# bekk Documentation

***Release 0.1***

**Stanislav Khrapov**

**Mar 01, 2018**

# Contents

This module allows to simulate and estimate the BEKK(1,1) model proposed in[1].

The model assumes that demeaned returns $u_t$ are conditionally normal:

$$u_t = e_t H_t^{1/2}, \quad e_t \sim N(0, I),$$

with variance matrix evolving accrding to the following recursion:

$$H_t = CC' + Au_{t-1}u'_{t-1}A' + BH_{t-1}B'.$$

[1] Robert F. Engle and Kenneth F. Kroner "Multivariate Simultaneous Generalized Arch", Econometric Theory, Vol. 11, No. 1 (Mar., 1995), pp. 122-150, <http://www.jstor.org/stable/3532933>

CHAPTER 1

# References

CHAPTER 2

# Notes

Check this repo for related R library: https://github.com/vst/mgarch/

Alternative optimization library: http://www.pyopt.org/

# CHAPTER 3

# Contents

## 3.1 Parameter classes

### 3.1.1 Generic parameter class

**class** bekk.param_generic.**ParamGeneric**(*nstocks=2*, *abstart=(0.1*, *0.85)*, *target=None*)

Class to hold parameters of the BEKK model.

#### Attributes

| | |
|---|---|
| **amat, bmat, cmat** | Matrix representations of BEKK parameters |

#### Methods

| | |
|---|---|
| *from_abc*([amat, bmat, cmat]) | Initialize from A, B, and C arrays. |
| *find_cmat*([amat, bmat, ccmat, target]) | Find C matrix given A, B, and H. |
| *from_target*([amat, bmat, target]) | Initialize from A, B, and variance target. |
| *find_stationary_var*([amat, bmat, cmat]) | Find fixed point of H = CC' + AHA' + BHB' given A, B, C. |
| *get_uvar*() | Unconditional variance matrix regardless of the model. |
| *constraint*() | Compute the largest eigenvalue of BEKK model. |

**constraint**()

Compute the largest eigenvalue of BEKK model.

**Returns** float

Largest eigenvalue

**static find_ccmat**(*amat=None*, *bmat=None*, *target=None*)
Find CC' matrix given A, B, and H in H = CC' + AHA' + BHB' given A, B, H.

**Parameters** **amat, bmat, target** : (nstocks, nstocks) arrays

Parameter matrices

**Returns** (nstocks, nstocks) array

**static find_cmat**(*amat=None*, *bmat=None*, *ccmat=None*, *target=None*)
Find C matrix given A, B, and H. Solve for C in H = CC' + AHA' + BHB' given A, B, H.

**Parameters** **amat, bmat, target** : (nstocks, nstocks) arrays

Parameter matrices

**Returns** (nstocks, nstocks) array

Cholesky decomposition of CC'

**static find_stationary_var**(*amat=None*, *bmat=None*, *cmat=None*)
Find fixed point of H = CC' + AHA' + BHB' given A, B, C.

**Parameters** **amat, bmat, cmat** : (nstocks, nstocks) arrays

Parameter matrices

**Returns** (nstocks, nstocks) array

Unconditional variance matrix

**static fixed_point**(*uvar*, *amat=None*, *bmat=None*, *ccmat=None*)
Function for finding fixed point of H = CC' + AHA' + BHB' given A, B, C.

**Parameters** **uvar** : 1d array

Lower triangle of symmetric variance matrix

**amat, bmat, ccmat** : (nstocks, nstocks) arrays

Parameter matrices

**Returns** (nstocks, nstocks) array

**classmethod from_abc**(*amat=None*, *bmat=None*, *cmat=None*)
Initialize from A, B, and C arrays.

**Parameters** **amat, bmat, cmat** : (nstocks, nstocks) arrays

Parameter matrices

**Returns** **param** : BEKKParams instance

BEKK parameters

**classmethod from_target**(*amat=None*, *bmat=None*, *target=None*)
Initialize from A, B, and variance target.

**Parameters** **amat, bmat, target** : (nstocks, nstocks) arrays

Parameter matrices

**Returns** **param** : BEKKParams instance

BEKK parameters

**get_uvar**()
Unconditional variance matrix regardless of the model.

**Returns** (nstocks, nstocks) array

Unconditional variance amtrix

**penalty**()
Penalty in the likelihood for bad parameter values.

**uvar_bad**()
Check that unconditional variance is well defined.

### 3.1.2 Standadrd parameter class

**class** bekk.param_standard.**ParamStandard**(*nstocks=2*, *abstart=(0.1*, *0.6)*, *target=None*)
Class to hold parameters of the BEKK model.

#### Attributes

| | |
|---|---|
| **amat, bmat, cmat** | Matrix representations of BEKK parameters |

#### Methods

| | |
|---|---|
| *from_theta*([theta, nstocks, cfree, ...]) | Initialize from theta vector. |
| *get_theta*([restriction, use_target, cfree]) | Convert parameter mratrices to 1-dimensional array. |

**classmethod from_theta**(*theta=None*, *nstocks=None*, *cfree=True*, *restriction='scalar'*, *target=None*)
Initialize from theta vector.

**Parameters** **theta** : 1d array

Length depends on the model restrictions and variance targeting

**If target is not None:**

- 'full' - 2*n**2
- 'diagonal' - 2*n
- 'scalar' - 2

**If target is None:**

- +(n+1)*n/2 for parameter C

**nstocks** : int

Number of stocks in the model

**restriction** : str

**Can be**

- 'full'
- 'diagonal'
- 'scalar'

**target** : (nstocks, nstocks) array

Variance target matrix

**Returns** **param** : BEKKParams instance

BEKK parameters

**get_theta**(*restriction='scalar'*, *use_target=True*, *cfree=True*)

Convert parameter mratrices to 1-dimensional array.

**Parameters** **restriction** : str

**Can be**

- 'full'
- 'diagonal'
- 'scalar'

**use_target** : bool

Whether to estimate only A and B (True) or C as well (False)

**Returns** **theta** : 1d array

Length depends on the model restrictions and variance targeting

**If use_target is True:**

- 'full' - 2*n**2
- 'diagonal' - 2*n
- 'scalar' - 2

**If use_target is False:**

- +(n+1)*n/2 for parameter cmat

### 3.1.3 Spatial parameter class

**class** bekk.param_spatial.**ParamSpatial**(*nstocks=2*)

Class to hold parameters of the BEKK model.

#### Attributes

| | |
|---|---|
| **amat, bmat, cmat, avecs, bvecs, dvecs** | Matrix representations of BEKK parameters |
| **groups** | List of related items |
| **weights** | Spatial relation matrices |

#### Methods

| | |
|---|---|
| *from_theta*([theta, groups, cfree, ...]) | Initialize from theta vector. |
| *get_theta*([restriction, use_target, cfree]) | Convert parameter matrices to 1-dimensional array. |
| *get_weight*([groups]) | Generate weighting matrices given groups. |

**classmethod from_theta**(*theta=None*, *groups=None*, *cfree=False*, *restriction='shomo'*, *target=None*, *solve_dvecs=False*)

Initialize from theta vector.

**Parameters** **theta** : 1d array

Length depends on the model restrictions and variance targeting

**weights** : (ncat, nstocks, nstocks) array

Weight matrices

**groups** : list of lists of tuples

Encoded groups of items

**cfree** : bool

Whether to leave C matrix free (True) or not (False)

**target** : (nstocks, nstocks) array

Variance target matrix

**restriction** : str

**Can be**

- 'hetero' (heterogeneous)
- 'ghomo' (group homogeneous)
- 'homo' (homogeneous)
- 'shomo' (scalar homogeneous)

**Returns** **param** : BEKKParams instance

BEKK parameters

**get_theta**(*restriction='shomo'*, *use_target=False*, *cfree=False*)

Convert parameter matrices to 1-dimensional array.

**Parameters** **restriction** : str

**Can be**

- 'hetero' (heterogeneous)
- 'ghomo' (group homogeneous)
- 'homo' (homogeneous)
- 'shomo' (scalar homogeneous)

**use_target** : bool

Whether to estimate only A and B (True) or C as well (False)

**cfree** : bool

Whether to leave C matrix free (True) or not (False)

**Returns** **theta** : 1d array

Length depends on the model restrictions and variance targeting

**If use_target is True:**

- 'hetero' - 2*n*(m+1)

- 'ghomo' - 2*(n+m*k)
- 'homo' - 2*(n+m)
- 'shomo' - 2*(m+1)

**If use_target is False and cfree is False:**

- 'hetero' - +n*(m+1)
- 'ghomo' - +(n+m*k)
- 'homo' - +(n+m)
- 'shomo' - +(n+m)

**If use_target is False and cfree is True:**

- +n*(n+1)/2

**static get_weight**(*groups=None*)

Generate weighting matrices given groups.

**Parameters** **groups** : list of lists of tuples

Encoded groups of items

**Returns** (ngroups, nitems, nitems) array

Spatial weights

### Examples

```
>>> print(ParamSpatial.get_weight(groups=[[(0, 1)]]))
[[[ 0.  1.]
  [ 1.  0.]]]
>>> print(ParamSpatial.get_weight(groups=[[(0, 1, 2)]]))
[[[ 0.   0.5  0.5]
  [ 0.5  0.   0.5]
  [ 0.5  0.5  0. ]]]
>>> print(ParamSpatial.get_weight(groups=[[(0, 1), (2, 3)]]))
[[[ 0.  1.  0.  0.]
  [ 1.  0.  0.  0.]
  [ 0.  0.  0.  1.]
  [ 0.  0.  1.  0.]]]
>>> print(ParamSpatial.get_weight(groups=[[(0, 1), (2, 3, 4)]]))
[[[ 0.   1.   0.   0.   0. ]
  [ 1.   0.   0.   0.   0. ]
  [ 0.   0.   0.   0.5  0.5]
  [ 0.   0.   0.5  0.   0.5]
  [ 0.   0.   0.5  0.5  0. ]]]
```

## 3.2 BEKK estimation

Estimation is performed using Quasi Maximum Likelihood (QML) method. Specifically, the individual contribution to the Gaussian log-likelihood is

$$l_t(\theta) = -\ln|H_t| - u_t' H_t^{-1} u_t.$$

**class** bekk.bekk_estimation.**BEKK**(*innov*)

BEKK estimation class.

### Attributes

| | |
|---|---|
| **innov** | Return innovations |
| **hvar** | Condiational variance |

### Methods

| | |
|---|---|
| *estimate*([param_start, restriction, cfree, ...]) | Estimate parameters of the BEKK model. |
| *collect_losses*([param_start, innov_all, ...]) | Collect forecast losses using rolling window. |

**static collect_losses**(*param_start=None*, *innov_all=None*, *window=1000*, *model='standard'*, *use_target=False*, *groups=('NA', 'NA')*, *restriction='scalar'*, *cfree=False*, *method='SLSQP'*, *use_penalty=False*, *ngrid=5*, *alpha=0.05*, *kind='equal'*, *tname='losses'*, *path=None*)

Collect forecast losses using rolling window.

**Parameters** **param_start** : ParamStandard or ParamSpatial instance

Initial parameters for estimation

**innov_all: (nobs, nstocks) array**

Inovations

**window** : int

Window length for in-sample estimation

**model** : str

Specific model to estimate.

**Must be**

- 'standard'
- 'spatial'

**restriction** : str

Restriction on parameters.

**Must be**

- 'full' = 'diagonal'
- 'group' (for 'spatial' model only)
- 'scalar'

**groups** : tuple

First item is the string code. Second is spatial groups specification.

**use_target** : bool

Whether to use variance targeting (True) or not (False)

**cfree** : bool

Whether to leave C matrix free (True) or not (False)

**ngrid** : int

Number of starting values in one dimension

**use_penalty** : bool

Whether to include penalty term in the likelihood

**alpha** : float

Risk level. Usually 1% or 5%.

**kind** : str

Portfolio weighting scheme. Either 'equal' or 'minvar' (minimum variance).

**tname** : str

Name to be used while writing data to the disk

**Returns** float

Average loss_frob function

**estimate**(*param_start=None*, *restriction='scalar'*, *cfree=False*, *use_target=False*, *model='standard'*, *groups=None*, *method='SLSQP'*, *cython=True*, *use_penalty=False*)

Estimate parameters of the BEKK model.

**Parameters** **param_start** : ParamStandard or ParamSpatial instance

Starting parameters. See Notes for more details.

**model** : str

Specific model to estimate.

**Must be**

- 'standard'
- 'spatial'

**restriction** : str

Restriction on parameters.

**Must be**

- 'full'
- 'diagonal'
- 'group' (only applicable with 'spatial' model)
- 'scalar'

**use_target** : bool

Whether to use variance targeting (True) or not (False)

**cfree** : bool

Whether to leave C matrix free (True) or not (False)

**groups** : list of lists of tuples

Encoded groups of items

**method** : str

Optimization method. See scipy.optimize.minimize

**cython** : bool

Whether to use Cython optimizations (True) or not (False)

**use_penalty** : bool

Whether to include penalty term in the likelihood

**Returns** BEKKResults instance

Estimation results object

### Notes

If no param_start is given, the program will estimate parameters in the order 'from simple to more complicated' (from scalar to diagonal to full) while always using variance targeting.

## 3.3 BEKK results

**class** bekk.bekk_results.**BEKKResults**(*innov=None*, *hvar=None*, *var_target=None*, *model=None*, *use_target=None*, *restriction=None*, *cfree=None*, *method=None*, *cython=None*, *time_delta=None*, *param_start=None*, *param_final=None*, *opt_out=None*)

Estimation results.

### Attributes

| | |
|---|---|
| **innov** | Return innovations |
| **hvar** | Filtered variance matrices |
| **var_target** | Estimated varinace target |
| **param_start** | Starting parameters |
| **param_final** | Estimated parameters |
| **model** | Specific model to estimate |
| **restriction** | Restriction on parameters |
| **use_target** | Variance targeting flag |
| **method** | Optimization method. See scipy.optimize.minimize |
| **cython** | Whether to use Cython optimizations (True) or not (False) |

### Methods

| | |
|---|---|
| *loss_var_ratio*([kind]) | Ratio of realized and predicted variance. |
| *portf_evar*([kind]) | Portfolio predicted variance. |
| *portf_mvar*([kind]) | Portfolio mean variance. |
| *portf_rvar*([kind]) | Portfolio predicted variance. |
| *weights*([kind]) | Portfolio weights. |

Continued on next page

Table 3.5 – continued from previous page

| | |
|---|---|
| *weights_equal*() | Equal weights. |
| *weights_minvar*() | Minimum variance weights. |

**loss_var_ratio**(*kind='equal'*)
Ratio of realized and predicted variance.

**Parameters** **kind** : str

Either 'equal' or 'minvar' (minimum variance).

**Returns** (nobs, ) array

**portf_evar**(*kind='equal'*)
Portfolio predicted variance.

**Parameters** **kind** : str

Either 'equal' or 'minvar' (minimum variance).

**Returns** (nobs, ) array

**portf_mvar**(*kind='equal'*)
Portfolio mean variance.

**Parameters** **kind** : str

Either 'equal' or 'minvar' (minimum variance).

**Returns** float

**portf_rvar**(*kind='equal'*)
Portfolio predicted variance.

**Parameters** **kind** : str

Either 'equal' or 'minvar' (minimum variance).

**Returns** (nobs, ) array

**weights**(*kind='equal'*)
Portfolio weights.

**Parameters** **weight** : str

Either 'equal' or 'minvar' (minimum variance).

**Returns** (nobs, nstocks) array

**weights_equal**()
Equal weights.

**Returns** (nobs, nstocks) array

**weights_minvar**()
Minimum variance weights.

**Returns** (nobs, nstocks) array

## 3.4 Data generation

bekk.generate_data.**simulate_bekk**(*param*, *nobs=1000*, *distr='normal'*, *degf=10*, *lam=0*)
Simulate data.

**Parameters** **param** : BEKKParams instance

Attributes of this class hold parameter matrices

**nobs** : int

Number of observations to generate. Time series length

**distr** : str

Name of the distribution from which to generate innovations.

**Must be**

- 'normal'
- 'student'
- 'skewt'

**degf** : int

Degrees of freedom for Student or SkewStudent distributions

**lam** : float

Skewness parameter for Student or SkewStudent distributions. Must be between (-1, 1)

**Returns** **innov** : (nobs, nstocks) array

Multivariate innovation matrix

bekk.generate_data.**download_data**(*tickers=None*, *nobs=None*, *start='2002-01-01'*, *end='2015-12-31'*)

Download stock market data and save it to disk.

**Parameters** **tickers** : list of str

Tickers to download

**nobs** : int

Number of observations in the time series

**start** : str

First observation date

**end** : str

Last observation date

**Returns** **ret** : DataFrame

Demeaned returns

# Python Module Index

## b

# Index