at $j = 0$.

`scale` The normalising constant c.

`homogeneity` $\sum_k a_k + \sum_j c w_j$, which equals 1 up to the truncation error in `horizon`.

`beta`, `d`, `m` Inputs.

References

Tinsley, P.A. (1993). Fitting both data and theories: Polynomial adjustment costs and error-correction decision rules. Federal Reserve Board FEDS working paper 93-21.

Brayton, F., Davis, M. and Tulip, P. (2000). Polynomial adjustment costs in FRB/US. Federal Reserve Board.

See Also

pac_target_loading to collapse the infinite forward sum when the target follows a linear state process, and pac_simulate to simulate a PAC equation along a given target path.

Examples

```
# First-order adjustment costs
p <- pac_weights(beta = 0.99, d = 1.5)
p$roots
p$lag_coef
head(p$fwd_weight)
p$homogeneity          # 1 up to truncation

# Second-order costs give richer lag dynamics
p2 <- pac_weights(beta = 0.99, d = c(1.0, 0.5))
p2$lag_coef
```

---

parameter_sensitivity *Parameter Sensitivity Analysis for DSGE Models*

Description

Evaluates the sensitivity of key model outputs to one-at-a-time parameter perturbations. For each free parameter, the model is re-solved at theta +/- delta, and changes in the log-likelihood, impulse responses, steady state, and policy matrix are recorded.

Usage

```
parameter_sensitivity(x, ...)

## S3 method for class 'dsge_fit'
parameter_sensitivity(
  x,
  what = c("loglik", "irf"),
  delta = 0.01,
  irf_horizon = 20L,
  ...
)
```

```
## S3 method for class 'dsge_bayes'
parameter_sensitivity(
  x,
  what = c("loglik", "irf"),
  delta = 0.01,
  irf_horizon = 20L,
  ...
)
```

Arguments

`x`	A `dsge_fit` or `dsge_bayes` object.
`...`	Additional arguments (currently unused).
`what`	Character vector specifying which outputs to assess. Any subset of `c("loglik", "irf", "steady_state", "policy")`. Default is `c("loglik", "irf")`.
`delta`	Numeric. Perturbation size as a fraction of the parameter value. Default is 0.01 (1 percent).
`irf_horizon`	Integer. Number of IRF periods. Default is 20.

Details

For each parameter θ_j, the model is solved at $\theta_j(1+\delta)$ and $\theta_j(1-\delta)$. The numerical derivative is approximated as a central difference. Elasticities are reported as $(\theta_j/f)\cdot(df/d\theta_j)$, representing the percentage change in the output for a 1 percent change in the parameter.

Value

An object of class `"dsge_sensitivity"` containing:

loglik Data frame of log-likelihood sensitivities (if requested).

irf Data frame of IRF sensitivities (if requested).

steady_state Data frame of steady-state sensitivities (if requested).

policy Data frame of policy matrix sensitivities (if requested).

param_names Character vector of parameter names.

delta Perturbation fraction used.

Examples

```
m <- dsge_model(
  obs(y ~ z),
  state(z ~ rho * z),
  start = list(rho = 0.5)
)
set.seed(1)
z <- numeric(100); for (i in 2:100) z[i] <- 0.8*z[i-1]+rnorm(1)
fit <- estimate(m, data = data.frame(y = z))
sa <- parameter_sensitivity(fit)
print(sa)
```

`particle_filter` *Bootstrap Particle Filter*

Description

Evaluates the log-likelihood of a DSGE model using the bootstrap (sequential importance resampling) particle filter. Unlike the Kalman filter, this method is valid for fully nonlinear models and does not require a linearized solution.

Usage

```
particle_filter(y, H, M, Z, n_particles = 1000L, meas_sd = 0.001, seed = NULL)
```

Arguments

`y`	Matrix of observed data (T x n_obs), demeaned if appropriate.
`H`	State transition matrix (n_s x n_s) from `solve_dsge()`.
`M`	Shock impact matrix (n_s x n_shocks).
`Z`	Observation matrix (n_obs x n_s), i.e. `D %*% G`.
`n_particles`	Integer. Number of particles. Default 1000.
`meas_sd`	Numeric scalar or vector (length n_obs). Standard deviation of measurement error added to each observation equation. A small positive value (e.g., 0.001) stabilises the filter when the model has exact observations. Default 0.001.
`seed`	Optional integer random seed.

Details

The bootstrap particle filter proceeds as follows each period t:

1. **Propagate**: draw proposed particles by simulating the transition equation from the filtered particles at t-1.
2. **Weight**: assign importance weights proportional to the observation density $p(y_t|x_t^{(i)})$.
3. **Normalise**: rescale weights to sum to one.
4. **Log-likelihood contribution**: $\log \bar{w}_t$ where $\bar{w}_t$ is the average unnormalised weight.
5. **Resample**: systematic resampling when the effective sample size drops below `n_particles / 2`.

Value

A list with:

`loglik` Scalar log-likelihood estimate.

`filtered_states` Matrix (T x n_s) of weighted particle means.

`ess` Vector (length T) of effective sample sizes.

`n_particles` Number of particles used.

See Also

`particle_filter_loglik`, `bayes_particle`

---

particle_filter_loglik

Particle Filter Log-Likelihood for a DSGE Solution

Description

Convenience wrapper around `particle_filter` that accepts a `dsge_solution` object and observed data directly.

Usage

```
particle_filter_loglik(
  sol,
  y,
  n_particles = 1000L,
  meas_sd = 0.001,
  seed = NULL
)
```

Arguments

`sol`	A `dsge_solution` object from `solve_dsge`.
`y`	Matrix or data frame of observed data (T x n_obs).
`n_particles`	Integer. Number of particles. Default 1000.
`meas_sd`	Numeric. Measurement error standard deviation. Default 0.001.
`seed`	Optional integer random seed.

Value

Scalar log-likelihood estimate.

See Also

`particle_filter`, `bayes_particle`

Examples

```
m <- dsge_model(
  obs(y ~ z), state(z ~ rho * z), start = list(rho = 0.8)
)
set.seed(1)
z <- numeric(100); for (i in 2:100) z[i] <- 0.8 * z[i-1] + rnorm(1)
dat <- data.frame(y = z - mean(z))
sol <- solve_dsge(m, params = c(rho = 0.8), shock_sd = c(z = 0.2))
ll  <- particle_filter_loglik(sol, dat, n_particles = 500, seed = 1)
```

---

perfect_foresight *Perfect Foresight / Deterministic Transition Paths*

Description

Simulate deterministic transition paths for DSGE models under perfect foresight. Supports temporary shocks, permanent shocks, and initial condition experiments using the linearized solution.

Usage

```
perfect_foresight(
  x,
  shocks = NULL,
  initial = NULL,
  horizon = 40L,
  params = NULL,
  shock_sd = NULL,
  in_sd = FALSE
)
```

Arguments

`x` A solved DSGE model object. Can be a `dsge_solution` (from `solve_dsge`), `dsge_fit` (from `estimate`), or `dsge_bayes` (from `bayes_dsge`).

`shocks` Deterministic shock specification. Can be:

- A named list where each element is a numeric vector giving the shock path for that shock variable (e.g., `list(u = c(1, 0, 0))`). Unnamed periods after the vector ends are treated as zero.
- A matrix of dimension `horizon` x `n_shocks` with column names matching shock names.

Shock values are in units of the shock variable (not standard deviations). If `NULL` (default), no shocks are applied (useful with `initial` to study convergence from displaced initial conditions).

Multi-period shock paths. A non-scalar vector specifies a sequence of shocks hitting at successive periods, e.g. `shocks = list(u = c(0, 0, 0, 1, 0, 0))` simulates a shock of size 1 arriving at $t = 4$. Note that the linearized perfect-foresight path uses the recursive policy function $x_{t+1} = Hx_t + M\varepsilon_{t+1}$, so each shock is treated as a period-by-period surprise (agents do not react in advance). For true *anticipated / news* shocks where agents adjust at $t = 1$ in expectation of a future shock, use `perfect_foresight_nonlinear()`: the stacked-time Newton solver respects forward-looking expectations and correctly captures anticipation.

`initial` Named numeric vector of initial state deviations from steady state. Names must match state variable names. Unspecified states default to zero. Default is `NULL` (all states start at steady state).

`horizon` Integer. Number of periods to simulate. Default 40.

`params` Named numeric vector of parameters. Required only when `x` is a model object that has not been solved. For solved objects, parameters are extracted automatically.

`shock_sd` Named numeric vector of shock standard deviations. Used only when `x` is a model object. Default `NULL`.

`in_sd` Logical. If `TRUE`, shock values in `shocks` are interpreted as multiples of the shock standard deviation. Default `FALSE` (shocks are in level units).

Details

The deterministic transition path is computed using the linearized state-space representation:

$$x_{t+1} = Hx_t + M\varepsilon_{t+1}$$

$$y_t = Gx_t$$

where x_t are state deviations from steady state, y_t are control deviations, and ε_t are deterministic shocks.

This uses the first-order linearized solution, so results are approximate for large shocks. For small to moderate shocks, the linearized paths are accurate.

Value

An object of class `"dsge_perfect_foresight"` containing:

states Matrix (horizon x n_states) of state deviations from SS

controls Matrix (horizon x n_controls) of control deviations

state_levels Matrix of state levels (SS + deviation), if SS available

control_levels Matrix of control levels, if SS available

steady_state Named numeric vector of steady-state values

shock_path Matrix (horizon x n_shocks) of applied shocks

initial Named vector of initial state deviations

horizon Integer horizon

state_names Character vector of state names

control_names Character vector of control names

shock_names Character vector of shock names

H State transition matrix used

G Policy matrix used

M Shock impact matrix used

Examples

```
# Simple AR(1) model
mod <- dsge_model(
  obs(p ~ x),
  state(x ~ rho * x),
  start = list(rho = 0.9)
)
sol <- solve_dsge(mod, params = list(rho = 0.9), shock_sd = c(x = 0.01))

# One-time shock at period 1
pf <- perfect_foresight(sol, shocks = list(x = 0.01), horizon = 40)
plot(pf)

# Displaced initial condition
pf2 <- perfect_foresight(sol, initial = c(x = 0.05), horizon = 40)
plot(pf2)

# Anticipated (news) shock: known at t=1, hits at t=5
pf3 <- perfect_foresight(sol,
  shocks  = list(x = c(0, 0, 0, 0, 0.01)),
  horizon = 40)
plot(pf3)
```

---

perfect_foresight_expect_err

Perfect Foresight Simulation with Expectation Errors

Description

Generalises `perfect_foresight` by allowing the realised shock path to differ from the path agents anticipate at each point in time. In standard perfect foresight, agents at $t = 1$ see every future shock; in the with-expectation-errors variant, agents form subjective expectations at each period k, solve the residual perfect-foresight problem, then nature delivers the actual one-period shock (which may be a surprise).

Usage

```
perfect_foresight_expect_err(
  x,
  actual_shocks,
  expected_shocks = NULL,
  initial = NULL,
  horizon = 40L
)
```

Arguments

x	A dsge_solution object.
actual_shocks	Named list (or matrix) describing the *realised* shock path – same format as the shocks argument of perfect_foresight.
expected_shocks	Optional named list (or matrix) describing what agents *expect* at $t = 1$. Default NULL meaning agents expect no further shocks beyond those already realised. When agents expect the same path that actually materialises, this function reproduces perfect_foresight().
initial	Optional named numeric vector of initial state deviations.
horizon	Integer. Number of periods. Default 40.

Details

This is the analogue of Dynare's perfect_foresight_with_expectation_errors_solver command. A typical use case: study how the economy reacts to a sequence of news/MIT shocks that arrive unexpectedly even though each shock, once it lands, is treated as fully credible going forward.

Value

An object of class c("dsge_perfect_foresight_expecterr", "dsge_perfect_foresight") containing the same fields as perfect_foresight plus an extra element expectation_paths – a list of per-period subjective forecast paths (one matrix per starting period) so users can inspect how agent expectations evolved.

See Also

perfect_foresight (no expectation errors), perfect_foresight_nonlinear.

Examples

```
m <- dsge_model(
  obs(y ~ beta * lead(y) + 0.1 * x),
  state(x ~ 0.9 * x),
  fixed = list(beta = 0.99))
sol <- solve_dsge(m, params = c(), shock_sd = c(x = 1))
# Agents expect no shocks; nature delivers a one-time shock at t = 5
pf <- perfect_foresight_expect_err(sol,
  actual_shocks = list(x = c(0, 0, 0, 0, 1)),
```

```
  horizon = 30)
plot(pf)
```

`perfect_foresight_nonlinear`

Perfect Foresight for Nonlinear DSGE Models

Description

Computes deterministic perfect foresight transition paths for nonlinear DSGE models using a stacked-time Newton solver (Juillard et al., 1998). Unlike the linearized `perfect_foresight()`, this function solves the full nonlinear equilibrium conditions simultaneously over the entire horizon, giving exact (up to Newton tolerance) paths even for large shocks.

Usage

```
perfect_foresight_nonlinear(
  model,
  params,
  shock_sd,
  shocks = NULL,
  initial = NULL,
  horizon = 40L,
  tol = 1e-08,
  max_iter = 50L,
  verbose = FALSE
)
```

Arguments

`model`	A `dsgenl_model` object.
`params`	Named numeric vector or list of free parameter values.
`shock_sd`	Named numeric vector of shock standard deviations with names matching `model$variables$exo_state` Used only for the linearized warm-start step; the nonlinear solver uses `shocks` directly.
`shocks`	Shock path specification (same interface as `perfect_foresight`): a named list where each element is a numeric vector giving the shock values for that exogenous state over time, or a matrix of dimension `horizon` x `n_exo` with column names matching exogenous state names. Unspecified periods are zero. Supports **anticipated (news) shocks**: because the nonlinear system is solved simultaneously over the whole horizon under perfect foresight, agents see the entire future shock path at $t = 1$ and react in advance. Place the shock value at the desired arrival period in the vector (e.g. `list(Z = c(0, 0, 0, 0.1))` for a TFP shock announced at $t = 1$ but arriving at $t = 4$).

`initial`	Named numeric vector of initial DEVIATIONS from steady state for state variables. Defaults to zero (all states start at SS).
`horizon`	Integer. Number of periods to simulate. Default 40.
`tol`	Convergence tolerance (maximum absolute equation residual). Default `1e-8`.
`max_iter`	Maximum Newton iterations. Default 50.
`verbose`	Logical. If `TRUE`, prints iteration progress. Default `FALSE`.

Details

The algorithm (Juillard et al., 1998) stacks the $n \times T$ nonlinear equilibrium conditions into a single system and solves it by Newton's method. The Jacobian is block-bidiagonal (each period's equations depend only on the current and next-period variables) and is assembled numerically using forward finite differences. The block structure is exploited via block back-substitution, giving $O(Tn^3)$ cost per Newton step. Armijo backtracking stabilises convergence for large initial steps.

The initial state x_1 is pinned to `steady_state + initial` via equality constraints that replace the state equations at $t = 1$. For $t = T + 1$ the terminal condition $v_{T+1} = \bar{v}$ (steady state) is imposed. The solver is warm-started from the linearized `perfect_foresight()` path, which is exact for small shocks and provides a good initial guess for large ones.

Value

A `"dsge_perfect_foresight"` object with the same fields as `perfect_foresight`, so all `plot`, `print`, and `summary` methods apply unchanged. Three extra fields are appended:

nonlinear `TRUE`

newton_iters Number of Newton iterations taken

converged Logical; `TRUE` if `max|R| < tol`

References

Juillard, M., Laxton, D., McAdam, P. and Pioro, H. (1998). An algorithm competition: First-order iterations versus Newton-based techniques. *Journal of Economic Dynamics and Control*, 22, 1291–1318.

See Also

`perfect_foresight` for linearized paths, `dsgenl_model` for nonlinear model specification.

Examples

```
rbc <- dsgenl_model(
  "1/C = beta / C(+1) * (alpha * exp(Z) * K^(alpha-1) + 1 - delta)",
  "K(+1) = exp(Z) * K^alpha - C + (1 - delta) * K",
  "Z(+1) = rho * Z",
  observed = "C", endo_state = "K", exo_state = "Z",
  fixed = list(alpha = 0.33, beta = 0.99, delta = 0.025),
  start = list(rho = 0.9)
)
```

```
pf <- perfect_foresight_nonlinear(
  rbc, params = c(rho = 0.9), shock_sd = c(Z = 0.01),
  shocks = list(Z = 0.1), horizon = 40
)
plot(pf)
```

`plot.dsge_bayes` *Plot Bayesian DSGE Results*

Description

Produces diagnostic plots for posterior draws from a Bayesian DSGE fit.

Usage

```
## S3 method for class 'dsge_bayes'
plot(
  x,
  type = c("trace", "density", "prior_posterior", "running_mean", "acf", "pairs", "all",
    "irf"),
  pars = NULL,
  ...
)
```

Arguments

`x` A `dsge_bayes` object from `bayes_dsge()`.

`type` Character. Plot type:

- `"trace"` Trace plots showing MCMC chains for each parameter. All chains are overlaid with distinct colors. Useful for assessing convergence and mixing.
- `"density"` Posterior density plots for each parameter with the prior distribution overlaid as a red dashed line. Useful for seeing how much the data updated the prior.
- `"prior_posterior"` Dedicated prior-vs-posterior comparison. Same layout as `"density"` but with the title "Prior vs Posterior" to emphasize the comparison.
- `"running_mean"` Cumulative posterior mean by iteration for each parameter. All chains are shown. Useful for assessing whether the chain has settled.
- `"acf"` Autocorrelation function plots for each parameter, pooled across chains. Useful for diagnosing slow mixing.
- `"pairs"` Pairwise scatter plots of posterior draws (lower triangle) with correlation coefficients (upper triangle) and marginal histograms (diagonal). Useful for detecting parameter correlations. Draws are thinned for readability.
- `"all"` A combined diagnostic panel showing trace plot (left) and posterior density with prior (right) side by side for each parameter.

"irf" Posterior impulse-response functions with credible bands. Calls irf() internally and plots the result. Additional arguments periods, impulse, response, n_draws, and level are passed through.

pars — Character vector of parameter names to include. If NULL (default), all parameters are shown. Applies to trace, density, prior_posterior, running_mean, acf, pairs, and all. Ignored for irf (use impulse/response instead).

... — Additional arguments. For type = "irf", arguments periods, impulse, response, n_draws, and level are passed to irf.dsge_bayes().

Details

All plot types except pairs and irf handle any number of parameters by paginating across multiple plot pages (up to 4 parameters per page). In interactive sessions, devAskNewPage() is used to prompt between pages.

For the "pairs" plot, at most 1000 draws are used to keep the plot readable. The correlation matrix is printed to the console.

Forecast plotting is not currently supported for Bayesian fits. Use irf() for posterior impulse-response analysis.

Value

Invisibly returns the dsge_bayes object x. Called for the side effect of producing diagnostic plots on the active graphics device.

Examples

```
m <- dsge_model(
  obs(y ~ z),
  state(z ~ rho * z),
  start = list(rho = 0.5)
)
set.seed(42)
z <- numeric(200); for (i in 2:200) z[i] <- 0.8 * z[i-1] + rnorm(1)
fit <- bayes_dsge(m, data = data.frame(y = z),
                  priors = list(rho = prior("beta", shape1 = 2, shape2 = 2)),
                  chains = 2, iter = 2000, seed = 1)
plot(fit, type = "trace")
plot(fit, type = "density")
plot(fit, type = "prior_posterior")
plot(fit, type = "running_mean")
plot(fit, type = "acf")
plot(fit, type = "all")
# Parameter selection
plot(fit, type = "trace", pars = "rho")
```

plot.dsge_decomposition *Plot Historical Shock Decomposition*

Description

Creates a stacked bar chart showing the contribution of each structural shock to the observed variables over time.

Usage

```
## S3 method for class 'dsge_decomposition'
plot(x, which = NULL, ...)
```

Arguments

x	A dsge_decomposition object.
which	Which observable(s) to plot. Integer or character. Default is all.
...	Additional arguments (currently unused).

Value

No return value, called for the side effect of producing stacked bar charts of the historical shock decomposition on the active graphics device.

plot.dsge_forecast *Plot DSGE Forecasts*

Description

Plots forecast paths for observed variables.

Usage

```
## S3 method for class 'dsge_forecast'
plot(x, ...)
```

Arguments

x	A dsge_forecast object from forecast.dsge_fit().
...	Additional arguments passed to base plotting functions.

Value

No return value, called for the side effect of producing forecast path plots on the active graphics device.

plot.dsge_irf — *Plot Impulse-Response Functions*

Description

Creates a multi-panel plot of impulse-response functions with optional confidence bands.

Usage

```
## S3 method for class 'dsge_irf'
plot(x, impulse = NULL, response = NULL, ci = TRUE, ...)
```

Arguments

x	A dsge_irf object from irf().
impulse	Character vector of impulse variables to plot. If NULL, plots all.
response	Character vector of response variables to plot. If NULL, plots all.
ci	Logical. If TRUE (default), plot confidence bands if available.
...	Additional arguments passed to base plotting functions.

Value

No return value, called for the side effect of producing a multi-panel impulse-response plot on the active graphics device.

plot.dsge_occbin — *Plot OccBin Simulation Results*

Description

Plot OccBin Simulation Results

Usage

```
## S3 method for class 'dsge_occbin'
plot(x, vars = NULL, compare = TRUE, shade = TRUE, max_panels = 9L, ...)
```

Arguments

x	A dsge_occbin object.
vars	Character vector of variable names to plot. Default: constrained variables plus a few others.
compare	Logical. If TRUE (default), overlay the unconstrained path for comparison.
shade	Logical. If TRUE (default), shade periods where constraints bind.
max_panels	Integer. Maximum panels per page. Default 9.
...	Additional arguments (unused).

Value

No return value, called for the side effect of producing OccBin simulation plots on the active graphics device. Constrained and unconstrained paths are shown, with shaded regions where constraints bind.

---

plot.dsge_perfect_foresight

Plot Perfect Foresight Transition Paths

Description

Plot the deterministic transition paths from a `perfect_foresight` result, optionally overlaying a second path for comparison (e.g. linearized vs. nonlinear).

Usage

```
## S3 method for class 'dsge_perfect_foresight'
plot(x, vars = NULL, type = "deviation", compare = NULL, max_panels = 9L, ...)
```

Arguments

`x`	A `dsge_perfect_foresight` object.
`vars`	Character vector of variable names to plot. If `NULL` (default), plots all variables with non-trivial paths.
`type`	Character. One of `"deviation"` (default) or `"level"`. Controls whether to plot deviations from SS or levels.
`compare`	Optional second `dsge_perfect_foresight` object to overlay on the same panels (e.g. pass the linearized `perfect_foresight()` result alongside a `perfect_foresight_nonlinear()` result to visualise the nonlinearity premium). Only variables common to both objects are overlaid.
`max_panels`	Integer. Maximum number of panels per plot page. Default 9.
`...`	Additional arguments (currently unused).

Value

No return value, called for the side effect of producing transition path plots on the active graphics device.

plot.dsge_smoothed *Plot Smoothed States*

Description

Plot Smoothed States

Usage

```
## S3 method for class 'dsge_smoothed'
plot(x, which = NULL, type = c("states", "fit"), ...)
```

Arguments

x	A dsge_smoothed object.
which	Which states to plot. Integer vector, character vector of state names, or NULL (all states).
type	Either "states" or "fit". "states" plots the smoothed state variables; "fit" plots the smoothed observables against data.
...	Additional arguments passed to plot().

Value

No return value, called for the side effect of producing smoothed state or fit plots on the active graphics device.

plot.dsge_variance_decomposition *Plot a Variance Decomposition*

Description

For unconditional decompositions, draws a horizontal stacked bar chart showing shock shares (in percent) for each observable. For FEVD, draws one stacked-bar panel per observable with horizons on the x-axis.

Usage

```
## S3 method for class 'dsge_variance_decomposition'
plot(x, ...)
```

Arguments

x	A dsge_variance_decomposition object.
...	Additional arguments (currently unused).

Value

Invisibly returns `x`; called for the side effect of producing a plot.

`policy_matrix` *Extract Policy Matrix*

Description

Returns the policy matrix G from a fitted or solved DSGE model. The policy matrix maps state variables to control variables: $y_t = Gx_t$.

Usage

```
policy_matrix(x, se = TRUE, level = 0.95)
```

Arguments

`x`	A `dsge_fit` or `dsge_solution` object.
`se`	Logical. If `TRUE` (default) and `x` is a `dsge_fit`, compute delta-method standard errors.
`level`	Confidence level for intervals. Default is 0.95.

Value

If `se = FALSE`, returns the G matrix. If `se = TRUE`, returns a list with `matrix`, `se`, `lower`, `upper`, and a data frame `table`.

`posterior_predictive` *Posterior predictive check*

Description

Simulates data from the posterior predictive distribution and compares summary statistics (variance, autocorrelation) with the observed data. This helps assess whether the estimated model can reproduce key features of the data.

Usage

```
posterior_predictive(object, ...)
```

Arguments

`object`	A `"dsge_bayes"` object.
`...`	Additional arguments passed to methods (e.g., `n_draws`, `statistics`, `seed`).

Value

An object of class `"dsge_ppc"` with posterior predictive distributions and p-values for each statistic.

predetermined — *Declare a Predetermined (Backward-Looking) State Variable*

Description

Thin alias for `state(formula, shock = FALSE)` that names the role of the variable explicitly, matching Dynare's `predetermined_variables` command. Use whenever you want a state-like variable whose current value is fully determined by past values (no contemporaneous shock channel).

Usage

```
predetermined(formula)
```

Arguments

`formula`	A formula of the form `variable ~ expression`. The right-hand side gives the law of motion – typically references to other states or controls.

Details

Mapping to Dynare: Dynare declares `var k; predetermined_variables k;` so that references to `k` inside the model block mean the *predetermined* (period-t) value, with `k(+1)` referring to the forward value. The equivalent in our framework is `predetermined(k ~ <law of motion>)`, which contributes a row to the state transition with no shock column attached. This is identical to `state(k ~ <law of motion>, shock = FALSE)` – the two declarations are interchangeable.

Common uses:

- Lagged controls: `predetermined(c_lag ~ c)` makes `c_lag` carry the previous period's consumption into the next period.
- Capital accumulation: `predetermined(k ~ (1 - delta) * k + inv)`.
- Stocks of any kind that evolve via accounting identities.

Value

A `dsge_equation` of state type with `shock = FALSE`.

See Also

`state`, `obs`, `unobs`, `dsge_model`.

Examples

```
# AR(1) with a lagged-output term in the observation equation
m <- dsge_model(
  obs(y ~ rho * y_lag + e),
  predetermined(y_lag ~ y),
  state(e ~ phi * e),
  fixed = list(rho = 0.5, phi = 0.5))
```

```
sol <- solve_dsge(m, params = c(rho = 0.5, phi = 0.5),
                  shock_sd = c(e = 1))
sol$stable
```

`predict.dsge_fit` *Predict Method for DSGE Models*

Description

Computes one-step-ahead predictions or filtered state estimates from a fitted DSGE model.

Usage

```
## S3 method for class 'dsge_fit'
predict(
  object,
  type = c("observed", "state"),
  method = c("onestep", "filter"),
  newdata = NULL,
  ...
)
```

Arguments

`object`	A `dsge_fit` object.
`type`	Character. `"observed"` (default) for predicted values of observed control variables, or `"state"` for filtered latent state estimates.
`method`	Character. `"onestep"` (default) for one-step-ahead predictions using only past data, or `"filter"` for filtered estimates using past and contemporaneous data.
`newdata`	Optional new data for prediction. If `NULL`, uses the estimation data.
`...`	Additional arguments (currently unused).

Value

A matrix of predictions or state estimates.

`prediction_accuracy` *Prediction accuracy measures for a fitted DSGE model*

Description

Computes root mean squared error (RMSE), mean absolute error (MAE), and mean error (bias) of one-step-ahead predictions.

Usage

```
prediction_accuracy(object, ...)
```

Arguments

`object`	A `"dsge_fit"` object.
`...`	Additional arguments (currently unused).

Value

An object of class `"dsge_prediction_accuracy"` containing:

rmse Named numeric vector of RMSE by variable.

mae Named numeric vector of MAE by variable.

bias Named numeric vector of mean error by variable.

n Number of observations.

variables Variable names.

Examples

```
m <- dsge_model(
  obs(y ~ z),
  state(z ~ rho * z),
  start = list(rho = 0.5)
)
set.seed(1)
z <- numeric(100); for (i in 2:100) z[i] <- 0.8 * z[i-1] + rnorm(1)
fit <- estimate(m, data = data.frame(y = z))
acc <- prediction_accuracy(fit)
print(acc)
```

prediction_interval *Prediction intervals for DSGE models*

Description

Computes point predictions and prediction intervals using the one-step-ahead innovation variance from the Kalman filter.

Usage

```
prediction_interval(object, level = 0.95, ...)
```

Arguments

`object`	A `"dsge_fit"` object.
`level`	Confidence level for prediction intervals (default 0.95).
`...`	Additional arguments passed to `predict()`.

Value

An object of class `"dsge_prediction_interval"` with components `fit`, `lower`, `upper`, `se`, `level`, and `variables`.

Examples

```
m <- dsge_model(
  obs(y ~ z),
  state(z ~ rho * z),
  start = list(rho = 0.5)
)
set.seed(1)
z <- numeric(100); for (i in 2:100) z[i] <- 0.8 * z[i-1] + rnorm(1)
fit <- estimate(m, data = data.frame(y = z))
pi <- prediction_interval(fit, level = 0.95)
print(pi)
```

prior *Specify a Prior Distribution*

Description

Creates a prior distribution object for use in Bayesian DSGE estimation.

Usage

```
prior(distribution, ...)
```

Arguments

`distribution`	Character string specifying the distribution family. One of `"normal"`, `"beta"`, `"gamma"`, `"uniform"`, `"inv_gamma"`, `"inv_gamma1"`.
`...`	Distribution parameters (see Details).

Details

Distribution parameterizations:

normal `mean`, `sd`

beta `shape1`, `shape2` (alpha, beta parameters)

gamma `shape`, `rate`

uniform `min`, `max`

inv_gamma `shape`, `scale` — density: $p(x) \propto x^{-(shape+1)} \exp(-scale/x)$

inv_gamma1 `s`, `nu` — inverse gamma of type 1, a prior on a standard deviation whose square is inverse gamma (Dynare's `inv_gamma_pdf`): $p(x) \propto x^{-(nu+1)} \exp(-s/(2x^2))$. Alternatively give `mean` and `sd` (use `sd = Inf` for an infinite variance) and `s`, `nu` are derived as in Dynare.

Value

An object of class `"dsge_prior"`.

Examples

```
prior("normal", mean = 0, sd = 1)
prior("beta", shape1 = 2, shape2 = 2)
prior("inv_gamma", shape = 0.01, scale = 0.01)
prior("inv_gamma1", mean = 0.1, sd = 2)
```

`prior_posterior_update` *Prior-Posterior Update Diagnostics*

Description

Computes diagnostics measuring how informative the data was for each parameter, by comparing the posterior distribution to the prior.

Usage

```
prior_posterior_update(x, ...)

## S3 method for class 'dsge_bayes'
prior_posterior_update(x, ...)
```

Arguments

x	A dsge_bayes object.
...	Additional arguments (currently unused).

Details

For each estimated parameter, the following diagnostics are computed:

sd_ratio Ratio of posterior SD to prior SD. Values near 1 indicate the data was uninformative (posterior tracks the prior). Values much less than 1 indicate strong data information.

mean_shift Absolute difference between posterior mean and prior mean, measured in units of prior SD. Large shifts indicate the data substantially updated beliefs.

update Classification: "strong" if sd_ratio < 0.5 or mean_shift > 2; "moderate" if sd_ratio < 0.8 or mean_shift > 1; "weak" otherwise (posterior closely resembles the prior).

The SD ratio is the primary indicator of data informativeness. A parameter with sd_ratio close to 1 and small mean_shift is effectively determined by the prior, not the data.

Value

An object of class "dsge_prior_posterior" containing:

summary Data frame with per-parameter diagnostics including prior mean/sd, posterior mean/sd, SD ratio, mean shift, and update classification.

param_names Character vector of parameter names.

Examples

```
m <- dsge_model(
  obs(y ~ z),
  state(z ~ rho * z),
  start = list(rho = 0.5)
)
set.seed(1)
z <- numeric(100); for (i in 2:100) z[i] <- 0.8*z[i-1]+rnorm(1)
fit <- bayes_dsge(m, data = data.frame(y = z),
  priors = list(rho = prior("beta", shape1 = 2, shape2 = 2)),
  chains = 2, iter = 2000, seed = 1)
pp <- prior_posterior_update(fit)
print(pp)
```

`ramsey_policy` *Ramsey Optimal Policy for a DSGE Model*

Description

Solves for the welfare-maximising policy using the linear-quadratic approach. The planner chooses a feedback rule for the policy instrument(s) to minimise the expected discounted welfare loss function

$$L = E_0 \sum_{t=0}^{\infty} \beta^t (x_t' Q_{xx} x_t + y_t' Q_{yy} y_t + 2 x_t' Q_{xy} y_t)$$

subject to the model's equilibrium conditions.

Usage

```
ramsey_policy(
  model,
  params = NULL,
  shock_sd = NULL,
  instruments,
  welfare_weights,
  beta = 0.99,
  tol = 1e-10,
  max_iter = 10000L
)
```

Arguments

`model`	A `dsge_model` or `dsgenl_model` object.
`params`	Named numeric vector of model parameters.
`shock_sd`	Named numeric vector of shock standard deviations.
`instruments`	Character vector. Names of the policy instrument variables (subset of the model's control variables).
`welfare_weights`	A named list specifying the welfare loss weights. Elements: `Q_xx` Square matrix (n_s x n_s) on state variables. If a named vector is provided a diagonal matrix is constructed. `Q_yy` Square matrix (n_c x n_c) on control variables. If a named vector is provided a diagonal matrix is constructed. `Q_xy` Cross-weight matrix (n_s x n_c). Default is zero. At least one of `Q_xx` or `Q_yy` must be supplied.
`beta`	Numeric. Discount factor (0 < beta < 1). Default 0.99.
`tol`	Numeric. Convergence tolerance for the Riccati iteration. Default 1e-10.
`max_iter`	Integer. Maximum Riccati iterations. Default 10000.

Details

The algorithm:

1. Solve the model to first order to obtain G (controls as a function of states) and H (state transition).
2. Identify the columns of G and rows of the structural matrices corresponding to the policy instruments `u_t`.
3. Formulate the constrained LQR problem:

$$x_{t+1} = Ax_t + Bu_t + \eta\varepsilon_t$$

$$\min \sum \beta^t (x_t' R_x x_t + u_t' R_u u_t + 2x_t' N u_t)$$

4. Solve the DARE

$$P = R_x + \beta A'PA - \beta^2 (A'PB + N)(R_u + \beta B'PB)^{-1}(B'PA + N')$$

   via value-function iteration.
5. The optimal feedback rule is

$$F = -(R_u + \beta B'PB)^{-1}(\beta B'PA + N')$$

Value

An object of class `"dsge_ramsey"` containing:

`F` Optimal feedback matrix (n_instruments x n_s): the Ramsey policy rule $u_t = Fx_t$.

`H_ram` State transition matrix under the optimal policy.

`G_ram` Policy matrix under the optimal policy.

`welfare_loss` Steady-state welfare loss (unconditional mean).

`P` Solution to the discrete-time Riccati equation.

`converged` Logical: did the Riccati iteration converge?

`n_iter` Number of iterations taken.

`instruments` Names of policy instruments.

`first_order_sol` The first-order solution used as the basis.

References

Dennis, R. (2007). Optimal policy in rational-expectations models: New solution algorithms. *Macroeconomic Dynamics*, 11(1), 31-55.

Judd, K. L. (1992). Projection methods for solving aggregate growth models. *Journal of Economic Theory*, 58(2), 410-452.

See Also

`welfare_loss`, `solve_dsge`

Examples

```
# Simple NK model: planner minimises inflation and output gap variance
m <- dsge_model(
  obs(y ~ z),
  state(z ~ rho * z),
  start = list(rho = 0.8)
)
set.seed(1)
sol <- solve_dsge(m, params = c(rho = 0.8), shock_sd = c(z = 0.1))
n_s <- ncol(sol$H); n_c <- nrow(sol$G)
ram <- ramsey_policy(
  m,
  params    = c(rho = 0.8),
  shock_sd  = c(z = 0.1),
  instruments = "y",
  welfare_weights = list(
    Q_yy = matrix(1, 1, 1, dimnames = list("y", "y"))
  )
)
print(ram)
```

`read_dynare` *Import a Dynare .mod File*

Description

Reads a Dynare model file and translates it into a nonlinear dsge model, together with the calibration, shock standard deviations, measurement errors, priors, optimal-policy problem and occasionally binding constraints it declares, so that the model can be solved, simulated or estimated in R without Dynare, MATLAB or Octave.

Usage

```
read_dynare(file, text = NULL, observed = NULL, defines = NULL)
```

Arguments

`file`	Path to a `.mod` file.
`text`	Alternatively, the model code as a character vector (one element per line, or a single string). Used instead of `file` when supplied.
`observed`	Optional character vector of observed variables. Overrides the file's `varobs` declaration.
`defines`	Optional named list of macro variables, the equivalent of Dynare's `-D` command-line option (e.g. `list(N = 3)`).

Details

Declarations and model. `var`, `varexo`, `varexo_det`, `parameters`, `predetermined_variables`, `varobs`, parameter assignments (and constants assigned to undeclared names), the `model` block (including `model(linear)`, equation tags and # model-local variables), leads and lags of any length on variables and shocks, and `STEADY_STATE(x)`, which always equals the steady state of `x` at the current parameter values.

MATLAB code. Dynare passes MATLAB statements in a `.mod` file through to MATLAB; `read_dynare()` runs them with a built-in MATLAB interpreter (matrices, cell arrays, structures, indexing, loops, functions, `eval`, `fsolve`, `fzero`, `csolve`, ...), so calibrations computed in MATLAB (e.g. `sigma = sqrt(V(1, 1))`, `verbatim` blocks or `set_param_value()`) are reproduced. Parameters take the values they have when the file first solves, simulates or estimates the model (the first `stoch_simul`, `estimation`, ...); later assignments and MATLAB code that post-processes results (plots, tables) are skipped and listed in `notes`.

Macro processor. `@#define`, `@#if`/`@#elseif`/`@#else`/`@#endif`, `@#ifdef`, `@#ifndef`, `@#for` (over arrays, ranges and tuples, with optional `when` filters), `@#include`, `@#includepath`, `@#echo`, `@#error`, simple macro functions and `@{...}` substitution are expanded before the model is translated.

Timing. No manual re-timing is needed. Every Dynare variable becomes a control; lags become auxiliary state variables named `x_lag1`, `x_lag2`, ...; leads beyond one period become auxiliary controls `x_lead1`, ...; and each shock becomes an exogenous state that holds the current innovation. Impulse responses therefore have the same timing as in Dynare. The auxiliary variables also appear in solution and IRF output.

Steady state and shocks. `steady_state_model` becomes the model's steady-state function (variables it does not set keep their `initval` value, 0 by default, as in Dynare) and `initval` supplies starting values for the numerical solver. A MATLAB steady-state file `<model>_steadystate.m` next to the `.mod` file is run with the MATLAB interpreter instead, including helper functions in other `.m` files of the same folder. Parameters that `steady_state_model` or the steady-state file set (calibrated to targets) are recomputed from the other parameters whenever the model is solved, as in Dynare. Models declared `model(linear)` are linearised with an exact Jacobian. In the `shocks` block, standard deviations, variances, covariances and correlations are supported; correlated shocks are orthogonalised by Cholesky factorisation in `varexo` order, which reproduces Dynare's impulse responses. Deterministic paths (`periods`/`values`) are returned in `shock_paths`.

Observed variables and measurement errors. A model may have fewer observed variables than shocks. A `stderr` on an observed endogenous variable (in `shocks` or `estimated_params`) is a measurement error: the variable `y` is observed as `y_obs = y + y_me`, where `y_me` is an i.i.d. shock. `estimate()` and `bayes_dsge()` rename a data column `y` to `y_obs` automatically. With more observed variables than shocks and measurement errors, the likelihood would be singular, so the extra variables are dropped with a note.

Priors. Dynare's mean/standard-deviation prior parameterisation is converted to dsge's, exactly: `normal_pdf`, `beta_pdf`, `gamma_pdf`, `uniform_pdf`, `inv_gamma2_pdf`, and `inv_gamma_pdf` / `inv_gamma1_pdf` (a prior on a standard deviation, translated to the `"inv_gamma1"` distribution of `prior()` with Dynare's own parameterisation). Shape names are case-insensitive. Shifted or generalised priors and `weibull_pdf` are not translated and are listed in `notes`.

Estimation options. `presample`, `first_obs` and `nobs` from the file's `estimation` command are stored in `estimation` and used by `estimate()` and `bayes_dsge()`: the data are restricted to the estimation sample and the first `presample` observations only initialise the Kalman filter. The filter starts from the stationary distribution (Dynare's `lik_init = 1`, the default) or, with `lik_init`

= 2, from Dynare's covariance of 10 times the identity on its state vector (the observed and predetermined variables); other lik_init values are reported in notes. Set x$model$kalman_init to NULL to use the stationary initialisation instead.

Optimal policy. With planner_objective and ramsey_model or ramsey_policy, the planner's first-order conditions are derived symbolically and added to the model together with Lagrange multipliers MULT_1, ..., as in Dynare, so solve_dsge() returns the Ramsey equilibrium; the steady state is found with the multipliers concentrated out. With discretionary_policy (linear models), the time-consistent rule is computed with the Dennis (2007) algorithm at the calibrated parameters and the model is closed with the planner's time-consistent targeting rule. With osr_params, osr_params_bounds and optim_weights, osr() can be called directly on the imported model.

OccBin. Equations tagged bind = 'c' / relax = 'c' and an occbin_constraints block define occasionally binding constraints; simulate_occbin() on the imported model solves them with the piecewise-linear algorithm of Guerrieri and Iacoviello (2015), as Dynare's occbin_solver does, using the file's shocks(surprise) block by default.

Perfect foresight. initval, endval, histval, steady, the deterministic shocks (periods/values), mcp equation tags and perfect_foresight_setup(periods =) / simul are stored in perfect_foresight; simulate_perfect_foresight() solves the deterministic path as Dynare's perfect_foresight_solver does.

MATLAB data and toolbox functions. The MATLAB interpreter reads data with load (Octave text files; MATLAB .mat files up to version 7 with the **R.matlab** package), xlsread / readmatrix (spreadsheets, with **readxl**), csvread and dlmread, and provides fmincon, fminunc, lsqnonlin, hpfilter, ksdensity, interp1, polyfit and other common functions. When a file cannot be run (e.g. a data file is missing), the error names the MATLAB statement that failed.

Not supported: external_function, trend_var, EXPECTATION(), diff(), adl(), PAC and VAR expectation operators; these raise an error. Other blocks and commands are recorded but not run, as is MATLAB code that calls Dynare's internal functions (e.g. perfect_foresight_solver_core).

Value

An object of class "dsge_dynare", a list with components:

model The translated dsgenl_model. Parameters listed in estimated_params are free (with starting values); all other parameters are fixed at their calibrated values.

params Named numeric vector of calibrated parameter values.

shock_sd Named numeric vector of standard deviations of the model's shocks, including measurement errors (0 for shocks with no declared variance, as in Dynare).

priors Named list of prior() objects translated from estimated_params, ready for bayes_dsge(), or NULL.

estimated_params Data frame describing each estimated_params entry and how it was translated.

observed Observed variables (Dynare names).

measurement_errors, data_map Observed variables with a measurement error, and the model variable (y_obs) each is mapped to.

variables, shocks, shocks_det, parameters Names declared in the file.

shock_paths Period-by-shock matrix of deterministic shock values from shocks blocks with periods/values, or NULL.

aux Data frame of auxiliary variables created for leads, lags and measurement errors.

policy Optimal-policy problem (`ramsey`, `discretion` or `osr`), or `NULL`.

occbin Occasionally binding constraints and regime models, or `NULL`.

estimation Options of the file's `estimation` command used by `estimate()` and `bayes_dsge()` (`presample`, `first_obs`, `nobs`).

commands List of Dynare commands found in the file (such as `stoch_simul` or `estimation`), recorded but not executed.

perfect_foresight The file's perfect-foresight setup, used by `simulate_perfect_foresight()`.

notes Character vector of translation notes, including anything that was ignored or approximated.

References

Dennis, R. (2007). Optimal policy in rational expectations models: new solution algorithms. *Macroeconomic Dynamics*, 11(1), 31-55.

Guerrieri, L. and Iacoviello, M. (2015). OccBin: A toolkit for solving dynamic models with occasionally binding constraints easily. *Journal of Monetary Economics*, 70, 22-38.

See Also

`dsgenl_model()`, `solve_dsge()`, `bayes_dsge()`, `osr()`, `simulate_occbin()`, `simulate_perfect_foresight()`

Examples

```
rbc <- read_dynare(system.file("examples", "rbc.mod", package = "dsge"))
rbc
sol <- solve_dsge(rbc)
irf(sol, periods = 20, se = FALSE)

# Model code can also be passed as text, including macro directives
ar <- read_dynare(text = "
  @#define lags = 2
  var y;
  varexo e;
  parameters rho;
  rho = 0.5;
  model;
    y = e
  @#for k in 1:lags
      + rho^@{k} * y(-@{k})
  @#endfor
    ;
  end;
  shocks;
    var e; stderr 0.01;
  end;
")
solve_dsge(ar)
```

residuals.dsge_fit *Residuals from a fitted DSGE model*

Description

Returns one-step-ahead prediction errors.

Usage

```
## S3 method for class 'dsge_fit'
residuals(object, ...)
```

Arguments

`object`	A `dsge_fit` object.
`...`	Additional arguments (currently unused).

Value

A matrix of prediction errors.

robust_vcov *Robust (sandwich) variance-covariance matrix*

Description

Computes the sandwich (Huber-White) variance-covariance matrix for ML-estimated DSGE model parameters. This provides standard errors that are robust to model misspecification.

Usage

```
robust_vcov(object, ...)
```

Arguments

`object`	A `"dsge_fit"` object estimated via ML.
`...`	Additional arguments passed to methods (e.g., `step` for numerical gradient step size).

Details

The sandwich estimator is: V_robust = inv(H) B inv(H), where H is the Hessian of the negative log-likelihood and B is the outer product of the per-observation score vectors.

Value

An object of class "dsge_robust_vcov" containing:

vcov Robust variance-covariance matrix.

se Robust standard errors.

se_conventional Conventional (Hessian-based) standard errors for comparison.

param_names Parameter names.

Examples

```
m <- dsge_model(
  obs(y ~ z),
  state(z ~ rho * z),
  start = list(rho = 0.5)
)
set.seed(1)
z <- numeric(100); for (i in 2:100) z[i] <- 0.8 * z[i-1] + rnorm(1)
fit <- estimate(m, data = data.frame(y = z))
rv <- robust_vcov(fit)
print(rv)
```

---

shock_decomposition.dsge_solution

Historical Shock Decomposition

Description

Decomposes the observed variables into the contributions of each structural shock. At each time t, the observed deviation from steady state is written as a sum of contributions from current and past shocks plus the initial condition contribution.

Usage

```
## S3 method for class 'dsge_solution'
shock_decomposition(x, data, ...)

shock_decomposition(x, ...)

## S3 method for class 'dsge_fit'
shock_decomposition(x, ...)

## S3 method for class 'dsge_bayes'
shock_decomposition(x, ...)
```

Arguments

`x`	A `dsge_fit` or `dsge_bayes` object.
`data`	Matrix or data frame of observed data with columns matching the model's observable names. Required when `x` is a `dsge_solution`.
`...`	Additional arguments (currently unused).

Details

The state-space solution gives:

$$x_t = H^t x_0 + \sum_{j=1}^{t} H^{t-j} M \varepsilon_j$$

The historical decomposition partitions the observed variables $y_t = Z x_t$ into the contribution of each structural shock $\varepsilon_j^{(k)}$ accumulated through the propagation mechanism. The sum of all contributions (including the initial condition term) reproduces the smoothed observables exactly.

Value

An object of class `"dsge_decomposition"` containing:

decomposition A 3D array with dimensions `[T, n_obs, n_shocks + 1]`. The last slice contains the initial condition contribution.

obs_names Character vector of observed variable names.

shock_names Character vector of shock names (plus "initial").

observed T x n_obs matrix of observed data (deviations).

Examples

```
m <- dsge_model(
  obs(y ~ z),
  state(z ~ rho * z),
  start = list(rho = 0.5)
)
set.seed(1)
e <- rnorm(100)
z <- numeric(100); for (i in 2:100) z[i] <- 0.8*z[i-1]+e[i]
fit <- estimate(m, data = data.frame(y = z))
hd <- shock_decomposition(fit)
plot(hd)
```

simulate_2nd_order *Simulate Using Second-Order Approximation (Pruned)*

Description

Simulates sample paths using the pruned second-order approximation following Kim, Kim, Schaumburg, and Sims (2008).

Usage

```
simulate_2nd_order(sol, n = 200L, n_burn = 100L, seed = NULL)
```

Arguments

sol	A dsge_solution object with order = 2.
n	Integer. Number of periods to simulate.
n_burn	Integer. Burn-in periods to discard. Default 100.
seed	Random seed.

Value

A list with states, controls, state_levels, control_levels matrices (n x n_vars).

simulate_3rd_order *Simulate Using Third-Order Approximation (Pruned)*

Description

Simulates sample paths using the pruned third-order approximation following Andreasen, Fernandez-Villaverde, and Rubio-Ramirez (2018).

Usage

```
simulate_3rd_order(sol, n = 200L, n_burn = 200L, seed = NULL)
```

Arguments

sol	A dsge_solution object with order = 3.
n	Integer. Number of periods to simulate.
n_burn	Integer. Burn-in periods to discard. Default 200.
seed	Random seed.

Details

Pruning tracks first-, second-, and third-order state components separately and combines them, avoiding explosive simulation paths.

Value

A list with `states`, `controls`, `state_levels`, `control_levels` (all n x n_vars matrices), plus `order = 3`.

---

simulate_occbin *Simulate with Occasionally Binding Constraints*

Description

Computes deterministic transition paths under piecewise-linear occasionally binding constraints using an iterative shadow-shock method.

Usage

```
simulate_occbin(
  x,
  constraints,
  shocks = NULL,
  initial = NULL,
  horizon = 40L,
  max_iter = 100L,
  tol = 1e-08,
  in_sd = FALSE
)
```

Arguments

`x`	A solved DSGE model object (`dsge_solution`, `dsge_fit`, or `dsge_bayes`), or a Dynare model imported with `read_dynare` that has an `occbin_constraints` block. For an imported model the constraints come from the file and the model is solved with the piecewise-linear regime algorithm of Guerrieri and Iacoviello (2015), as in Dynare's `occbin_solver`; `shocks` are raw innovations (default: the file's `shocks(surprise)` block) treated as surprises, and `constraints`, `initial`, `tol` and `in_sd` are not used.
`constraints`	A list of constraints. Each element can be: • An `obc_constraint` object (from `obc_constraint()`) • A character string like `"r >= 0"` (parsed automatically)
`shocks`	Deterministic shock specification (as in `perfect_foresight`). Named list or matrix.
`initial`	Named numeric vector of initial state deviations from steady state. Default `NULL` (start at SS).

horizon	Integer. Simulation horizon. Default 40.
max_iter	Integer. Maximum OccBin iterations. Default 100.
tol	Numeric. Convergence tolerance for shadow shocks. Default 1e-8.
in_sd	Logical. If TRUE, shock values are in standard deviations. Default FALSE.

Details

The algorithm iteratively:

1. Simulates the path with current shadow shocks
2. Identifies periods where constraints are violated
3. Computes shadow shocks to enforce constraints at binding periods
4. Removes shadow shocks at non-binding periods
5. Repeats until the binding regime stabilises

This captures the feedback effect of constraint enforcement on future dynamics through the state transition.

Value

An object of class "dsge_occbin" containing:

states Matrix of state deviations (constrained path)

controls Matrix of control deviations (constrained path)

states_unc Matrix of state deviations (unconstrained path)

controls_unc Matrix of control deviations (unconstrained path)

binding Logical matrix (horizon x n_constraints) of binding indicators

shadow_shocks Matrix of shadow shocks applied

n_iter Number of OccBin iterations to convergence

converged Logical: did the algorithm converge?

constraints List of constraint objects

steady_state Steady state values if available

horizon Simulation horizon

state_names State variable names

control_names Control variable names

Examples

```
# Simple NK model with ZLB
nk <- dsge_model(
  obs(pi ~ beta * lead(pi) + kappa * x),
  unobs(x ~ lead(x) - (r - lead(pi) - g)),
  obs(r ~ psi * pi + u),
  state(u ~ rhou * u),
  state(g ~ rhog * g),
```

```
  fixed = list(beta = 0.99, kappa = 0.1, psi = 1.5),
  start = list(rhou = 0.5, rhog = 0.5)
)
sol <- solve_dsge(nk, params = list(rhou = 0.5, rhog = 0.5),
                  shock_sd = c(u = 0.5, g = 0.5))
obc <- simulate_occbin(sol,
  constraints = list("r >= 0"),
  shocks = list(g = -0.05),
  horizon = 40)
plot(obc)
```

---

simulate_perfect_foresight

Perfect-foresight simulation of an imported Dynare model

Description

Solves the model's deterministic path exactly as Dynare's `perfect_foresight_setup` + `perfect_foresight_solver` (or `simul`) do, using the file's `initval`, `endval`, `histval` and `steady` statements and its deterministic `shocks` (with `periods`/`values`), and complementarity conditions from `mcp` equation tags (Dynare's `lmmcp` option, e.g. a zero lower bound).

Usage

```
simulate_perfect_foresight(
  x,
  periods = NULL,
  shocks = NULL,
  params = NULL,
  lmmcp = NULL,
  tol = 1e-10,
  max_iter = 100L
)
```

Arguments

`x`	A model imported with `read_dynare()`.
`periods`	Number of simulation periods. Defaults to the file's `perfect_foresight_setup(periods = )`, `simul(periods = )` or `periods` statement.
`shocks`	Optional matrix (or data frame) of exogenous values by period (rows 1, 2, ...) with columns named after shocks, replacing the file's deterministic shocks for those shocks.
`params`	Optional named parameter values overriding the calibration.
`lmmcp`	Logical: treat equations tagged `mcp` as complementarity conditions (Dynare's `lmmcp` option). Defaults to `TRUE` when a `perfect_foresight_solver` command of the file uses `lmmcp`; without it Dynare ignores the tags.

tol	Convergence tolerance on the largest equation residual.
max_iter	Maximum number of Newton iterations.

Value

An object of class "dsge_dynare_pf": a list with path (a matrix of the endogenous variables, one row per period from the initial condition(s) to the terminal one(s), as Dynare's oo_.endo_simul transposed), exo (the exogenous path, as oo_.exo_simul), periods, initial and terminal (steady states or initial/terminal values), converged, iterations and max_residual.

Examples

```
m <- read_dynare(text = "
  var c k;
  varexo a;
  parameters alpha beta delta;
  alpha = 0.33; beta = 0.99; delta = 0.025;
  model;
    1/c = beta/c(+1) * (alpha * exp(a(+1)) * k^(alpha - 1) + 1 - delta);
    k = exp(a) * k(-1)^alpha + (1 - delta) * k(-1) - c;
  end;
  initval; a = 0; k = 30; c = 2.3; end;
  steady;
  endval; a = 0.1; end;
  steady;
  perfect_foresight_setup(periods = 100);
  perfect_foresight_solver;
")
pf <- simulate_perfect_foresight(m)
head(pf$path)
```

---

smm_estimate *Simulated Method of Moments (SMM) Estimation*

Description

Estimates structural parameters by minimising the distance between simulated and empirical moments. Use when analytic model moments are intractable (e.g. perturbed nonlinear models). For linear DSGEs produced by solve_dsge, SMM and GMM give the same large-sample point estimates but SMM is noisier; prefer gmm_estimate unless you have a specific reason to simulate.

Usage

```
smm_estimate(
  model,
  data,
  moments,
  params_start,
  shock_sd_start,
```

```
  weight = NULL,
  lower = NULL,
  upper = NULL,
  method = "Nelder-Mead",
  control = list(),
  sim_periods = 1000L,
  sim_replic = 5L,
  seed = NULL
)
```

Arguments

model	A dsge_model object.
data	Matrix or data frame of observed variables; column names must match a subset of the model's observables.
moments	Character vector naming the moments to match. Each element uses the same naming convention as endogenous_prior: • "sd:<var>" – standard deviation of <var>. • "var:<var>" – variance of <var>. • "cov:<v1>:<v2>" – covariance between <v1> and <v2>. • "ac1:<var>" – lag-1 autocorrelation of <var>. • "cor:<v1>:<v2>" – contemporaneous correlation.
params_start	Named numeric vector of starting values for the structural parameters.
shock_sd_start	Named numeric vector of starting values for shock standard deviations.
weight	Optional positive-definite weighting matrix. Default identity (one-step GMM). Pass "optimal" to run a two-step estimator where the second-step weight is the inverse Newey-West covariance of the empirical moments.
lower, upper	Bounds (-Inf, Inf by default).
method	Optimiser method for optim. Default "Nelder-Mead".
control	optim control list.
sim_periods	Integer. Length of each simulated path used to compute model moments. Default 1000.
sim_replic	Integer. Number of independent simulations averaged into the moment estimates. Default 5.
seed	Optional integer seed for reproducible simulations.

Value

An object of class c("dsge_smm","dsge_gmm").

`smooth_shocks.dsge_solution`
Extract Smoothed Structural Shocks

Description

Recovers the structural shocks from the smoothed states using the state transition equation: $\hat{\varepsilon}_{t+1} = M^{+}(x_{t+1|T} - Hx_{t|T})$ where M^{+} is the Moore-Penrose pseudo-inverse of M.

Usage

```
## S3 method for class 'dsge_solution'
smooth_shocks(x, data, ...)

smooth_shocks(x, ...)

## S3 method for class 'dsge_fit'
smooth_shocks(x, ...)

## S3 method for class 'dsge_bayes'
smooth_shocks(x, ...)
```

Arguments

`x`	A `dsge_fit` or `dsge_bayes` object.
`data`	Matrix or data frame of observed data with columns matching the model's observable names. Required when `x` is a `dsge_solution`.
`...`	Additional arguments (currently unused).

Details

From the state transition $x_{t+1} = Hx_t + M\varepsilon_{t+1}$, the smoothed innovation is $x_{t+1|T} - Hx_{t|T}$. The structural shocks are recovered by projecting onto M: $\hat{\varepsilon}_{t+1} = (M'M)^{-1}M'(x_{t+1|T} - Hx_{t|T})$.

Value

An object of class `"dsge_smoothed_shocks"` containing:

shocks (T-1) x n_shocks matrix of smoothed structural shocks.

shock_names Character vector of shock names.

smooth_states.dsge_solution

Smoothed State Estimates from an Estimated DSGE Model

Description

Computes the Rauch-Tung-Striebel (RTS) smoother to produce optimal state estimates using all available observations. Compared to the filtered states (which only use past data), smoothed states also incorporate future observations.

Usage

```
## S3 method for class 'dsge_solution'
smooth_states(x, data, ...)

smooth_states(x, ...)

## S3 method for class 'dsge_fit'
smooth_states(x, ...)

## S3 method for class 'dsge_bayes'
smooth_states(x, ...)
```

Arguments

`x`	A `dsge_fit` or `dsge_bayes` object.
`data`	Matrix or data frame of observed data with columns matching the model's observable names. Required when `x` is a `dsge_solution`; ignored for `dsge_fit` / `dsge_bayes` objects (which carry their own data).
`...`	Additional arguments (currently unused).

Details

The smoother uses the state-space representation:

$$x_{t+1} = Hx_t + M\varepsilon_{t+1}$$

$$y_t = Zx_t$$

where $Z = D \cdot G$. The smoothed states are the expectation of the state vector conditional on all observations: $x_{t|T} = E[x_t|y_1, \ldots, y_T]$.

For Bayesian models, the smoother is evaluated at the posterior mean.

Value

An object of class "dsge_smoothed" containing:

smoothed_states T x n_s matrix of smoothed state estimates.

filtered_states T x n_s matrix of filtered state estimates.

smoothed_obs T x n_obs matrix of smoothed observable fits.

residuals T x n_obs matrix of observation residuals.

state_names Character vector of state variable names.

obs_names Character vector of observed variable names.

steady_state Steady-state values (if available).

Examples

```
m <- dsge_model(
  obs(y ~ z),
  state(z ~ rho * z),
  start = list(rho = 0.5)
)
set.seed(1)
e <- rnorm(100)
z <- numeric(100); for (i in 2:100) z[i] <- 0.8 * z[i-1] + e[i]
fit <- estimate(m, data = data.frame(y = z))
sm <- smooth_states(fit)
```

---

solve_dsge *Solve a Linear or Linearized DSGE Model*

Description

Computes the state-space solution of a DSGE model using the Klein (2000) method. Accepts both linear models (dsge_model) and nonlinear models (dsgenl_model). For nonlinear models, the steady state is computed and the model is linearized automatically.

Usage

```
solve_dsge(model, params = NULL, shock_sd = NULL, tol = 1e-06, order = 1L)
```

Arguments

model	A dsge_model or dsgenl_model object, or a Dynare model imported with read_dynare().
params	Named numeric vector of parameter values. If NULL, uses the model's fixed and start values (for an imported Dynare model, its calibration).

`shock_sd`	Named numeric vector of shock standard deviations. If `NULL`, defaults to 1 for all shocks (for an imported Dynare model, the standard deviations from its `shocks` block).
`tol`	Tolerance for classifying eigenvalues as stable (|lambda| < 1 + tol). Default is 1e-6.
`order`	Integer. Approximation order: 1 (default), 2 for second-order, or 3 for third-order perturbation. Orders 2 and 3 require a `dsgenl_model`.

Details

The stable solution is computed by cyclic reduction (as Dynare's `cycle_reduction` option); if that fails, for example with unit roots or a singular lead matrix, the stable deflating subspace of the model's matrix pencil is found with the inverse-free spectral divide of Bai, Demmel and Gu (1997), which needs no QZ decomposition; an undetermined-coefficients iteration is the last resort. As in Dynare, eigenvalues with modulus up to 1 + 1e-6 count as stable, so unit roots are allowed. Saddle-path stability requires that all eigenvalues of H have modulus below 1 + `tol`.

For nonlinear models, the solver first computes the deterministic steady state, then linearizes the model via first-order Taylor expansion, and finally solves the resulting linear system.

Value

An object of class `"dsge_solution"` containing:

G Policy matrix (n_controls x n_states).

H State transition matrix (n_states x n_states).

M Shock coefficient matrix (n_states x n_shocks).

D Observation selection matrix.

eigenvalues Complex vector of eigenvalues.

stable Logical: is the system saddle-path stable?

n_stable Number of stable eigenvalues.

params The parameter values used.

model Reference to the model object.

`stability` *Check Stability of DSGE Model*

Description

Checks saddle-path stability of the model by examining eigenvalues. A model is stable when the number of eigenvalues with modulus less than 1 equals the number of state variables.

Usage

```
stability(x)
```

Arguments

x	A dsge_fit or dsge_solution object.

Value

A list of class "dsge_stability" with:

stable Logical: is the system saddle-path stable?

eigenvalues Complex eigenvalue vector.

moduli Moduli of eigenvalues.

classification Character vector: "stable" or "unstable" for each.

n_stable Number of stable eigenvalues.

n_states Number of state variables (required stable count).

---

state *Define a State Variable Equation*

Description

Wraps a formula to mark it as an equation for a state variable in a linear DSGE model. State equations describe the evolution of state variables one period ahead.

Usage

```
state(formula, shock = TRUE)
```

Arguments

formula	A formula of the form variable ~ expression. The left-hand side variable represents the one-period lead of the state.
shock	Logical. If TRUE (default), an unobserved shock is attached to this state equation (exogenous state). If FALSE, no shock is attached (endogenous state / deterministic state).

Value

A list with class "dsge_equation" containing the parsed equation and its type.

See Also

obs(), unobs(), dsge_model()

steady_state — *Solve for the Deterministic Steady State*

Description

Finds the steady-state values of all model variables by solving the system of nonlinear equations with all leads set equal to current values.

Usage

```
steady_state(model, ...)

## S3 method for class 'dsgenl_model'
steady_state(
  model,
  params = NULL,
  guess = NULL,
  maxiter = 200L,
  tol = 1e-10,
  ...
)
```

Arguments

`model`	A `dsgenl_model` object.
`...`	Additional arguments (currently unused).
`params`	Named numeric vector of parameter values. If `NULL`, uses the model's fixed and start values.
`guess`	Named numeric vector of initial guesses for variable values. Overrides the model's `ss_guess`.
`maxiter`	Maximum Newton-Raphson iterations. Default is 200.
`tol`	Convergence tolerance. Default is 1e-10.

Value

An object of class `"dsgenl_steady_state"` with:

values Named numeric vector of steady-state values.

residuals Equation residuals at the solution.

params Parameter values used.

converged Logical: did the solver converge?

iterations Number of iterations used.

summary.dsge_perfect_foresight *Summary of Perfect Foresight Transition*

Description

Summary of Perfect Foresight Transition

Usage

```
## S3 method for class 'dsge_perfect_foresight'
summary(object, ...)
```

Arguments

`object`	A `dsge_perfect_foresight` object.
`...`	Additional arguments (unused).

Value

Invisibly returns the `dsge_perfect_foresight` object. Called for the side effect of printing impact effects, peak deviations, and convergence diagnostics to the console.

transition_matrix *Extract State Transition Matrix*

Description

Returns the transition matrix H from a fitted or solved DSGE model. The transition matrix describes state evolution: $x_{t+1} = Hx_t + M\varepsilon_{t+1}$.

Usage

```
transition_matrix(x, se = TRUE, level = 0.95)
```

Arguments

`x`	A `dsge_fit` or `dsge_solution` object.
`se`	Logical. If `TRUE` (default) and `x` is a `dsge_fit`, compute delta-method standard errors.
`level`	Confidence level for intervals. Default is 0.95.

Value

Same structure as `policy_matrix()`.

unobs — *Define an Unobserved Control Variable Equation*

Description

Wraps a formula to mark it as an equation for an unobserved control variable in a linear DSGE model.

Usage

```
unobs(formula)
```

Arguments

`formula`	A formula of the form `variable ~ expression`.

Value

A list with class `"dsge_equation"` containing the parsed equation and its type.

See Also

`obs()`, `state()`, `dsge_model()`

variance_decomposition — *Variance Decomposition*

Description

Computes the share of each observable variable's variance attributable to each structural shock.

Usage

```
variance_decomposition(x, horizon = NULL, ...)

## S3 method for class 'dsge_solution'
variance_decomposition(x, horizon = NULL, ...)

## S3 method for class 'dsge_fit'
variance_decomposition(x, horizon = NULL, ...)

## S3 method for class 'dsge_bayes'
variance_decomposition(x, horizon = NULL, ...)
```

Arguments

`x`	A `dsge_solution`, `dsge_fit`, or `dsge_bayes` object.
`horizon`	`NULL` (default) for the unconditional decomposition, or an integer vector of horizons for FEVD.
`...`	Additional arguments (currently unused).

Details

Two flavours are supported, controlled by the `horizon` argument:

- **Unconditional** (default, `horizon = NULL`): the decomposition of the long-run / steady-state variance. For each shock j, the state covariance contribution $\Sigma_x^{(j)}$ solves the discrete Lyapunov equation $\Sigma_x^{(j)} = H\Sigma_x^{(j)}H' + M_j\sigma_j^2 M_j'$ and the observable variance share is $\mathrm{diag}(G\Sigma_x^{(j)}G')$.
- **Forecast-error variance decomposition (FEVD)** (`horizon = 1:H`): the share of each shock in the h-step-ahead forecast-error variance for a vector of horizons. Forecast-error variance at horizon h is $\sum_{k=0}^{h-1} H^k M\Sigma_\varepsilon M'(H^k)'$.

Value

An object of class `"dsge_variance_decomposition"` containing:

contribution Either a $n_o \times n_e$ matrix (unconditional) or a $n_h \times n_o \times n_e$ array (FEVD) of variance contributions in level units (squared standard deviations).

contribution_pct Same shape as `contribution`, but normalised so that each variable's shares across shocks sum to 100 percent.

obs_names Character vector of observable variable names.

shock_names Character vector of structural shock names.

horizon The horizon argument (or `Inf` for the unconditional case).

type `"unconditional"` or `"fevd"`.

Examples

```
nk <- dsge_model(
  obs(p   ~ beta * lead(p) + kappa * x),
  unobs(x ~ lead(x) - (r - lead(p) - g)),
  obs(r   ~ psi * p + u),
  state(u ~ rhou * u),
  state(g ~ rhog * g),
  fixed = list(beta = 0.99),
  start = list(kappa = 0.1, psi = 1.5, rhou = 0.7, rhog = 0.9)
)
sol <- solve_dsge(nk,
  params   = c(kappa = 0.1, psi = 1.5, rhou = 0.7, rhog = 0.9),
  shock_sd = c(e.u = 1.0, e.g = 0.5))
vd  <- variance_decomposition(sol)
print(vd)
plot(vd)
```

```
fevd <- variance_decomposition(sol, horizon = c(1, 4, 8, 20))
plot(fevd)
```

`vcov.dsge_fit` *Robust vcov via vcov generic*

Description

When `type = "robust"` is passed to `vcov()`, returns the sandwich variance-covariance matrix.

Usage

```
## S3 method for class 'dsge_fit'
vcov(object, type = c("conventional", "robust"), ...)
```

Arguments

`object`	A `"dsge_fit"` object.
`type`	Character. `"conventional"` (default) or `"robust"`.
`...`	Passed to `robust_vcov()` when type is "robust".

Value

Variance-covariance matrix.

`welfare_loss` *Evaluate Welfare Loss Under a Given Policy*

Description

Computes the unconditional expected welfare loss for an arbitrary (possibly non-optimal) linear feedback policy $u_t = Fx_t$.

Usage

```
welfare_loss(ramsey, F_alt = NULL)
```

Arguments

`ramsey`	A `"dsge_ramsey"` object from `ramsey_policy`.
`F_alt`	Numeric matrix (n_instruments x n_s). Alternative feedback matrix to evaluate. If `NULL`, evaluates the Ramsey-optimal rule.

Value

A list with:

`welfare_loss` Scalar welfare loss.

`F` The feedback matrix used.

`stable` Logical: is the closed-loop system stable?

Index

bayes_dsge, 4, *10*, *12–14*
bayes_dsge(), *66*, *82–84*
bayes_dsge_var, 6, *10*, *31*
bayes_dsge_var_mh, 8, *31*
bayes_factor, 10
bayes_particle, 12, *59*
bayes_smc, 14

calibrated_smoother, 15
check_identification, 16
conditional_forecast, 18
conditional_forecast.dsge_dsgevar, 20
conditional_forecast.dsge_dsgevar_mh, *10*
conditional_forecast.dsge_dsgevar_mh *(*conditional_forecast.dsge_dsgevar*)*, 20

discretionary_policy, 20
dsge_model, *4*, 22, *73*
dsge_model(), *25*, *27*, *43*, *50*, *98*, *101*
dsgenl_model, *4*, 23, *65*
dsgenl_model(), *84*

E, 25
E(), *43*
endogenous_prior, 25, *35*, *93*
endogenous_prior(), *5*
estimate, *18*, 27
estimate(), *82*, *84*
extended_path, 28

fitted.dsge_fit, 30
forecast, 30
forecast.dsge_dsgevar, 31
forecast.dsge_dsgevar_mh, *10*
forecast.dsge_dsgevar_mh *(*forecast.dsge_dsgevar*)*, 31
forecast.dsge_fit, *19*, 32
forecast.dsge_fit(), *30*, *68*

geweke_test, 32
global_sensitivity, 33
gmm_estimate, 35, *92*

irf *(*irf.dsge_bayes*)*, 37
irf(), *67*, *69*
irf.dsge_bayes, 37
irf.dsge_bayes(), *67*
irf_2nd_order, 38
irf_match, 39

kalman_filter_skewed, 41, *49*

lead, 43
lead(), *25*
linearize, 43

marginal_likelihood, *11*, *12*, 44
mcmc_diagnostics, 45
model_covariance, 45
model_latex, 46
ms_filter, 48

obc_constraint, 49
obs, 50, *73*
obs(), *22*, *98*, *101*
optim, *36*, *40*, *51*, *52*, *93*
osr, *22*, 51
osr(), *83*, *84*

pac_simulate, 53, *56*
pac_target_loading, 54, *56*
pac_weights, *53*, *54*, 55
parameter_sensitivity, 56
particle_filter, *14*, *42*, 58, *59*
particle_filter_loglik, *14*, 59, *59*
perfect_foresight, 60, *62–65*
perfect_foresight_expect_err, 62
perfect_foresight_nonlinear, *61*, *63*, 64
plot(), *71*
plot.dsge_bayes, 66

plot.dsge_decomposition, 68
plot.dsge_forecast, *19*, 68
plot.dsge_irf, 69
plot.dsge_occbin, 69
plot.dsge_perfect_foresight, 70
plot.dsge_smoothed, 71
plot.dsge_variance_decomposition, 71
policy_matrix, 72
policy_matrix(), *100*
posterior_predictive, 72
predetermined, 73
predict.dsge_fit, 74
prediction_accuracy, 75
prediction_interval, 76
prior, 76
prior(), *82, 83*
prior_posterior_update, 77

ramsey_policy, *21, 22*, *52*, 79, *103*
read_dynare, *51*, 81, *89*
read_dynare(), *4, 5*, *27*, *91*, *96*
residuals.dsge_fit, 85
robust_vcov, 85

shock_decomposition, *16*
shock_decomposition
(shock_decomposition.dsge_solution), 86
shock_decomposition.dsge_solution, *16*, 86
simulate_2nd_order, 88
simulate_3rd_order, 88
simulate_occbin, 89
simulate_occbin(), *83, 84*
simulate_perfect_foresight, 91
simulate_perfect_foresight(), *83, 84*
smm_estimate, 92
smooth_shocks, *16*
smooth_shocks
(smooth_shocks.dsge_solution), 94
smooth_shocks.dsge_solution, *16*, 94
smooth_states, *16*
smooth_states
(smooth_states.dsge_solution), 95
smooth_states.dsge_solution, *16*, 95
solve_dsge, *16*, *41*, *48*, *51*, *59*, *80*, *92*, 96
solve_dsge(), *22*, *83, 84*

stability, 97
state, *73*, 98
state(), *22*, *50*, *101*
stats::optim(), *27*
steady_state, 99
summary.dsge_perfect_foresight, 100

transition_matrix, 100

unobs, *73*, 101
unobs(), *22*, *50*, *98*

variance_decomposition, 101
vcov.dsge_fit, 103

welfare_loss, *22*, *52*, *80*, 103