# Quantitative Macroeconomics

# Winter 2023/24

# Week 6

Willi Mutschler
willi@mutschler.eu

Version: 1.0
Latest version available on: GitHub

# Contents

**1. Matrix Algebra** **1**

**2. Understanding multivariate time series concepts** **2**

**3. Dimensions and VAR(1) representation** **3**

**A. Solutions** **5**

# 1. Matrix Algebra

Let

$$A = \begin{pmatrix} 0.5 & 0 & 0 \\ 0.1 & 0.1 & 0.3 \\ 0 & 0.2 & 0.3 \end{pmatrix} \qquad \Sigma_u = \begin{pmatrix} 2.25 & 0 & 0 \\ 0 & 1 & 0.5 \\ 0 & 0.5 & 0.74 \end{pmatrix} \qquad R = \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix}$$

1. Compute the eigenvalues of A. What would this imply for the system $y_t = c + Ay_{t-1} + u_t$ with $u_t$ being white noise?

2. Consider the matrices D: $m \times n$, E: $n \times p$ and F: $p \times k$. Show that

$$vec(DEF) = \left(F' \otimes D\right) vec(E),$$

   where $\otimes$ is the Kronecker product and $vec$ the vectorization operator either on paper or using a symbolic toolbox.

3. Show that R is an orthogonal matrix. Why is this matrix called a rotation matrix?

4. Compute a regular lower triangular matrix $W \in \mathbb{R}^{3\times3}$ and a diagonal matrix $\Sigma_\varepsilon \in \mathbb{R}^{3\times3}$ such that $\Sigma_u = W\Sigma_\varepsilon W'$.

   Hint: Use the Cholesky factorization $\Sigma_u = PP' = W\Sigma_\varepsilon^{\frac{1}{2}}(W\Sigma_\varepsilon^{\frac{1}{2}})'$.

5. Solve the discrete Lyapunov matrix equation $\Sigma_y = A\Sigma_y A' + \Sigma_u$ using
   a) the Kronecker product and vectorization
   b) the following iterative algorithm:

$$\begin{aligned} \Sigma_{y,0} = I, A_0 = A, \Sigma_{u,0} = \Sigma_u \\ \Sigma_{y,i+1} = A_i \Sigma_{y,i} A_i' + \Sigma_{u,i} \\ \Sigma_{u,i+1} = A_i \Sigma_{u,i} A_i' + \Sigma_{u,i} \\ A_{i+1} = A_i A_i \end{aligned}$$

      Write a loop until either a maximal number of iterations (say 500) is reached or each element of $\Sigma_{y,i+1} - \Sigma_{y,i}$ is less than $10^{-25}$ in absolute terms.
   c) Compare both approaches for A and $\Sigma_u$ given above.

**Readings:**

- E. W. Anderson et al. (1996, Ch. 4.2)
- B. Anderson and Moore (1979, Ch. 6.7)
- Lütkepohl (2005, App. A)
- Uribe and Schmitt-Grohe (2017, Ch. 4.10)

## 2. Understanding multivariate time series concepts

Let $y_t$ be a K-dimensional time series and $u_t$ a K-dimensional white noise process.

1. Why are we concerned with multivariate time series? For example, why do we model the VAR(1) process

$$\begin{pmatrix} y_{1,t} \\ y_{2,t} \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} + \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} y_{1,t-1} \\ y_{2,t-1} \end{pmatrix} + \begin{pmatrix} u_{1,t} \\ u_{2,t} \end{pmatrix}$$

   simultaneously instead of two models for each variable separately?

2. Interpret $E[y_t]$ and $\Gamma_h := Cov[y_t, y_{t-h}]$.

3. How does the definition of covariance stationary change in the multivariate case?

4. Consider the following VAR(1) process $y_t = c + Ay_{t-1} + u_t$

$$\begin{pmatrix} y_{1,t} \\ y_{2,t} \\ y_{3,t} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0.5 & 0 & 0 \\ 0.1 & 0.1 & 0.3 \\ 0 & 0.2 & 0.3 \end{pmatrix} \begin{pmatrix} y_{1,t-1} \\ y_{2,t-1} \\ y_{3,t-1} \end{pmatrix} + \begin{pmatrix} u_{1,t} \\ u_{2,t} \\ u_{3,t} \end{pmatrix}$$

   provided that $u_t \sim WN(0, \Sigma_u)$ and

$$\Sigma_u = \begin{pmatrix} 2.25 & 0 & 0 \\ 0 & 1 & 0.5 \\ 0 & 0.5 & 0.74 \end{pmatrix}$$

   Compute the coefficients $\Phi_0, \Phi_1, \cdots \in \mathbb{R}^{3\times 3}$ of the lag polynomial $\Phi(L) := \sum_{i=0}^{\infty} \Phi_i L^i$, and a $\nu \in \mathbb{R}^3$ such that we get the following $MA(\infty)$ process:

$$y_t = \nu + \Phi(L)u_t$$

### Readings

- Lütkepohl (2005, Ch. 2)

## 3. Dimensions and VAR(1) representation

Let $y_t$ be a K-dimensional covariance stationary random vector. Consider the VAR(p)-process

$$y_t = \nu + \sum_{i=1}^{p} A_i y_{t-i} + u_t$$

1. What are the dimensions of $\nu, A_i$ and $u_t$?
2. Assume that $E(u_t) = 0; E(u_t u_t') = \Sigma_u$ with $\Sigma_u$ being symmetric and positive definite. Which additional assumptions do we need to assure that $u_t$ is a multivariate white noise process?
3. Consider a VAR(2) model with $K = 4$ variables and a constant term. How many parameters do we need to estimate?
4. Show how to represent a VAR(3) model as a VAR(1) model. Hint: stack $y_t, y_{t-1}$ and $y_{t-2}$ into a vector.
5. Write a function to compute the "companion VAR(1) form" of any VAR(p) model with constant term.

### Readings

- Kilian and Lütkepohl (2017, Ch. 2.2)
- Lütkepohl (2005, Ch. 2)

## References


Anderson, Brian and John Moore (1979). *Optimal Filtering*. Dover ed., unabridged republ. Dover Books on Engineering. Mineola, NY: Dover Publ. ISBN: 978-0-486-43938-9.

Anderson, Evan W. et al. (1996). "Mechanics of Forming and Estimating Dynamic Linear Economies". In: *Handbook of Computational Economics*. Vol. 1. Elsevier, pp. 171–252. ISBN: 978-0-444-89857-9. URL: https://doi.org/10.1016/S1574-0021(96)01006-4.

Kilian, Lutz and Helmut Lütkepohl (2017). *Structural Vector Autoregressive Analysis*. Themes in Modern Econometrics. Cambridge: Cambridge University Press. ISBN: 978-1-107-19657-5. URL: https://doi.org/10.1017/9781108164818.

Lütkepohl, Helmut (2005). *New Introduction to Multiple Time Series Analysis*. Berlin: Springer. ISBN: 978-3-540-40172-8.

Uribe, Martin and Stephanie Schmitt-Grohe (2017). *Open Economy Macroeconomics*. Princeton, NJ: Princeton University Press. ISBN: 978-0-691-15877-8.

# A. Solutions

## 1 Solution to Matrix Algebra

1. progs/matlab/matrixAlgebraEigenvalues.m

```
% ------------------------------------------------------------------------
% Compute Eigenvalues of a Matrix
% ------------------------------------------------------------------------
% Willi Mutschler, November 16, 2021
% willi@mutschler.eu
% ------------------------------------------------------------------------
clearvars; clc; close all;

A = [0.5,   0,   0;
     0.1, 0.1, 0.3;
     0,   0.2, 0.3];

EV_A = eig(A);
disp(abs(EV_A)<1);
```

In the univariate AR(1) model we would check whether the autocorrelation coefficient is between -1 and 1, i.e. whether $|\phi| = |A| < 1$ such that $\sum_{j=0}^{\infty}(AL)^j = 1/(1-AL)$ where $L$ is the lag operator. In the multivariate case, we want to check the same thing, i.e. $\sum_{j=0}^{\infty}(AL)^j = (1-AL)^{-1}$. Note that $A$ is a square matrix and taking the power of a matrix is not a trivial task. One convenient way to do so, is to consider an eigenvalue decomposition (if it exists):

$$A = Q\Lambda Q^{-1}$$

where $Q$ is a square matrix whose columns contain the eigenvectors $q_i$ corresponding to the eigenvalues $\lambda_i$ found on the diagonal of $\Lambda = [\lambda_i]_{ii}$. Moreover, $\Lambda$ is a diagonal matrix and $Q$ is an orthogonal matrix $Q^{-1} = Q'$. Using this decomposition one can show that it is very easy to compute any power of a matrix:

- matrix inverse $A^{-1} = Q\Lambda^{-1}Q^{-1}$ where the inverse of $[\Lambda^{-1}]_{ii} = 1/\lambda_i$ is very easy to calculate as it is a diagonal matrix
- matrix powers: $A^2 = (Q\Lambda Q^{-1})(Q\Lambda Q^{-1}) = Q\Lambda(Q^{-1}Q)\Lambda Q^{-1} = Q\Lambda^2 Q^{-1}$ or more generally: $A^j = Q\Lambda^j Q^{-1}$.

So, for $\sum_{j=0}^{\infty}(AL)^j = (1-AL)^{-1}$ we need that $\lim_{j\to\infty}\Lambda^j = 0$. As this is a diagonal matrix, the task simplifies as we only need to look at each eigenvalue whether it is between -1 and 1: $|\lambda_i| < 1$. In other words, for VAR(1) systems $y_t = c + Ay_{t-1} + u_t$ we need to check whether the eigenvalues of $A$ are inside the unit circle. If they are, then the VAR(1) model is said to be both stable and covariance-stationary.

2. Example for vectorization and Kronecker product:

$$\underbrace{vec\begin{pmatrix}1&3&2\\1&0&0\\1&2&2\end{pmatrix}}_{3\times 3} = \underbrace{\begin{pmatrix}1\\1\\1\\3\\0\\2\\2\\0\\2\end{pmatrix}}_{9\times 1}, \qquad \underbrace{\begin{pmatrix}1&3&2\\1&0&0\\1&2&2\end{pmatrix}}_{3\times 3} \otimes \underbrace{\begin{pmatrix}0&5\\5&0\\1&1\end{pmatrix}}_{3\times 2} = \underbrace{\begin{pmatrix} 1\cdot\begin{pmatrix}0&5\\5&0\\1&1\end{pmatrix} & 3\cdot\begin{pmatrix}0&5\\5&0\\1&1\end{pmatrix} & 2\cdot\begin{pmatrix}0&5\\5&0\\1&1\end{pmatrix} \\ 1\cdot\begin{pmatrix}0&5\\5&0\\1&1\end{pmatrix} & 0\cdot\begin{pmatrix}0&5\\5&0\\1&1\end{pmatrix} & 0\cdot\begin{pmatrix}0&5\\5&0\\1&1\end{pmatrix} \\ 1\cdot\begin{pmatrix}0&5\\5&0\\1&1\end{pmatrix} & 2\cdot\begin{pmatrix}0&5\\5&0\\1&1\end{pmatrix} & 2\cdot\begin{pmatrix}0&5\\5&0\\1&1\end{pmatrix} \end{pmatrix}}_{9\times 6}$$

Using this definition, we can show that $vec(DEF) = (F' \otimes D)vec(E)$ using e.g. a symbolic toolbox:

progs/matlab/matrixAlgebraKroneckerFormula.m

```
% ------------------------------------------------------------------------
% Show that vec(D*E*F) = kron(F',D)*vec(E) using MATLAB's symbolic toolbox
% ------------------------------------------------------------------------
% Willi Mutschler, November 16, 2021
% willi@mutschler.eu
% ------------------------------------------------------------------------
clearvars; clc; close all;

%% Some basics on the symbolic toolbox
syms x % create symbolic variable
% note that matlab does not simplify or expand by default! example:
f1 = (x - 2)^2;
f2 = x^2 - 4*x + 4;
expand(f1)
simplify(f2)
isequal(f1,f2)
isequal(expand(f1),f2)
isequal(f1,simplify(f2)) % note that simplify usually takes longer than expand

%% Show that vec(D*E*F) = kron(F',D)*vec(E)
dim = randi([1 10],1,4); % generate 4 random integers between 1 and 10 as
    dimensions
% create symbolic matrices
D = sym('d',[dim(1) dim(2)]);
E = sym('e',[dim(2) dim(3)]);
F = sym('f',[dim(3) dim(4)]);
DEF = D*E*F; % check whether matrix product is defined
vecDEF = DEF(:); %vectorization

% correct: compare expanded symbolic expressions
if isequal(expand(vecDEF),expand(kron(transpose(F),D)*E(:)))
    fprintf('Expanded expressions are identical\n');
else
    error('Expanded expressions are not identical');
end
```

Of course you can do this on paper as well:

$$
\begin{aligned}
DEF &= D \begin{pmatrix} e_1 & e_2 & \cdots & e_p \end{pmatrix} \begin{pmatrix} f_{11} & f_{12} & \cdots & f_{1k} \\ f_{21} & f_{22} & \cdots & f_{2k} \\ \vdots & \vdots & \vdots & \vdots \\ f_{p1} & f_{p2} & \cdots & f_{pk} \end{pmatrix} \\
&= D \underbrace{\begin{pmatrix} e_1 f_{11} + e_2 f_{21} + \cdots + e_p f_{p1}, & e_1 f_{12} + e_2 f_{22} + \cdots + e_p f_{p2}, & \ldots, & e_1 f_{1k} + e_2 f_{2k} + \cdots + e_p f_{pk} \end{pmatrix}}_{n \times k}
\end{aligned}
$$

Vectorizing:

$$vec(DEF) = \begin{pmatrix} f_{11}De_1 + f_{21}De_2 + \cdots + f_{p1}De_p \\ f_{12}De_1 + f_{22}De_2 + \cdots + f_{p2}De_p \\ \vdots \\ f_{1k}De_1 + f_{2k}De_2 + \cdots + f_{pk}De_p \end{pmatrix} = \begin{pmatrix} f_{11}D & f_{21}D & \cdots & f_{p1}D \\ f_{12}D & f_{22}D & \cdots & f_{p2}D \\ \vdots & \vdots & \vdots & \vdots \\ f_{1k}D & f_{2k}D & \cdots & f_{pk}D \end{pmatrix} \begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_p \end{pmatrix}$$
$$= (F' \otimes D)\, vec(E)$$

3. An orthogonal matrix is characterized by $R' = R^{-1}$ and therefore $R'R = RR' = I$. Here:

$$R'R = \begin{pmatrix} (\cos(\phi))^2 + (\sin(\phi))^2 & -\cos(\phi)\sin(\phi) + \sin(\phi)\cos(\phi) \\ -\sin(\phi)\cos(\phi) + \cos(\phi)\sin(\phi) & (\sin(\phi))^2 + (\cos(\phi))^2 \end{pmatrix}$$

with $(\cos(\phi))^2 + (\sin(\phi))^2 = 1$ (so-called trigonometric Pythagoras).

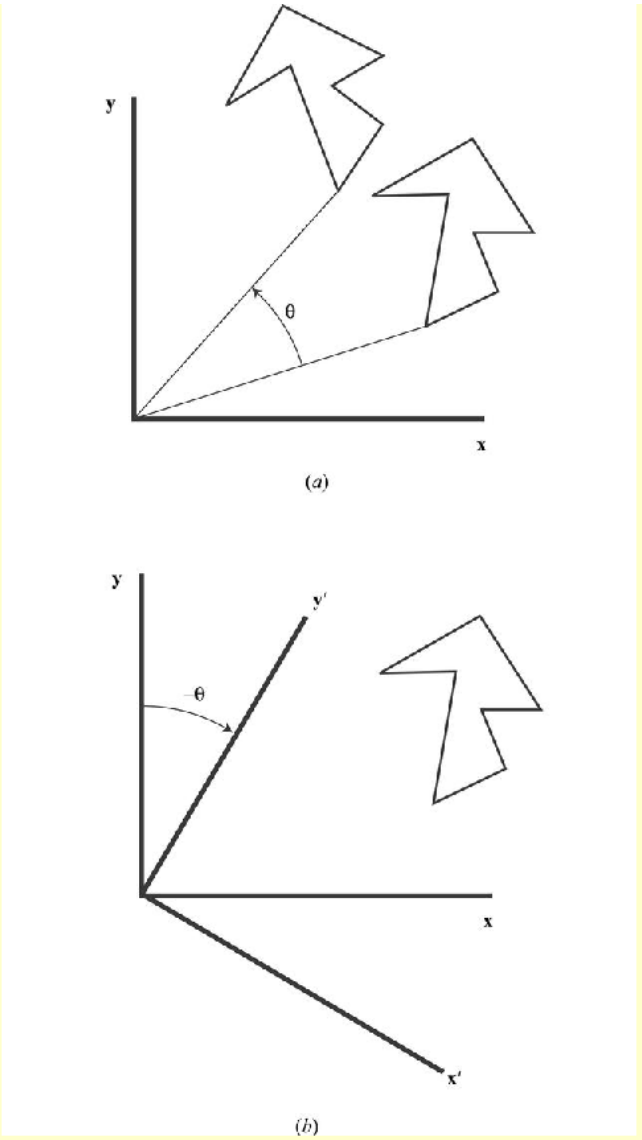
progs/matlab/matrixAlgebraRotation.m

```
% ------------------------------------------------------------------------
% Show orthogonality of 2-dimensional rotation matrix using MATLAB's
% symbolic toolbox
% ------------------------------------------------------------------------
% % Willi Mutschler, November 16, 2021
% willi@mutschler.eu
% ------------------------------------------------------------------------
clearvars; clc; close all;

theta = sym('theta');
R = [cos(theta), -sin(theta);
     sin(theta), cos(theta)];

simplify(transpose(R)*R)
simplify(R*transpose(R))
Rinv = R\eye(size(R,1));
simplify(transpose(R) - Rinv)
```

$R$ is called a rotation matrix, because it rotates vectors or objects in the Euclidian space without stretching or shrinking the object.




In this example the matrix R rotates the vector counter-clockwise given angle $\phi$. An active rotation means that the vector is multiplied by the rotation matrix and this rotates the vector counterclockwise $x' = Rx$. A passive rotation means that the coordinate system is rotated and therefore the vector is also rotated: $x' = R^{-1}x$. Later on we will need rotation matrices for identification of structural shocks!

4. progs/matlab/matrixAlgebraCholesky.m

```
% ------------------------------------------------------------------------
% Decompose a covariance matrix using the Cholesky decomposition, i.e.
% SIGu = W*SIGe*W'
% ------------------------------------------------------------------------
% Willi Mutschler, November 16, 2021
% willi@mutschler.eu
% ------------------------------------------------------------------------
clearvars; clc; close all;

SIGu = [2.25 0 0; 0 1 0.5; 0 0.5 0.74];
P = chol(SIGu,'lower');
% Note that P = W*SIGe^(1/2)
SIGe_sqrt = diag(P);
SIGe = diag(SIGe_sqrt.^2);
```

```
% Find W which is solution to equation W*SIGe^(1/2) = P
% — A\B (mldivide) solves A*x = B
% — A/B (mrdivide) solves x*B = A <— we want this to get W
W = P/diag(SIGe_sqrt);
isequal(W*SIGe*W',SIGu)
```

$$\underbrace{\begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0.5 & 1\end{pmatrix}}_{W}\underbrace{\begin{pmatrix}2.25 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 0.49\end{pmatrix}}_{\Sigma_\varepsilon}\underbrace{\begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0.5\\ 0 & 0 & 1\end{pmatrix}}_{W'} = \underbrace{\begin{pmatrix}2.25 & 0 & 0\\ 0 & 1 & 0.5\\ 0 & 0.5 & 0.74\end{pmatrix}}_{\Sigma}$$

5. Solving this equation can be done either analytically or using an algorithm:

   a) Analytically:

$$\begin{aligned} vec(\Sigma_y) &= vec(A\Sigma_y A') + vec(\Sigma_u) = (A\otimes A)vec(\Sigma_y) + vec(\Sigma_u)\\ (I - A\otimes A)vec(\Sigma_y) &= vec(\Sigma_u)\\ vec(\Sigma_y) &= (I - A\otimes A)^{-1}vec(\Sigma_u)\end{aligned}$$

   b) Doubling algorithm:

progs/matlab/dlyapdoubling.m

```
% ========================================================================
% dlyapdoubling.m
% ========================================================================

function SIGy = dlyapdoubling(A,SIGu)
% ========================================================================
% Solves the Lyapunov equation SIGy = A*SIGy*A' + SIGu using the doubling
% algorithm
% ========================================================================
% SIGy = dlyapdoubling(A,SIGu)
% ------------------------------------------------------------------------
% INPUT
%   — A     : square matrix [n x n] (usually autoregressive or state space
    matrix)
%   — SIGu  : square matrix (n x n] (usually covariance matrix)
% ------------------------------------------------------------------------
% OUTPUT
%       — SIGy: square matrix (usually covariance matrix) [n x n] that solves
%       the Lyapunov equation
% ========================================================================
% Willi Mutschler, October 30, 2021
% willi@mutschler.eu
% ========================================================================

max_iter   = 500;
A_old      = A;
SIGu_old   = SIGu;
SIGy_old   = eye(size(A));
difference = .1;
index1     = 1;
```

```
tol        = 1e-25;
while (difference > tol) && (index1 < max_iter)
    SIGy       = A_old*SIGy_old*transpose(A_old) + SIGu_old;
    difference = max(abs(SIGy(:)-SIGy_old(:)));
    SIGu_old   = A_old*SIGu_old*transpose(A_old) + SIGu_old;
    A_old      = A_old*A_old;
    SIGy_old   = SIGy;
    index1     = index1 + 1;
end

end %function end
```

The basic idea of the doubling algorithm is to start at some $\Sigma_{y,0}$ and find new values for $\Sigma_{y,i+1}$ using the equation $A\Sigma_{y,i}A' + \Sigma_u$ until the difference $\Sigma_{y,i+1} - \Sigma_{y,i}$ becomes very small or a certain maximum number of iterations is reached.

The doubling algorithm, however, allows one to pass in one iteration from $\Sigma_{y,i}$ to $\Sigma_{y,2i}$ rather than $\Sigma_{y,i+1}$, provided that one updates three other matrices. There are also other (generalized) algorithms to solve such matrix Lyapunov (or Sylvester) equations.

c) Comparison:

progs/matlab/matrixAlgebraLyapunovComparison.m

```
% ------------------------------------------------------------------------
% Compares different ways to compute solution of Lyapunov equation
% SIGy = A*SIGy*A' + SIGu, i.e. analytically using the Kronecker formula
% and the doubling algorithm
% ------------------------------------------------------------------------
% Willi Mutschler, November 16, 2021
% willi@mutschler.eu
% ------------------------------------------------------------------------
clearvars; clc; close all;

%% Small example
A = [0.5,   0,   0;
     0.1, 0.1, 0.3;
     0,   0.2, 0.3];
SIGu = [2.25 0 0; 0 1 0.5; 0 0.5 0.74];

tic
vecSIGy   = (eye(size(A,1)^2)-kron(A,A)) \ SIGu(:);
SIGy_kron = reshape(vecSIGy,size(A));
toc

tic
SIGy_dlyap = dlyapdoubling(A,SIGu);
toc
fprintf('The maximum absolute difference of entries is %d\n',max(abs(
    SIGy_kron(:)-SIGy_dlyap(:))));

% re-run to see that dlyapdoubling becomes faster
% (quicker access to ram and just-in-time compilation)

%% Example with larger matrices
```

```
% Define the matrix size
n = 100;
% Loop until A has all eigenvalues inside the unit circle
while true
    SIGu = randn(n,n);
    SIGu = SIGu'*SIGu;
    Arand = rand(n, n);
    % Normalize the matrix to ensure its spectral radius (maximum absolute
        eigenvalue) is less than 1
    spectral_radius = max(abs(eig(Arand)));
    A = Arand / (spectral_radius + 1e-5); % Adding a small value to avoid
        division by zero
    % Check if all eigenvalues are inside the unit circle
    if max(abs(eig(A))) < 1
        break;
    end
end

% run comparison
tic
vecSIGy   = (eye(size(A,1)^2)-kron(A,A)) \ SIGu(:);
SIGy_kron = reshape(vecSIGy,size(A));
toc

tic
SIGy_dlyap = dlyapdoubling(A,SIGu);
toc
fprintf('The maximum absolute difference of entries is %d\n',max(abs(
    SIGy_kron(:)-SIGy_dlyap(:))));

% MATLAB's built-in functions
tic
lyap(A,SIGu);
toc
tic
dlyap(A,SIGu);
toc
tic
dlyapchol(A,chol(SIGu));
toc
```

The doubling algorithm is faster than the analytical closed-form expression based on the Kronecker product.

## 2 Solution to Understanding multivariate time series concepts

1. For the specification of multi-equation models we require a clear distinction between exogenous and endogenous variables. In economic theory this is often not clear or arbitrarily made in practice. Vectorautoregressive models do not need this distinction, they can rather be understood as a dynamic version of a simultaneous multi-equation model. This corresponds to reality, because economic variables are generated by dynamic processes which are often dependent. For example, we are able to consider correlations between $u_{1,t}$ and $u_{2,t}$, which we would not be able if we considered each equation separately. VAR models therefore provide a powerful instrument.

   Additionally, they can also deal with issues like non-stationarity (co-integration and long-term equilibria) as well as the analysis of dynamics of random shocks / impulses. Lastly, VAR models tend to have better predictive power than multi-equation models and are often used as a benchmark.

2. *Interpretation of* $E[y_t]$*:* Each component of $y_t$ has its own expectation. Therefore, $E[y_t]$ is the unconditional expectation of $y_t$ at period $t$. The expected value as a linear operator can be dragged into a vector or matrix:

$$E[y_t] = E\left[\begin{pmatrix} y_{1,t} \\ \vdots \\ y_{K,t} \end{pmatrix}\right] = \begin{pmatrix} E[y_{1,t}] \\ \vdots \\ E[y_{K,t}] \end{pmatrix}$$

   *Interpretation of* $\Gamma_y(h)$*:* In the univariate case we defined the autocovariance as the covariance of a random variable with its own lagged values. In the multivariate case we also consider covariances between different variables and different points in time. The autocovariance matrix $\Gamma_y(h) := Cov[y_t, y_{t-h}]$ summarizes this information up in a neat fashion and is therefore a powerful tool in multivariate time series analysis:

$$\begin{aligned}
&Cov[y_t, y_{t-h}] = E\left[(y_t - E[y_t])(y_{t-h} - E[y_{t-h}])'\right] \\
&= E\left[\begin{pmatrix} y_{1,t} - E[y_{1,t}] \\ \vdots \\ y_{K,t} - E[y_{K,t}] \end{pmatrix} \begin{pmatrix} y_{1,t-h} - E[y_{1,t-h}] & \cdots & y_{K,t-h} - E[y_{K,t-h}] \end{pmatrix}\right] \\
&= E\left[\begin{pmatrix} (y_{1,t} - E[y_{1,t}])(y_{1,t-h} - E[y_{1,t-h}]) & \cdots & (y_{1,t} - E[y_{1,t}])(y_{K,t-h} - E[y_{K,t-h}]) \\ \vdots & \ddots & \vdots \\ (y_{K,t} - E[y_{K,t}])(y_{1,t-h} - E[y_{1,t-h}]) & \cdots & (y_{K,t} - E[y_{K,t}])(y_{K,t-h} - E[y_{K,t-h}]) \end{pmatrix}\right] \\
&= \begin{pmatrix} E[(y_{1,t} - E[y_{1,t}])(y_{1,t-h} - E[y_{1,t-h}])] & \cdots & E[(y_{1,t} - E[y_{1,t}])(y_{K,t-h} - E[y_{K,t-h}])] \\ \vdots & \ddots & \vdots \\ E[(y_{K,t} - E[y_{K,t}])(y_{1,t-h} - E[y_{1,t-h}])] & \cdots & E[(y_{K,t} - E[y_{K,t}])(y_{K,t-h} - E[y_{K,t-h}])] \end{pmatrix} \\
&= \begin{pmatrix} Cov[y_{1,t}, y_{1,t-h}] & \cdots & Cov[y_{1,t}, y_{K,t-h}] \\ \vdots & \ddots & \vdots \\ Cov[y_{K,t}, y_{1,t-h}] & \cdots & Cov[y_{K,t}, y_{K,t-h}] \end{pmatrix}
\end{aligned}$$

   On the diagonals we have the autocovariance of each variable, on the off-diagonals we have the covariances between the different variables.

3. Basically it is the same: A stochastic process is called weakly stationary (or covariance stationary), if for each period in time it has the same expectation independent of time and the autocovariance between any two points in time is only dependent on the distance of these two points. The difference is that in the multivariate case we also consider the autocovariances between different variables and require these to be only dependent on the distance in time, but not on time itself.

`Definition`: The K-dimensional stochastic process $y_t$ is called covariance stationary, if for all $h, t, \tau \in \mathbb{Z}$:

$$E[y_t] = E[y_\tau] \tag{1}$$
$$Cov(y_t, y_{t-h}) = Cov(y_{t+\tau}, y_{t-h+\tau}) \tag{2}$$

4. We will use the **method of matching coefficients** to transform the VAR(1) model into a VMA($\infty$) representation.

$$\begin{aligned} y_t &= Ay_{t-1} + u_t \\ \underbrace{(I_3 - AL)}_{A(L)} y_t &= u_t \\ A(L)y_t &= u_t \\ A(L)^{-1}A(L)y_t &= A(L)^{-1}u_t \\ y_t &= A(L)^{-1}u_t = \nu + \Phi(L)u_t \\ &\Rightarrow \nu = 0, \Phi(L) = A(L)^{-1} \end{aligned}$$

In the method of matching coefficient we compare the coefficient matrices multiplied to each power of $L$. That is, the expression on the left hand side has to match the expression on the right hand side. In our case:

$$\begin{aligned} \Phi(L) &= A(L)^{-1} \\ A(L)\Phi(L) &= I_3 \\ (I_3 - AL)\left(\sum_{i=0}^{\infty} \Phi_i L^i\right) &= I_3 \end{aligned}$$

$$\begin{aligned} &\Phi_0 L^0 + \Phi_1 L^1 + \Phi_2 L^2 + \dots \\ &\quad - A\phi_0 L^1 - A\phi_1 L^2 - \dots = I_3 L^0 \end{aligned}$$

$$\begin{aligned} L^0 : \Phi_0 &= I_3 \Rightarrow \Phi_0 = I_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \\ L^1 : \Phi_1 - A\Phi_0 &= 0 \Rightarrow \Phi_1 = A\Phi_0 = A = \begin{pmatrix} 0.5 & 0 & 0 \\ 0.1 & 0.1 & 0.3 \\ 0 & 0.2 & 0.3 \end{pmatrix} \\ L^2 : \Phi_2 - A\Phi_1 &= 0 \Rightarrow \Phi_2 = A\Phi_1 = A^2 = \begin{pmatrix} 0.25 & 0 & 0 \\ 0.06 & 0.07 & 0.12 \\ 0.02 & 0.08 & 0.15 \end{pmatrix} \\ &\vdots \\ &\text{In general} : \Phi_s = A^s \end{aligned}$$

## 3 Solution to Dimensions and VAR(1) representation

1. $\nu$ snd $u_t$ are both K-dimensional vectors: $\nu, u_t \in \mathbb{R}^{K\times 1}$. $A_i$ is a $K \times K$ matrix.

2. We must have $\Gamma_u(h) = Cov(u_t, u_{t-h}) = 0_{K\times K}$ for any $h \neq 0$, that is all autocovariances are zero (except for $h = 0$ which is the covariance matrix). Importantly: we do not need a distributional assumption!

3. General: $K$ constants + $K^2 \cdot p$ autoregressive coefficients + $K(K+1)/2$ covariance terms (due to symmetry). Here: $4 + 4^2 \cdot 2 + 4 \cdot (4+1)/2 = 46$. That's a lot! Therefore we will try to restrict some parameters (e.g. set equal to zero or by using Bayesian priors) or consider only small VAR systems, e.g. $K = 3$ or $p = 1$, etc.

4. VAR(3): $y_t = \nu + A_1 y_{t-1} + A_2 y_{t-2} + A_3 y_{t-3} + u_t$. Idea: Stack $y_t, y_{t-1}$ and $y_{t-2}$ into a vector and note that $y_{t-1} = y_{t-1}$ and $y_{t-2} = y_{t-2}$. That is

$$\underbrace{\begin{bmatrix} y_t \\ y_{t-1} \\ y_{t-2} \end{bmatrix}}_{\widetilde{y}_t} = \underbrace{\begin{bmatrix} \nu \\ 0 \\ 0 \end{bmatrix}}_{\widetilde{\nu}_t} + \underbrace{\begin{bmatrix} A_1 & A_2 & A_3 \\ I & 0 & 0 \\ 0 & I & 0 \end{bmatrix}}_{\widetilde{A}} \underbrace{\begin{bmatrix} y_{t-1} \\ y_{t-2} \\ y_{t-3} \end{bmatrix}}_{\widetilde{y}_{t-1}} + \underbrace{\begin{bmatrix} u_t \\ 0 \\ 0 \end{bmatrix}}_{\widetilde{u}_t}$$

   where $I$ is the $K$-dimensional identity matrix and 0 the $K$-dimensional zero matrix. Therefore: $\widetilde{y}_t = \widetilde{A}\widetilde{y}_{t-1} + \widetilde{u}_t$. This is called the Companion Form. It is particularly useful, when checking the stability and covariance-stationarity properties of VAR(p) processes as we can simply compute the Eigenvalues of $\widetilde{A}$ and check whether all of them are inside the unit circle, i.e. between $-1$ and 1. No need to find the roots of the general Lag-polynomials.

5. progs/matlab/companionForm.m

```
function A = companionForm(Coefs,p)
% A = companionForm(Coefs,p)
% ------------------------------------------------------------------------
% Computes matrix A of companion form of a VAR(p) model with constant
%
% That is, any VAR(p) model with constant
%   y_t = c + A_1*y_{t-1} + ... + A_p*y_{t-p} + u_t
% can be represented as
%   Y_t = C + A*Y_{t-1} + U_t
% where
%   Y_t = [y_t;y_{t-1};...;y_{t-p+1}]
%   C   = [c;0;...;0]
%   A   = [A_1 A_2 ... A_{p-1} A_p;
%          I_K 0_K ... 0_K     0_K;
%          0_K I_K ... 0_K     0_K:
%          ... ... ... ...     ...;
%          O_K 0_K ... I_K     0_K]
%   U_t = [u_t; 0; ...;0]
% ------------------------------------------------------------------------
% INPUT
%   - Coefs: matrix of coefficients Coef = [nu A_1 A_2 ... A_p]
%   - p:     number of lags
% ------------------------------------------------------------------------
% OUTPUT
%   - A: companion matrix
% ------------------------------------------------------------------------
```

```
% Willi Mutschler, December 6, 2021
% willi@mutschler.eu
%--------------------------------------------------------------------------

K = size(Coefs,1); %number of variables
const = size(Coefs,2) - K*p; %if any deterministic terms are upfront

A = [Coefs(:,const+1:end);
     eye(K*(p-1)) zeros(K*(p-1),K)];

end % function end
```