Computationally Efficient Gaussian Maximum Likelihood Methods for Vector ARFIMA Models

Rebecca J. Sela
New York University

July 24, 2008

Contents

1 Introduction

This document summarizes the functions in the `R` code associated with the paper "Computationally Efficient Methods for Two Multivariate Fractionally Integrated Models" (Sela and Hurvich). This code is available online at `http://pages.stern.nyu.edu/~rsela/VARFI/code.html`. For questions or to report bugs, please contact `rsela@stern.nyu.edu`. All code is provided "as is," without warranty, and for academic and non-profit use only.

In this documentation, we follow the `R` documentation convention of writing the default value of a variable in brackets next to its name. If there is no default value, a value must be supplied.

2 Functions for Maximum Likelihood Estimation

Function: `FIVAR.GPH`

- This function computes the GPH estimator of the differencing parameters for K time series.
- Inputs:
  - `X` - vector of length KT containing the observations, grouped by series
  - `K` - number of series
  - `numPoints` [$T^{0.4}$] - number of points to use in the log periodogram regression
  - `verbose` [FALSE] - whether the regression output should be printed
- Output: vector of GPH estimates of d

Function: `FIVAR.loglikelihood`

- This function computes the log likelihood of the given $FIVAR(1, d)$ model.
- Note: For some extreme parameter values, the determinant calculation method returns NaN or a very small number. In that case, we return MinVal and print a warning.
- Inputs:
  - `X` - vector of length KT containing the observations, grouped by series
  - `A1` - $K \times K$ matrix describing the $VAR(1)$ matrix
  - `Sigma` - $K \times K$ covariance matrix of the innovations
  - `d` - vector of length K containing the differencing parameters

  - `Mean` [(0,...,0)] - vector of length K containing the means of each series
  - `determinant` [TRUE] - if TRUE, the likelihood uses the regression approximation for the determinant; if FALSE, the likelihood uses the naive approximation
  - `MinVal` [-1E10] - the "minimum value" which is returned if the determinant computation fails or the computed value is less than the minimum value
- Output: the log likelihood of the given parameters based on the given data

Function: `FIVAR.MLE`

- This function computes the maximum likelihood estimator for a $FIVAR(1, d)$ model.
- Inputs:
  - `X` - vector of length KT containing the observations, grouped by series
  - `K` - number of series
  - `init` [NULL] - initial parameter values, given as a list with elements named any of the following:
    * `A1` - $K \times K$ matrix describing the $VAR(1)$ matrix
    * `Sigma` - $K \times K$ covariance matrix of the innovations
    * `d` - vector of length K containing the differencing parameters
  - `determinant` [TRUE] - if TRUE, the likelihood uses the regression approximation for the determinant; if FALSE, the likelihood uses the naive approximation
  - `demean` [FALSE] - whether all the series should be demeaned before estimation begins
  - `verbose` [FALSE] - whether the parameters should be printed at each step of estimation
  - `boundaryDistance` [0.01] - describes how close to the boundaries (-0.5 and 0.5 for d, 1 for the singular values of A_1); we also bound parameters values (such as variances) $1/boundaryDistance$ instead of ∞
  - `computeHessian` [FALSE] - should the Hessian for the original parameter values be computed?
- Output: a list containing the estimated parameter values and the log likelihood at those parameter values

Function: `FIVAR.spectrum`

- This function computes the spectrum of a $FIVAR(1, d)$ process at a given frequency.
- Inputs:

  - `nu` - frequency at which the spectrum should be computed
  - `A1` - $K \times K$ matrix describing the $VAR(1)$ matrix
  - `Sigma` - $K \times K$ covariance matrix of the innovations
  - `d` - vector of length K containing the differencing parameters
- Output: the value of the spectrum at that point

Function: `FIVAR.WhittleLikelihood`

- This function takes in the cross-periodogram of the data and the parameters of the model and returns the Whittle approximation to the log likelihood for a $FIVAR(1, d)$ model.
- Note: For some extreme parameter values, the calculation method returns NaN, a very small number, or a number with a non-trivial imaginary part. In that case, we return MinVal and print a warning.
- Inputs:
  - `pgram` - $K \times K \times T$ array containing the cross-periodogram of the data
  - `A` - $K \times K$ matrix describing the $VAR(1)$ matrix
  - `Sigma` - $K \times K$ covariance matrix of the innovations
  - `d` - vector of length K containing the differencing parameters
  - `MinVal` [-1E10] - the "minimum value" which is returned if the determinant computation fails or the computed value is less than the minimum value
  - `zeroCutoff` [1E-12] - the largest value for which the imaginary part of the likelihood can be considered "negligible" (if the imaginary part is larger, then a warning is printed and `MinVal` is returned)
- Output: the Whittle approximation to the log likelihood of the given parameters based on the given data

Function: `FIVAR.Whittle`

- This function computes the Whittle estimator for a $FIVAR(1, d)$ model.
- Inputs:
  - `X` - vector of length KT containing the observations, grouped by series
  - `K` - number of series
  - `init` [NULL] - initial parameter values, given as a list with elements named any of the following:

    - `A1` - $K \times K$ matrix describing the VAR(1) matrix
    - `Sigma` - $K \times K$ covariance matrix of the innovations
    - `d` - vector of length K containing the differencing parameters
  - `verbose` [FALSE] - whether the parameters should be printed at each step of estimation
  - `demean` [FALSE] - whether all the series should be demeaned before estimation begins
  - `boundaryDistance` [0.01] - describes how close to the boundaries (-0.5 and 0.5 for d, 1 for the singular values of A_1); we also bound parameters values (such as variances) $1/boundaryDistance$ instead of ∞
  - `computeHessian` [FALSE] - should the Hessian for the original parameter values be computed?
- Output: a list containing the estimated parameter values and the Whittle approximation to the log likelihood at those parameter values

Function: `FIVAR.SowellLogLikelihood`

- This function computes the log likelihood of the given $FIVAR(1, d)$ model, using Sowell's version of the Durbin-Levinson algorithm.
- Inputs:
  - `X` - vector of length KT containing the observations, grouped by series
  - `A1` - $K \times K$ matrix describing the $VAR(1)$ matrix
  - `Sigma` - $K \times K$ covariance matrix of the innovations
  - `d` - vector of length K containing the differencing parameters
  - `determinant` [TRUE] - if TRUE, the likelihood uses the exact value for the determinant; if FALSE, the likelihood uses the naive approximation
- Output: the log likelihood of the given parameters based on the given data

Function: `FIVAR.Sowell`

- This function computes the maximum likelihood estimator for a $FIVAR(1, d)$ model, using Sowell's version of the Durbin-Levinson algorithm.
- Inputs:
  - `X` - vector of length KT containing the observations, grouped by series
  - `K` - number of series

- `init` [NULL] - initial parameter values, given as a list with elements named any of the following:
  - `A1` - $K \times K$ matrix describing the $VAR(1)$ matrix
  - `Sigma` - $K \times K$ covariance matrix of the innovations
  - `d` - vector of length K containing the differencing parameters
- `determinant` [TRUE] - if TRUE, the likelihood uses the exact value for the determinant; if FALSE, the likelihood uses the naive approximation
- `demean` [FALSE] - whether all the series should be demeaned before estimation begins
- `verbose` [FALSE] - whether the parameters should be printed at each step of estimation
- `boundaryDistance` [0.01] - describes how close to the boundaries (-0.5 and 0.5 for d, 1 for the singular values of A_1); we also bound parameters values (such as variances) $1/boundaryDistance$ instead of ∞

- Output: a list containing the estimated parameter values and the log likelihood at those parameter values

Function: `VARFI.loglikelihood`

- This function computes the log likelihood of the given $VARFI(1, d)$ model.
- Note: For some extreme parameter values, the determinant calculation method returns NaN or a very small number. In that case, we return MinVal and print a warning.
- Inputs:
  - `X` - vector of length KT containing the observations, grouped by series
  - `A1` - $K \times K$ matrix describing the $VAR(1)$ matrix
  - `Sigma` - $K \times K$ covariance matrix of the innovations
  - `d` - vector of length K containing the differencing parameters
  - `Mean` [(0,...,0)] - vector of length K containing the means of each series
  - `determinant` [TRUE] - if TRUE, the likelihood uses the regression approximation for the determinant; if FALSE, the likelihood uses the naive approximation
  - `MinVal` [-1E10] - the "minimum value" which is returned if the determinant computation fails or the computed value is less than the minimum value
- Output: the log likelihood of the given parameters based on the given data

Function: `VARFI.MLE`

- This function computes the maximum likelihood estimator for a $VARFI(1, d)$ model.
- Inputs:
  - `X` - vector of length KT containing the observations, grouped by series
  - `K` - number of series
  - `init` [NULL] - initial parameter values, given as a list with elements named any of the following:
    - `A1` - $K \times K$ matrix describing the $VAR(1)$ matrix
    - `Sigma` - $K \times K$ covariance matrix of the innovations
    - `d` - vector of length K containing the differencing parameters
  - `determinant` [TRUE] - if TRUE, the likelihood uses the regression approximation for the determinant; if FALSE, the likelihood uses the naive approximation
  - `demean` [FALSE] - whether all the series should be demeaned before estimation begins
  - `verbose` [FALSE] - whether the parameters should be printed at each step of estimation
  - `boundaryDistance` [0.01] - describes how close to the boundaries (-0.5 and 0.5 for d, 1 for the singular values of A_1); we also bound parameters values (such as variances) $1/boundaryDistance$ instead of ∞
  - `computeHessian` [FALSE] - should the Hessian for the original parameter values be computed?
- Output: a list containing the estimated parameter values and the log likelihood at those parameter values

Function: `VARFI.spectrum`

- This function computes the spectrum of a $VARFI(1, d)$ process at a given frequency.
- Inputs:
  - `nu` - frequency at which the spectrum should be computed
  - `A1` - $K \times K$ matrix describing the $VAR(1)$ matrix
  - `Sigma` - $K \times K$ covariance matrix of the innovations
  - `d` - vector of length K containing the differencing parameters
- Output: the value of the spectrum at that point

Function: `VARFI.WhittleLikelihood`

- This function takes in the cross-periodogram of the data and the parameters of the model and returns the Whittle approximation to the log likelihood of a $VARFI(1, d)$.
- Note: For some extreme parameter values, the determinant calculation method returns NaN or a very small number. In that case, we return MinVal and print a warning.
- Inputs:
  - `pgram` - $K \times K \times T$ array containing the cross-periodogram of the data
  - `A` - $K \times K$ matrix describing the $VAR(1)$ matrix
  - `Sigma` - $K \times K$ covariance matrix of the innovations
  - `d` - vector of length K containing the differencing parameters
  - `MinVal` [-1E10] - the "minimum value" which is returned if the determinant computation fails or the computed value is less than the minimum value
  - `zeroCutoff` [1E-12] - the largest value for which the imaginary part of the likelihood can be considered "negligible" (if the imaginary part is larger, then a warning is printed and `MinVal` is returned)
- Output: the Whittle approximation to the log likelihood of the given parameters based on the given data

Function: `VARFI.Whittle`

- This function computes the Whittle estimator for a $VARFI(1, d)$ model.
- Note: For some extreme parameter values, the calculation method returns NaN, a very small number, or a number with a non-trivial imaginary part. In that case, we return MinVal and print a warning.
- Inputs:
  - `X` - vector of length KT containing the observations, grouped by series
  - `K` - number of series
  - `init` [NULL] - initial parameter values, given as a list with elements named any of the following:
    - `A1` - $K \times K$ matrix describing the $VAR(1)$ matrix
    - `Sigma` - $K \times K$ covariance matrix of the innovations
    - `d` - vector of length K containing the differencing parameters
  - `verbose` [FALSE] - whether the parameters should be printed at each step of estimation
  - `demean` [FALSE] - whether all the series should be demeaned before estimation begins

  - `boundaryDistance` [0.01] - describes how close to the boundaries (-0.5 and 0.5 for d, 1 for the singular values of A_1); we also bound parameters values (such as variances) $1/boundaryDistance$ instead of ∞
  - `computeHessian` [FALSE] - should the Hessian for the original parameter values be computed?
- Output: a list containing the estimated parameter values and the Whittle approximation to the log likelihood at those parameter values

Function: `VARFI.SowellLogLikelihood`

- This function computes the log likelihood of the given $VARFI(1, d)$ model, using Sowell's version of the Durbin-Levinson algorithm.
- Inputs:
  - `X` - vector of length KT containing the observations, grouped by series
  - `A1` - $K \times K$ matrix describing the $VAR(1)$ matrix
  - `Sigma` - $K \times K$ covariance matrix of the innovations
  - `d` - vector of length K containing the differencing parameters
  - `determinant` [TRUE] - if TRUE, the likelihood uses the exact value for the determinant; if FALSE, the likelihood uses the naive approximation
- Output: the log likelihood of the given parameters based on the given data

Function: `VARFI.Sowell`

- This function computes the maximum likelihood estimator for a $VARFI(1, d)$ model, using Sowell's version of the Durbin-Levinson algorithm.
- Inputs:
  - `X` - vector of length KT containing the observations, grouped by series
  - `K` - number of series
  - `init` [NULL] - initial parameter values, given as a list with elements named any of the following:
    - `A1` - $K \times K$ matrix describing the $VAR(1)$ matrix
    - `Sigma` - $K \times K$ covariance matrix of the innovations
    - `d` - vector of length K containing the differencing parameters
  - `determinant` [TRUE] - if TRUE, the likelihood uses the exact value for the determinant; if FALSE, the likelihood uses the naive approximation

- `demean` [FALSE] - whether all the series should be demeaned before estimation begins
- `verbose` [FALSE] - whether the parameters should be printed at each step of estimation
- `boundaryDistance` [0.01] - describes how close to the boundaries (-0.5 and 0.5 for d, 1 for the singular values of A_1); we also bound parameters values (such as variances) $1/boundaryDistance$ instead of ∞

- Output: a list containing the estimated parameter values and the log likelihood at those parameter values

3 Functions for Computing Autocovariances

Function: `FIVAR.autoCov`

- This function takes in the parameters of a $FIVAR(1, d)$ process and returns the specified number of autocovariances. Computations use the splitting method.
- Inputs:
  - `F` [NULL] - $K \times K$ matrix of with the $VAR(1)$ coefficients (NULL implies 0)
  - `Sigma` - $K \times K$ symmetric and positive definite covariance matrix of the errors
  - `d` - vector of length K of differencing parameters
  - `n` - number of lags returned (from $-(n-1)$ to $n-1$)
  - `tol` [1E-10] - maximum error allowed in the autocovariances
  - `verbose` [FALSE] - determines whether the number of lags of the VAR covariances used is printed
- Output: a $K \times K \times n$ array, with the r^{th} autocovariance at $(\cdot, \cdot, r+n)$

Function: `Sowell.autoCov`

- This function takes in the parameters of a $FIVAR(1, d)$ process and returns the specified number of autocovariances. Computations use the closed form with hypergeometric functions, given by Sowell (1989).
- Inputs:
  - `F` [NULL] - $K \times K$ matrix of with the $VAR(1)$ coefficients (NULL implies 0)
  - `Sigma` - $K \times K$ symmetric and positive definite covariance matrix of the errors

  - `d` - vector of length K of differencing parameters
  - `n` - number of lags returned (from $-(n-1)$ to $n-1$)
- Output: a $K \times K \times n$ array, with the r^{th} autocovariance at $(\cdot, \cdot, r+n)$

Function: `FIVAR.integralCov`

- This function takes in the parameters of a $FIVAR(1,d)$ process and returns the specified number of autocovariances. Computations use numerical integration methods.
- Inputs:
  - `F` [NULL] - $K \times K$ matrix of with the $VAR(1)$ coefficients (NULL implies 0)
  - `Sigma` - $K \times K$ symmetric and positive definite covariance matrix of the errors
  - `d` - vector of length K of differencing parameters
  - `n` - number of lags returned (from $-(n-1)$ to $n-1$)
- Output: a $K \times K \times n$ array, with the r^{th} autocovariance at $(\cdot, \cdot, r+n)$

Function: `VARFI.autoCov`

- This function takes in the parameters of a $VARFI(1,d)$ process and returns the specified number of autocovariances. Computations use the splitting method.
- Inputs:
  - `F` [NULL] - $K \times K$ matrix of with the $VAR(1)$ coefficients (NULL implies 0)
  - `Sigma` - $K \times K$ symmetric and positive definite covariance matrix of the errors
  - `d` - vector of length K of differencing parameters
  - `n` - number of lags returned (from $-(n-1)$ to $n-1$)
  - `tol` [1E-10] - maximum error allowed in the autocovariances
  - `verbose` [FALSE] - determines whether the number of lags of the VAR covariances used is printed
- Output: a $K \times K \times n$ array, with the r^{th} autocovariance at $(\cdot, \cdot, r+n)$

Function: `VARFI.integralCov`

- This function takes in the parameters of a $VARFI(1,d)$ process and returns the specified number of autocovariances. Computations use numerical integration methods.

- Inputs:
  - `F` [NULL] - $K \times K$ matrix of with the $VAR(1)$ coefficients (NULL implies 0)
  - `Sigma` - $K \times K$ symmetric and positive definite covariance matrix of the errors
  - `d` - vector of length K of differencing parameters
  - `n` - number of lags returned (from $-(n-1)$ to $n-1$)
- Output: a $K \times K \times n$ array, with the r^{th} autocovariance at $(\cdot, \cdot, r+n)$

Function: `Cointegrate`

- This function takes in an array of covariances for an uncointegrated model and a cointegrating matrix and returns the covariances of the cointegrated model.
- Inputs:
  - `oldCovs` - $K \times K \times (2n-1)$ array of covariances of the uncointegrated model
  - `coMatrix` - cointegrating matrix
- Output: $K \times K \times (2n-1)$ array of covariances of the cointegrated model

3.1 Functions for Computing Autocovariances in Special Cases

Function: `VAR.cov0`

- This function computes the unconditional covariance (autocovariance at lag 0) of an $VAR(1)$ process described by $K \times K$ lag coefficient matrix F and $K \times K$ error covariance matrix Σ.
- Inputs:
  - `F` - $K \times K$ matrix of coefficients on lagged variables
  - `Sigma` - $K \times K$ symmetric and positive definite covariance matrix of the errors
- Output: $K \times K$ matrix with the unconditional covariance

Function: `VAR.autoCov`

- This function computes the first M autocovariances of a $VAR(1)$ process described by $K \times K$ lag coefficient matrix F and KxK error covariance matrix Sigma.

- Inputs:
  - `F` - $K \times K$ matrix of coefficients on lagged variables
  - `Sigma` - $K \times K$ symmetric and positive definite covariance matrix of the errors
  - `M` - number of autocovariances returned (from 0 to $M-1$)
  - `known0` [NULL] - autocovariance at lag 0, if already known
- Output: a $K \times K \times M$ array, with the r^{th} autocovariance at $(\cdot, \cdot, r+1)$

Function: `ARFIMA.MAcoeff`

- This function computes the j^{th} moving average coefficient in the MA infinity expansion of an $ARFIMA(0, d, 0)$ model. The formula is: $\Gamma(j-d)/\Gamma(j+1)\Gamma(d)$.
- Inputs:
  - `j` - the lag of the coefficient in the MA expansion
  - `d` - the differencing parameter of the ARFIMA process
- Output: The j^{th} moving average coefficient

Function: `ARFIMA.autoCov`

- This function computes the lag-h autocovariance of an $ARFIMA(0, d, 0)$ model.
- Inputs:
  - `h` - lag of the autocovariance
  - `d` - differencing parameter
  - `sigmasq` [1] - innovation variance
- Output: The autocovariance of an $ARFIMA(0, d, 0)$ at lage h.

Function: `ARFIMA.crossCov`

- This computes the cross-covariance of two $ARFIMA(0, d, 0)$ processes driven by the same white noise, using Sowell's (1989) expression.
- Inputs:
  - `d1` - differencing parameter for the first process
  - `d2` - differencing parameter for the second process

  - `m` - the autocovariance lag to compute
  - `sigmasq` [1] - the variance of the innovations
- Output: The cross-covariance of two ARFIMA processes driven by the same innovations.

Function: `FIVAR.autoCov0`

- This function takes in the parameters of a $FIVAR(0, d)$ process and returns the specified number of autocovariances
- Inputs:
  - `Sigma` - $K \times K$ symmetric and positive definite covariance matrix of the errors
  - `d` - vector of length K of differencing parameters
  - `n` - number of lags returned (from $-(n-1)$ to $n-1$)
- Output: a $K \times K \times (2n-1)$ array, with the r^{th} autocovariance at $(\cdot, \cdot, r+1)$

Function: `VARFI.autoCov0`

- This function takes in the parameters of a $VARFI(0, d)$ process and returns the specified number of autocovariances. (This is identical to `FIVAR.autoCov0`.)
- Inputs:
  - `Sigma` - $K \times K$ symmetric and positive definite covariance matrix of the errors
  - `d` - vector of length K of differencing parameters
  - `n` - number of lags returned (from $-(n-1)$ to $n-1$)
- Output: a $K \times K \times (2n-1)$ array, with the r^{th} autocovariance at $(\cdot, \cdot, r+1)$

4 Functions for Simulation

Function: `FIVAR.simulationSetup`

- This function sets up everything necessary to simulate from a $FIVAR(1, d)$ process.
- Inputs:
  - `T` - number of periods to simulate

  - `d` - vector of differencing parameters
  - `Sigma` - innovation variance matrix
  - `F` [NULL] - $VAR(1)$ parameter; if it is NULL, the parameter is a matrix of zeroes
  - `verbose` [FALSE] - whether intermediate messages be printed (such as if the number of lags used in the circulant embedding matrix must be increased)
  - `retry` [2] - the number of times that the function should compute additional covariances to make the circulant embedding positive definite if it fails the first time
- Output: the array of blocks needed for FFT simulation, or FALSE if a positive definite circulant embedding was not found

Function: `VARFI.simulationSetup`

- This function sets up everything necessary to simulate from a $VARFI(1, d)$ process.
- Inputs:
  - `T` - number of periods to simulate
  - `d` - vector of differencing parameters
  - `Sigma` - innovation variance matrix
  - `F` [NULL] - VAR(1) parameter; if it is NULL, the parameter is a matrix of zeroes
  - `verbose` [FALSE] - whether intermediate messages be printed (such as if the number of lags used in the circulant embedding matrix must be increased)
  - `retry` [2] - the number of times that the function should compute additional covariances to make the circulant embedding positive definite if it fails the first time
- Output: the array of blocks needed for FFT simulation, or FALSE if a positive definite circulant embedding was not found

Function: `FFTsimulate`

- This function simulates T periods of a k-variate process, given the array of blocks defining the block circulant embedding of the covariance matrix.
- Inputs:
  - `T` - number of periods to simulate
  - `SqrtBlocks` - a $K \times K \times M$ array containing the square roots of the blocks of the transformed block circulant embedding ($M > 2T$)
- Output: a vector of length KT containing the simulated values, grouped by series

Function: `SetUpBlockSqrt`

- This function takes in a block Toeplitz covariance matrix (in the form of a $K \times K \times (2T-1)$ array) and returns a $K \times K \times (2T-1)$ array in which the r^{th} level is the square root matrix needed for simulation. It checks that the circulant embedding matrix implicit in the computation is positive definite and returns FALSE if it is not. This can be used for simulating from general multivariate processes where the covariance structure is known.
- Inputs:
  - `covarray` - $K \times K \times (2T-1)$ array of covariances
  - `verbose` [FALSE] - if true, the function will print information if the positivity constraint fails
- Output: $K \times K \times (2T-1)$ array of square roots used for simulation

5 Functions for Computing Quadratic Forms and Using the Preconditioned Gradient Algorithm

Function: `QuadraticFormWithPCG`

- This function computes the quadratic form, $x'V^{-1}x$ using the multivariate preconditioned gradient algorithm, when V is a block Toeplitz matrix, with K^2 blocks of size $T \times T$ and x is of length KT.
- Inputs:
  - `firstrowcol` - a $K \times K \times T$ or $K \times K \times (2T-1)$ matrix, in which the vector describing the first row (and possibly column) of the (i, j) block is in $(i, j, \cdot)$
  - `x` - $KT \times 1$ vector
  - `symmetric` [FALSE] - indicator of whether the the Toeplitz matrix is being specified by only the first row, since it is symmetric
- Output: $x'V^{-1}x$

Function: `MultivariatePCG`

- This is the function for the multivariate preconditioned conjugate gradient algorithm. This function solves the system $Ax = b$ for x, when A is a block Toeplitz matrix (with K^2 blocks of size $T \times T$). This takes in a $K \times K \times T$ or $K \times K \times (2T-1)$ matrix containing the first

row OR the first column and row of each block of the block Toeplitz matrix. (If the first row is specified, then the matrix is assumed to be symmetric. Otherwise, the first row and column must be specified, in this order: $(a(T,1), a(T-1,1), ..., a(1,1), a(1,2), ..., a(1,T))$. This corresponds to giving the covariances from lag $-(T-1)$ to lag $(T-1)$ in a time series context.)

- Inputs:
  - `firstrowcol` - a $K \times K \times T$ or $K \times K \times (2T-1)$ matrix, in which the vector describing the first row (and possibly column) of the (i,j) block is in $(i,j,\cdot)$
  - `x` - $KT \times 1$ vector
  - `symmetric` [FALSE] - indicator of whether the the Toeplitz matrix is being specified by only the first row, since it is symmetric
  - `verbose` [FALSE] - if this is TRUE, the number of iterations used in the PCG algorithm will be printed during the computation
- Output: $A^{-1}b$

Function: `MultivariateCG`

- This is the function for the multivariate conjugate gradient algorithm (without preconditioning). This function solves the system $Ax = b$ for x, when A is a block Toeplitz matrix (with K^2 blocks of size $T \times T$). This takes in a $K \times K \times T$ or $K \times K \times (2T-1)$ matrix containing the first row OR the first column and row of each block of the block Toeplitz matrix. (If the first row is specified, then the matrix is assumed to be symmetric. Otherwise, the first row and column must be specified, in this order: $(a(T,1), a(T-1,1), ..., a(1,1), a(1,2), ..., a(1,T))$. This corresponds to giving the covariances from lag $-(T-1)$ to lag $(T-1)$ in a time series context.)
- Inputs:
  - `firstrowcol` - a $K \times K \times T$ or $K \times K \times (2T-1)$ matrix, in which the vector describing the first row (and possibly column) of the (i,j) block is in $(i,j,\cdot)$
  - `x` - $KT \times 1$ vector
  - `symmetric` [FALSE] - indicator of whether the the Toeplitz matrix is being specified by only the first row, since it is symmetric
  - `verbose` [FALSE] - if this is TRUE, the number of iterations used in the PCG algorithm will be printed during the computation
- Output: $A^{-1}b$

Function: `Sowell.quadraticForm`

- This function computes the quadratic form, $X'\Omega^{-1}X$, given X and either the array of covariances that determine Σ or the Sowell matrices implied by Sigma.
- Inputs:
  - `X` - vector of length nK, grouped by TIME (not series)
  - `covarray` [NULL] - array of covariances that describes Sigma
  - `sowellMatrices` [NULL] - list of list of matrices from the Sowell algorithm (`Sowell.allMatrices`)
- Output: $X'\Omega^{-1}X$

6 Functions for Computing Determinants

Function: `FIVAR.logdeterminant`

- This function computes the log determinant of the covariance matrix of n realizations of a $FIVAR(1, d)$ process, to a given level of accuracy (in log terms). The model can be specified either by the array of covariances (`covarray`) or by specifying the parameters, Σ, F, and d.
- Inputs:
  - `n` - sample size
  - `covarray` [NULL] - the array of covariances that describe the model
  - `Sigma` [NULL] - innovation variance matrix
  - `d` [NULL] - vector of differencing parameters
  - `F` [NULL] - $VAR(1)$ parameter; if it is NULL, the parameter is a matrix of zeroes
  - `verbose` [FALSE] - whether the regression that is used for approximation be printed
- Output: the determinant of the $Kn \times Kn$ covariance matrix of all the observations

Function: `VARFI.logdeterminant`

- This function computes the log determinant of the covariance matrix of n realizations of a $VARFI(1, d)$ process, to a given level of accuracy (in log terms). The model can be specified either by the array of covariances (`covarray`) or by specifying the parameters, Σ, F, and d.
- Inputs:
  - `n` - sample size

  - `covarray` [NULL] - the array of covariances that describe the model
  - `Sigma` [NULL] - innovation variance matrix
  - `d` [NULL] - vector of differencing parameters
  - `F` [NULL] - $VAR(1)$ parameter; if it is NULL, the parameter is a matrix of zeroes
  - `verbose` [FALSE] - whether the regression that is used for approximation be printed
- Output: the determinant of the $Kn \times Kn$ covariance matrix of all the observations

Function: `Sowell.determinant`

- This function computes the determinant, $|\Omega|$, given X and either the array of covariances that determine Ω or the Sowell matrices implied by Sigma.
- Inputs:
  - `X` - vector of length nK, grouped by TIME (not series)
  - `covarray` [NULL] - array of covariances that describes Sigma
  - `sowellMatrices` [NULL] - list of list of matrices from the Sowell algorithm (`Sowell.allMatrices`)
- Output: $|\Omega|$

Function: `RegressionApprox`

- This function uses Sowell's algorithm to compute the initial determinants of the prediction variances and the PCG algorithm to compute some later ones. Then, it runs the regression of $j\sqrt(|v(j)|)$ on j and uses this function to interpolate the remaining determinants. It returns the entire sequence of approximate log determinants (which will presumably be added together to get the answer).
- Inputs:
  - `n` - Maximum prediction variance determinant to be calculated
  - `covarray` - the $K \times K \times (2n+1)$ array containing the autocovariances from lags $-n$ to n
  - `initialSowell` [$\min(32, n-1)$] - how many Sowell matrices should be computed for the initial part of the regression
  - `PCGpoints` [(n-1)] - a list of points at which PCG should be used to compute the exact value
  - `verbose` [FALSE] - whether the regression model used to fit the intermediate points should be printed
- Output: a vector containing the prediction variance determinants from 0 to n-1

7 Other Useful Functions

Function: `CrossPeriodogram`

- This function takes in a vector of length KT and returns the cross-periodogram of the data.
- Inputs:
  - `X` - KT vector of data
  - `K` - number of time series in X
- Output: an array of size $T \times K \times K$ containing the cross-periodogram of the data

Function: `FIVAR.predict`

- This function predicts h lags ahead for a FIVAR process, using the relationship: $E(X_{T+h}|X) = Cov(X, X_{T+h})Cov(X)^{-1}X$, where $X = (X_1, ..., X_T)'$.
- Inputs:
  - `X` - vector of length KT containing the observations
  - `lags` [1] - a vector containing all of the number of steps ahead that the prediction should be computed
  - `paramlist` - a list containing the parameters:
    - `A1` - $K \times K$ matrix containing the autoregressive parameter
    - `Sigma` - $K \times K$ innovation variance matrix
    - `d` - vector of differencing parameters of length K
    - `Mean` - vector of means of length K
- Output: a matrix of size K x length(lags), in which the predictions are given in the same order as the lags were given

Function: `VARFI.predict`

- This function predicts h lags ahead for a VARFI process, using the relationship: $E(X_{T+h}|X) = Cov(X, X_{T+h})Cov(X)^{-1}X$, where $X = (X_1, ..., X_T)'$.
- Inputs:
  - `X` - vector of length KT containing the observations

  - `lags` [1] - a vector containing all of the number of steps ahead that the prediction should be computed
  - `paramlist` - a list containing the parameters:
    * `A1` - $K \times K$ matrix containing the autoregressive parameter
    * `Sigma` - $K \times K$ innovation variance matrix
    * `d` - vector of differencing parameters of length K
    * `Mean` - vector of means of length K

- Output: a matrix of size K x length(lags), in which the predictions are given in the same order as the lags were given

8 Other Functions Used Within the Code

In this section, we give brief descriptions of other functions found in the code. These functions are not likely to be useful for data analysis, but some users might find them helpful. Please contact the author for more information.

- `fftmultwitheigen`: Used to multiply a vector by a circulant, given the eigenvalues of the circulant.
- `blockcircmult`: Used to multiply a vector by a block circulant matrix, given the eigenvalues of each block.
- `blocktoepmult`: Used to multiply a block Toeplitz matrix by a vector, given the eigenvalues of the circulant embedding of each block.
- `circembedFromFirstRow`: Computes the first row of a circulant embedding of a symmetric Toeplitz matrix, given the first row.
- `circembedFromFirstRowAndColumn`: Computes the first row of a circulant embedding of a Toeplitz matrix, given the first row and column.
- `toepmultFromFirstRow`: Multiplies a Toeplitz matrix by a vector, given the first row of a symmetric Toeplitz matrix.
- `circapproxFromFirstRow`: Computes the circulant approximation of a symmetric Toeplitz matrix, given the first row.
- `circapproxFromFirstRowAndColumn`: Computes the circulant approximation of a Toeplitz matrix, given the first row and column.
- `multicg`: Performs the conjugate gradient algorithm (without preconditioning) to compute $A^{-1}X$, given the eigenvalues of the circulant embedding of A.

- `multipcg`: Performs the preconditioned conjugate gradient algorithm to compute $A^{-1}X$, given the eigenvalue of the block circulant embedding of A and the eigenvalues of the block circulant preconditioner.
- `HyperSum`: Computes the hypergeometric function, for use in Sowell's algorithm for computing the autocovariances of a FIVAR process.
- `SowellC`: Computes all of the terms $C(d_i, d_j, h, \rho)$ for Sowell's autocovariance computation algorithm.
- `Sowell.AllMatrices`: Computes the decomposition of a block Toeplitz matrix (specified by its first rows and columns) using the Durbin-Levinson algorithm, as described by Sowell (1989).
- `Sowell.AllMatricesContinuation`: Computes the decomposition of a block Toeplitz matrix (specified by its first rows and columns) using the Durbin-Levinson algorithm, as described by Sowell (1989), given that the decomposition has been computed for some of the initial lags.
- `MatrixSqrt`: Computes the square root of a matrix, using the eigendecomposition.
- `MatrixSqrt`: Computes the inverse of the square root of a matrix, using the eigendecomposition.
- `convolution`: Computes all possible convolutions of two series, where at least one product is defined.
- `ListOfDeterminants`: Takes in a list of matrices and returns a vector containing their determinants.
- `PCGpredvar`: Uses the preconditioned conjugate gradient algorithm to compute the prediction variance, $v(r)$, for a given r.
- `ReconstructFromCholesky`: Computes the original matrix from the vectorized version of its Cholesky decomposition.
- `ConvertToCholesky`: Converts a positive definite matrix to a vector containing its Cholesky decomposition, by row.
- `ConvertToAK`: Converts a matrix with singular value less than a given maximum singular value and returns the scaled Ansley-Kohn representation that can range over all possible matrices.
- `ConstructFromAK`: Takes in the modified Ansley-Kohn representation and returns the original matrix.
- `ParseParameters`: Takes in a vector containing a representation of the parameters of a FIVAR or VARFI model and returns a list containing the original parameters.

- `CombineParameters`: Takes in the parameters of a VARFI or FIVAR process and returns a vector containing a representation of the parameters.
- `ParseParametersCoint`: Takes in a vector containing a representation of the parameters of a cointegrated FIVAR model and returns a list containing the original parameters.
- `CombineParametersCoint`: Takes in the parameters of a cointegrated FIVAR process and returns a vector containing a representation of the parameters.
- `VectorToMatrix`: Takes in a vector of T observations for each of K time series, grouped by series, and returns a $K \times T$ matrix with one column for each time series.
- `SmallBlockCircEmbed`: Compute a circulant embedding for a block Toeplitz matrix, without repeating the element on the main diagonal of the original matrix (for use in simulation).
- `SimulateFourierNormal`: Simulates multivariate normal random variables with a covariance matrix equal to the product of the Fourier matrix and its conjugate transpose.
- `SortByTime`: Takes in a vector which is grouped by series and returns a vector with the same data grouped by time.
- `SortBySeries`: Takes in a vector which is grouped by time and returns a vector with the same data grouped by series.