CUEDSID 1.0

System Identification Toolbox

User's Guide

Eric C. Kerrigan, Huixin Chen and Jan M. Maciejowski
Cambridge University Engineering Department
Cambridge CB2 1PZ England
`jmm@eng.cam.ac.uk`

Software for subspace identification of linear and bilinear dynamic systems
and for prediction-error identification using balanced parametrizations.
For use with *Matlab*.

25 June 2002

This software has been developed to work with:

- MATLAB Version 6.0 or higher,
- MATLAB System Identification Toolbox Version 5.0, and
- MATLAB Control System Toolbox Version 5.0.

Acknowledgement

The research leading to this software was supported by the
Engineering and Physical Sciences Research Council of the UK
under research grant GR/M08332/01
Integration of subspace and parametric system identification for linear and bilinear systems.

Contents

Chapter 1

Getting Started

The Cambridge University Engineering Department System Identification (CUEDSID) Toolbox for MATLAB can be downloaded from:

```
http://www-control.eng.cam.ac.uk/jmm/cuedsid
```

1.1 Installation

The *CUEDSID Toolbox* requires the following software to be pre-installed:

- MATLAB Version 6.0 or higher,
- MATLAB System Identification Toolbox Version 5.0, and
- MATLAB Control System Toolbox Version 5.0.

NB: It is assumed throughout this guide that the user is already acquainted with the use of MATLAB *and its System Identification and Control System Toolboxes. The* CUEDSID Toolbox *can be thought of as an add-on to the* MATLAB System Identification Toolbox.

Once the files have been downloaded, they should be decompressed into the user's MATLAB directory (e.g. `mymatlab`). The following lines should be added to the `startup.m` file in order to add the files to the MATLAB search path:

```
addpath mymatlab/cuedsid
addpath mymatlab/cuedsid/balpemsid
addpath mymatlab/cuedsid/bilinidsid
addpath mymatlab/cuedsid/linsid
```

Once MATLAB has been started, the on-line help can be invoked by typing:

```
>> help cuedsid
```

1.2 Main Functions

The three main functions provided with the *CUEDSID Toolbox* are:

`subid3b`: Subspace identification of linear systems — see Chapter 2.

`balpem`: System identification using balanced parameterizations — see Chapter 3.

`bilinid`: Subspace identification of bilinear systems — see Chapter 4.

1.3 Auxiliary Functions

A number of auxiliary functions are also provided with the *CUEDSID Toolbox*:

`bilin`	Create a discrete-time, bilinear, state-space system
`bilin/compare`	Compare simulated data with measured data
`bilin/isstable`	Determine whether a given bilinear system is stable
`bilin/pe`	Compute prediction errors associated with a data set
`bilin/sim`	Simulate a given bilinear system (with noise)
`blochank`	Assemble a block Hankel matrix from a given block matrix
`bloctoep`	Assemble a block Toeplitz matrix from two given block matrices
`coorproj`	Orthogonal projection onto a complement subspace
`isbalanced`	Determines whether a state-space system is balanced
`ismpbalanced`	Determines whether a system is minimum-phase balanced
`khatri`	Compute the Khatri-Rao product of two matrices
`mpbal`	Computes a minimum-phase balanced realization
`obmat`	Construct the observability matrix with a given index
`orthproj`	Orthogonal projection onto a subspace
`soltritoep`	Solve for a lower-triangular, block Toeplitz matrix

The first five functions above are intended for use in conjunction with `bilinid` and the functions `isbalanced`, `ismpbalanced` and `mpbal` are intended for use in conjunction with `balpem`. The remaining functions are used by `subid3b`, `balpem` and `bilinid`, but can also be used as stand-alone functions.

The reader is referred to the following chapters and the function reference at the end of the guide for more details.

Chapter 2

Subspace Identification of Linear Systems

Most methods of system identification rely on iterative, nonlinear optimisation to fit parameters in a pre-selected model structure, so as to best fit the observed data [Lj99]. *Subspace methods* are an alternative class of identification methods which are 'one-shot' rather than iterative, and rely on linear algebra rather than on optimisation. They are very easy to use, and generally give very good results. They can also be used as sources of initial models which can be refined further using the optimisation approach, if required.

Standard subspace algorithms, such as the one implemented in the MATLAB *System Identification Toolbox* function `n4sid`, split the available input-output data into two blocks, which can be thought of as the *past* and the *future*. The basic versions of these standard subspace methods suffer from systematic errors (bias) — unless the measured input is white — which reduce as the so-called *block size* parameter increases These errors can be avoided by algorithms of additional complexity (as implemented in `n4sid`, for example) [VODM96].

An alternative approach to reducing these systematic errors is presented in [CM98] and has been implemented in the *CUEDSID Toolbox* function `subid3b`. This algorithm splits the data into three blocks, which we denote the *past*, the *current* and the *future* blocks. The result of using this approach is that the bias is reduced, even when the noise is not white.

2.1 Overview

Given a set of input-output data, `subid3b` aims to identify a discrete-time, state-space model in innovation form:

$$\begin{aligned} x(t+T_s) &= Ax(t) + Bu(t) + Ke(t), \quad x(0) = x_0 \\ y(t) &= Cx(t) + Du(t) + e(t) \end{aligned}$$

where $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{p\times n}$, $D \in \mathbb{R}^{p\times m}$ and the Kalman gain $K \in \mathbb{R}^{n\times p}$; x denotes the state, u the input signal, y the output signal and e the process noise. T_s is the sample time and x_0 is the initial state.

The function `subid3b` has been written to be used in conjunction with the MATLAB *System*

Identification Toolbox. Input-output data has to be passed to `subid3b` as an `iddata` object and the estimated model is returned from `subid3b` as an `idss` object. The GUI, `ident`, and other MATLAB *System Identification Toolbox* functions, such as `idmodel/compare` and `idmodel/resid`, can be used to process data for identification and validation of the model.

When calling `subid3b` to identify a model, the user can choose whether or not to override the automatic selection of the following:

- System order n,
- Block size k,
- Whether D is to be estimated or set to zero, and
- The specific variant of the three-block algorithm, which can be one of the following approaches:
  - Markov parameter,
  - Shift invariance, or
  - State sequence.

For more details, the user is referred to the function reference and [CM98].

2.2 Example

The function `subid3b` is very easy to use. A short example will illustrate the typical steps involved in identifying and verifying a model using `subid3b`.

The first step is to process some data for identification and validation. This can be done using the MATLAB *System Identification Toolbox*. The following command will load some processed data into the MATLAB workspace:

```
>> load example3block
>> who

Your variables are:

data  sys

>> data
Data set with 300 samples.
Sampling interval: 1

Outputs      Unit (if specified)
   y1
   y2

Inputs       Unit (if specified)
```

```
   u1
   u2
   u3

>> sys
State-space model:  x(t+Ts) = A x(t) + B u(t) + K e(t)
                       y(t) = C x(t) + D u(t) + e(t)

A =
                       x1           x2           x3
           x1    -0.40759      0.45403    -0.045775
           x2   -0.072194     0.025234     -0.73298
           x3    -0.45058     -0.57994      0.37359


B =
                       u1           u2           u3
           x1      2.1157      -1.2466      -1.9128
           x2           0            0            0
           x3      1.0462      0.53008       1.5283


C =
                       x1           x2           x3
           y1      1.3543       1.8706     -0.45374
           y2           0       0.8012      0.94957


D =
                       u1           u2           u3
           y1      1.1574     0.088764      0.98312
           y2     -1.1595            0     -0.87049


K =
                       y1           y2
           x1           0            0
           x2           0            0
           x3           0            0


x(0) =

           x1           0
           x2           0
           x3           0
```

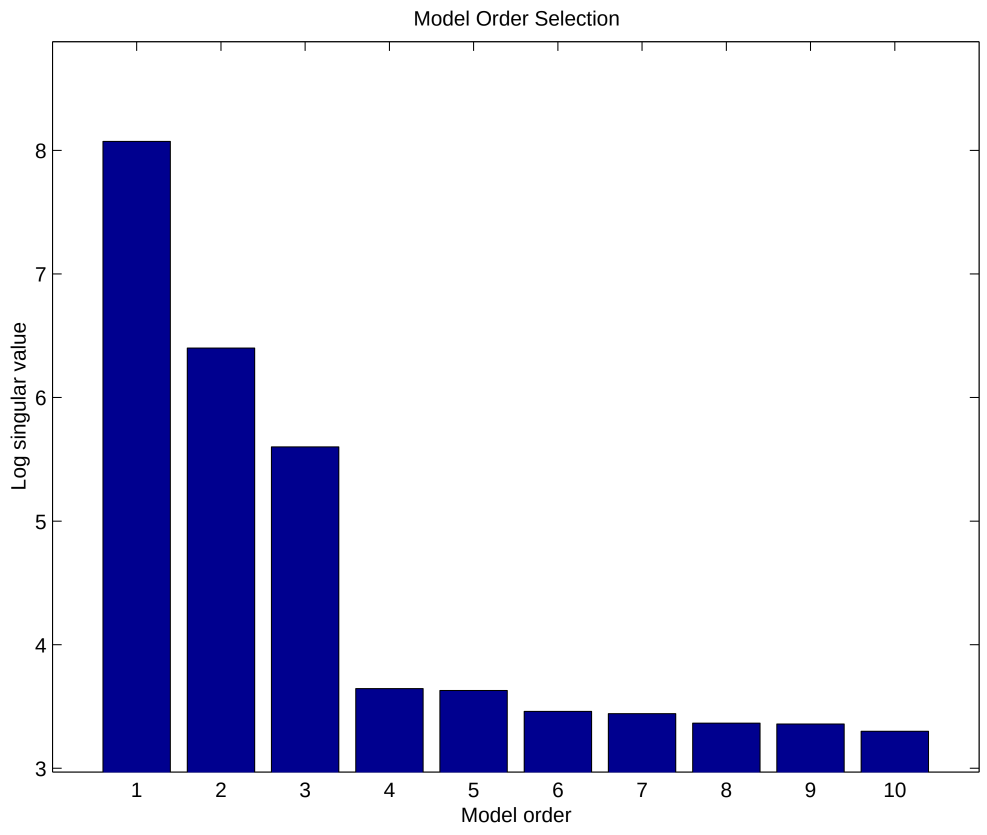

Figure 2.1: Plot of singular values resulting from SVD decomposition in three-block algorithm

```
This model was not estimated from data.
Sampling interval: 1
```

The variable `data` is an `iddata` object and `sys` is the actual system that was that was used to generate the data (after adding some noise). `sys` is an `idss` object.

The following commands extract different parts of `data` for identification and validation:

```
>> identdata = data(1:250);
>> valdata = data(251:300);
```

The following call to `subid3b` uses the shift invariance approach to help estimate the order of the system. A logarithmic plot of the singular values that resulted from the SVD decomposition is displayed and the user is prompted to select the system order. The block size k is not specified and hence chosen by the algorithm. As can be seen in Figure 2.1, the first three singular values are a lot larger than the rest, hence one can accurately estimate that $n = 3$.

```
>> Msi = subid3b(identdata,[1:10])
Block size k = 20.
Please select model order: 3
State-space model:  x(t+Ts) = A x(t) + B u(t) + K e(t)
```

```
                     y(t) = C x(t) + D u(t) + e(t)

A =
                        x1           x2           x3
            x1     0.95181    -0.084857   0.00053211
            x2   0.0021275     -0.61431       0.4012
            x3    0.057737     -0.33869     -0.34948


B =
                        u1           u2           u3
            x1    0.025562     0.071337      0.16243
            x2    -0.17306     0.050237     0.042829
            x3     0.02156     -0.05271     -0.10584


C =
                        x1           x2           x3
            y1     -17.233      -16.594      -2.3172
            y2      3.1321      -6.7352      -11.604


D =
                        u1           u2           u3
            y1       1.158     0.097347      0.98285
            y2     -1.1522    -0.017377     -0.87642


K =
                        y1           y2
            x1 -0.00030967   0.00023064
            x2   0.0014415   -0.0015581
            x3   0.0013428   -0.0019277


x(0) =

            x1           0
            x2           0
            x3           0

Estimated using SUBID3B - Shift invariance approach
Loss function
Sampling interval: 1
```

The next two commands estimate two additional models using the state sequence and Markov parameter approaches:

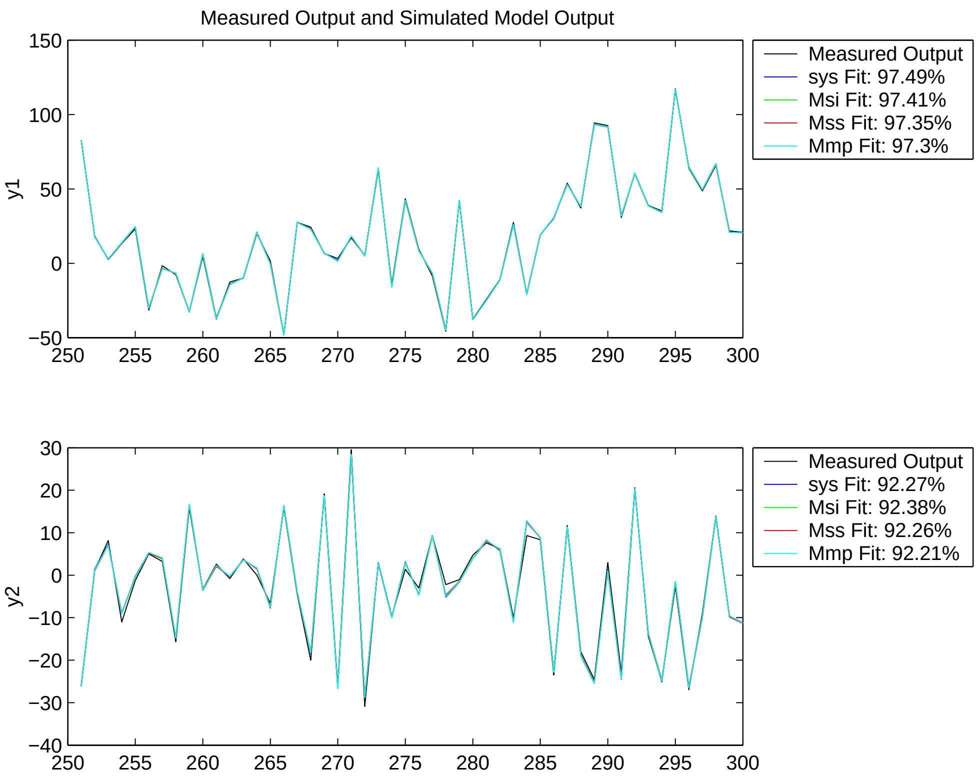

Figure 2.2: Comparison of different models obtained using the three-block algorithm

```
>> Mss = subid3b(identdata,3,[],[],'ss');
Block size k = 20.
>> Mmp = subid3b(identdata,3,[],[],'mp');
Block size k = 20.
```

Finally, each of the three models are compared against one another using the validation data. Figure 2.2 results from making the following call to the MATLAB *System Identification Toolbox* function `idmodel/compare`:

```
>> compare(valdata,sys,Msi,Mss,Mmp)
```

As can be seen in Figure 2.2, all three models fit the data well. The models appears to predict the data as well as the original system `sys` that was used to generate the data.

Chapter 3

System Identification of Linear Systems Using Balanced Parameterizations

As briefly mentioned at the beginning of Section 2, most methods of system identification rely on iterative, nonlinear optimisation to fit parameters in a pre-selected model structure, so as to best fit the observed data. Such optimisation has to be performed subject to certain constraints, in order to avoid undesirable models such as unstable ones, and to keep the search process well-conditioned. (The parameter space being searched is of higher dimension than the 'behaviour space'.)

A method of performing such an optimisation without constraints, by exploiting an explicit parametrisation of the lower-dimensional 'behaviour space', is presented in [CM97]. The approach is based on the so-called *balanced parameterization*, initially developed by Ober [Ob87], and has some advantages over the use of the better-known canonical forms, such as the *observable form*.

For example, a state-space system that is close to being non-minimal gives rise to an ill-conditioned parameter estimation problem; perturbations of the parameter estimates in certain directions (in the parameter space) have very little effect on the input-output behaviour of the estimated model. The consequence of this is that the accuracy of parameter estimation is low. In a sense, because the balanced realization of a given system can be thought of as the one that is 'furthest away' from non-minimality, the use of a balanced parameterization gives an estimation problem that is as well-conditioned as possible.

Balanced parametrizations of several classes of linear systems have been developed [Ob91]. These allow the parameters to vary almost without constraints (typically they are required to be positive), without leaving the class of system being parametrised. Parametrizations of the classes of *stable* systems and of *minimum-phase* systems are supported by the *CUEDSID Toolbox*.

3.1 Overview

The function `balpem`, that implements the prediction-error identification method of [CM97] using balanced parameterizations, has been designed around the MATLAB *System Identification Toolbox* function `pem`. To be more precise, `balpem` uses the `idgrey` class to set up the balanced parameterizations described in [CM97, Sect. IV] and [CM97, Sect. V.A], and then calls `pem` to improve on the initial estimate of the model.

The function `balpem` is easy to use; the minimum number of arguments required in `balpem` is the sequence of input-output data, given as an `iddata` object, and an initial estimate of the model, given as an `idss` object. The final estimate is returned from `balpem` as an `idss` object in innovation form:

$$\begin{aligned} x(t+T_s) &= Ax(t) + Bu(t) + Ke(t), \quad x(0) = x_0 \\ y(t) &= Cx(t) + Du(t) + e(t) \end{aligned}$$

where $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{p\times n}$, $D \in \mathbb{R}^{p\times m}$ and the Kalman gain $K \in \mathbb{R}^{n\times p}$; x denotes the state, u the input signal, y the output signal and e the process noise. T_s is the sample time and x_0 is the initial state.

Provided the initial estimate is stable[1], the final estimate is stable and balanced in one of the following two ways, as specified by the user:

- (A, B, C, D) is balanced in the usual sense of the controllability gramian of (A, B, C, D) being diagonal and equal to the observability gramian of (A, B, C, D).

  As a default, D is estimated without any constraints. Alternatively, the user can choose to set D to zero.

  The Kalman gain K is set to zero[2] (i.e. it is assumed that the only source of noise is measurement noise).

- (A, K, C, I) is minimum-phase and balanced in the sense that the controllability gramian of (A, K, C, I) is diagonal and equal to the observability gramian of the *inverse* of (A, K, C, I) (i.e. the observability gramian of $(A - KC, K, -C, I)$).

  As a default, B and D are estimated without any constraints. Alternatively, the user can choose to set B and/or D to zero.

Three auxiliary functions have also been provided for use in conjunction with `balpem`:

`isbalanced`: Tests whether a model is balanced in the usual sense.

`ismpbalanced`: Tests whether a model is balanced in the minimum-phase sense.

`mpbal`: Computes a minimum-phase balanced realization of a given *continuous-time*, state-space model.

The reader is referred to the function reference for details on using these functions.

[1]The function `balpem` also assumes that the initial estimate is minimal in some sense and that the Hankel singular values are distinct (see [CM97] for details and other, minor technical assumptions). Fortunately, most subspace algorithms, such as `subid3b` and `n4sid` usually provide good, initial estimates that satisfy the assumptions made by `balpem`.

[2]This can be overridden by the user, if desired, by setting the property `'DisturbanceModel'` to `'Estimate'`.

3.2 Example

A short example will illustrate the typical steps involved in identifying and verifying a model using `subid3b`. The following commands load some data and an initial estimate into the Matlab workspace:

```
>> data
Data set with 300 samples.
Sampling interval: 1

Outputs      Unit (if specified)
   y1
   y2

Inputs       Unit (if specified)
   u1
   u2
   u3

>> sys
State-space model:  x(t+Ts) = A x(t) + B u(t) + K e(t)
                       y(t) = C x(t) + D u(t) + e(t)

A =
                       x1            x2            x3
           x1     -0.40759       0.45403     -0.045775
           x2    -0.072194      0.025234      -0.73298
           x3     -0.45058      -0.57994       0.37359


B =
                       u1            u2            u3
           x1       2.1157       -1.2466       -1.9128
           x2            0             0             0
           x3       1.0462       0.53008        1.5283


C =
                       x1            x2            x3
           y1       1.3543        1.8706      -0.45374
           y2            0        0.8012       0.94957


D =
                       u1            u2            u3
           y1       1.1574      0.088764       0.98312
           y2      -1.1595             0      -0.87049
```

```
K =
                     y1            y2
           x1         0             0
           x2         0             0
           x3         0             0


x(0) =

           x1         0
           x2         0
           x3         0

This model was not estimated from data.
Sampling interval: 1
```

The variable `data` is an `iddata` object, `sys` is the actual system that was that was used to generate the data (after adding some noise) and `mi` is an initial estimate of the system, computed using `subid3b`; `sys` and `mi` are `idss` objects.

The following commands extract different parts of `data` for identification and validation:

```
>> identdata = data(1:250);
>> valdata = data(251:300);
```

Since the current `mi` is already a good estimate of `sys` (see Section 2.2), in order to make things more interesting `mi`, the D matrix of `mi` is set to zero:

```
>> mi.d=mi.d*0
State-space model:  x(t+Ts) = A x(t) + B u(t) + K e(t)
                       y(t) = C x(t) + D u(t) + e(t)

A =
                     x1            x2            x3
           x1     0.95181     -0.084857    0.00053211
           x2   0.0021275      -0.61431        0.4012
           x3    0.057737      -0.33869      -0.34948


B =
                     u1            u2            u3
           x1    0.025562      0.071337       0.16243
           x2    -0.17306      0.050237      0.042829
           x3     0.02156      -0.05271      -0.10584
```

```
C =
                     x1          x2          x3
           y1   -17.233     -16.594     -2.3172
           y2    3.1321     -6.7352     -11.604


D =
                     u1          u2          u3
           y1         0           0           0
           y2         0           0           0


K =
                     y1          y2
           x1 -0.00030967  0.00023064
           x2   0.0014415  -0.0015581
           x3   0.0013428  -0.0019277


x(0) =

           x1         0
           x2         0
           x3         0

Estimated using SUBID3B - Shift invariance approach
Loss function
Sampling interval: 1
```

Given this new `mi` as an initial estimate of `sys`, the following command calls `balpem` to obtain an improved, stable and balanced estimate of `sys`:

```
>> msb = balpem(identdata,mi)
Warning: Mi.K is not allowed to be non-zero if ALG is 'sb'. Setting Mi.K = 0.
State-space model:  x(t+Ts) = A x(t) + B u(t) + K e(t)
                       y(t) = C x(t) + D u(t) + e(t)

A =
                     x1          x2          x3
           x1   0.95009   -0.088849   0.0010244
           x2 -0.029647     -0.6211     0.38655
           x3  0.058536    -0.35747    -0.33828


B =
                     u1          u2          u3
           x1  -0.24568    -0.70906     -1.6163
           x2       1.7186     -0.48791     -0.41359
           x3     -0.16203      0.51618       1.0584


C =
                        x1           x2           x3
           y1       1.7646       1.6646      0.25344
           y2     -0.30981      0.64502       1.1528


D =
                        u1           u2           u3
           y1       1.1542      0.09694      0.99212
           y2       -1.156   -0.0057658     -0.87074


K =
                        y1           y2
           x1            0            0
           x2            0            0
           x3            0            0


x(0) =

           x1            0
           x2            0
           x3            0

Estimated using BALPEM from data set data
Loss function 1.0554 and FPE 1.21917
Sampling interval: 1
```

The following command verifies that the new estimate is indeed balanced:

```
>> isbalanced(msb)
System is balanced; norm(Wc-Wo) = 3.60e-14.
```

Given the same `mi` as an initial estimate of `sys`, the following command calls `balpem` to obtain an improved, stable and *minimum-phase* balanced estimate of `sys`:

```
>> mmp = balpem(identdata,mi,[],'mp')
State-space model:  x(t+Ts) = A x(t) + B u(t) + K e(t)
                       y(t) = C x(t) + D u(t) + e(t)


A =
                        x1           x2           x3
```

```
           x1      0.93912   -0.046301    0.021775
           x2     -0.28714    -0.31407     0.33817
           x3     0.011313    -0.41523    -0.64119


B =
                       u1          u2          u3
           x1      -7.4617     -14.014     -32.876
           x2      -11.797      4.5857      5.4178
           x3       21.166     -11.686     -17.272


C =
                       x1          x2          x3
           y1      0.05171    -0.15319    0.043295
           y2    -0.041494    -0.15013   -0.052353


D =
                       u1          u2          u3
           y1      0.23033    0.016489     0.19553
           y2     -0.24727  -0.0034688    -0.18705


K =
                       y1          y2
           x1     0.039789   -0.082934
           x2      0.10217    -0.17631
           x3     0.015062    0.026313


x(0) =

           x1            0
           x2            0
           x3            0

Estimated using BALPEM from data set data
Loss function 111.718 and FPE 135.445
Sampling interval: 1
```

The following command verifies that (A, K, C, I) of the new estimate `mmp` is indeed minimum-phase balanced:

```
>> ismpbalanced(ss(mmp.a,mmp.k,mmp.c,eye(2),1))
System is minimum-phase balanced; norm(Wc(SYS)-Wo(inv(SYS))) = 1.24e-16.
```

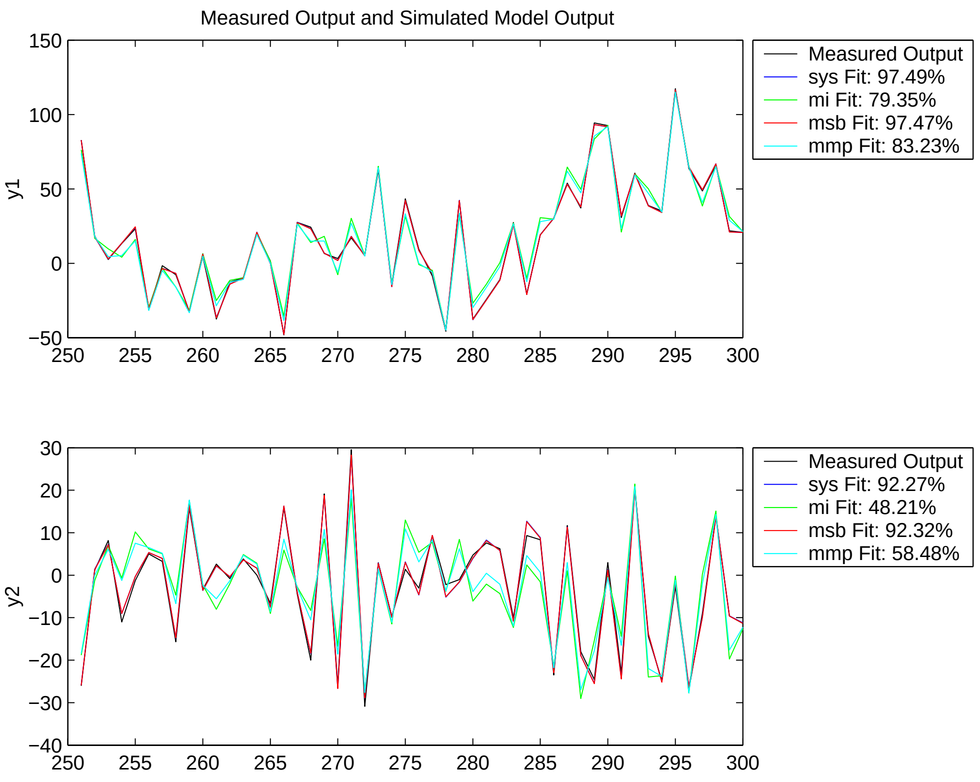

Figure 3.1: Comparison of the initial estimate and the new estimate obtained using `balpem`

Finally, the initial and new estimates are compared against one another using the validation data. Figure 3.1 results from making the following call to the MATLAB *System Identification Toolbox* function `idmodel/compare`:

```
>> compare(valdata,sys,mi,msb,mmp)
```

As can be seen in Figure 3.1, the initial model `mi` did not do very well in matching the data, whereas the new models `msb` and `mmp` are better estimates.

On further investigation, it is possible to improve a little on the minimum-phase balanced estimate `mmp` by increasing `'MaxIter'`.

```
>> mmp.estimationinfo

ans =

              Status: 'Estimated model (PEM)'
              Method: 'BALPEM'
             LossFcn: 111.7176
                 FPE: 135.4453
            DataName: 'data'
          DataLength: 250
              DataTs: 1
```

```
    DataInterSample: {3x1 cell}
            WhyStop: 'Maxiter reached'
         UpdateNorm: 113.2507
    LastImprovement: '0.069766%'
         Iterations: 20
       InitialState: 'Model'

>> mmp = balpem(identdata,mi,[],'mp','MaxIter',50);
>> mmp.estimationinfo

ans =

             Status: 'Estimated model (PEM)'
             Method: 'BALPEM'
            LossFcn: 82.9515
                FPE: 100.5695
           DataName: 'data'
         DataLength: 250
             DataTs: 1
    DataInterSample: {3x1 cell}
            WhyStop: 'Maxiter reached'
         UpdateNorm: 121.2604
    LastImprovement: '0.083681%'
         Iterations: 50
       InitialState: 'Model'

>> msb.estimationinfo

ans =

             Status: 'Estimated model (PEM)'
             Method: 'BALPEM'
            LossFcn: 1.0554
                FPE: 1.2192
           DataName: 'data'
         DataLength: 250
             DataTs: 1
    DataInterSample: {3x1 cell}
            WhyStop: 'Near (local) minimum, (norm(g)<tol).'
         UpdateNorm: 1.8941e-04
    LastImprovement: '0.00018941%'
         Iterations: 3
       InitialState: 'Model'

>> [yh,fit] = compare(valdata,sys,mi,msb,mmp); fit

fit(:,:,1) =
   97.4899   79.3547   97.4659   85.6413


fit(:,:,2) =

   92.2662   48.2088   92.3172   64.7854
```

It is interesting to note that, as the above analysis shows, `msb` results in a much better estimate than `mmp` after fewer iterations. As a matter of fact, `msb` fits the data as well as the original system `sys` that was used to generate it, whereas `mmp` can still be improved upon.

Chapter 4

Subspace Identification of Bilinear Systems

Most commonly the models obtained by system identification allow only linear relationships between the inputs and outputs. One of the objectives of this toolbox is to allow the identification of discrete-time *bilinear* models of the form

$$\begin{aligned} x(t+T_s) &= Ax(t) + N(u(t) \otimes x(t)) + Bu(t) + w(t), \quad x(0) = x_0 \\ y(t) &= Cx(t) + Du(t) + v(t) \end{aligned}$$

in which $u(t)$ and $y(t)$ are vectors of time-indexed observed input and output data, $x(t)$ is a time-indexed (unobserved) state vector, $w(t)$ and $v(t)$ are unobserved random processes, and A, B, C, D and N are matrices of suitable (but initially unknown) dimensions. The term $N(u(t) \otimes x(t))$, where $\otimes$ is the Kronecker product operator, is bilinear; if this term is absent then the model is linear.

Continuous-time bilinear models are important in process control (flow $x \times$ concentration u), aerodynamics (speed $x \times$ surface deflection u), and other applications. Discrete-time equivalents of continuous-time bilinear models are no longer exactly bilinear, but for small sampling times they remain approximately bilinear. Also discrete-time bilinear models can be regarded as a useful enlargement of the model class from linear models, even if there is no physical basis for expecting a bilinear structure. Furthermore, certain bilinear models can be regarded as examples of the increasingly important class of piecewise-linear models [Ve02].

The *CUEDSID Toolbox* provides `bilinid`, a subspace algorithm for the identification of discrete-time bilinear systems, analogous to the existing methods for linear systems — in particular, the ideas underlying the work reported in [CM98] (and implemented in `subid3b`) are exploited for this purpose.

Most subspace algorithms split the data into two blocks, which are conventionally labelled *past* and *future*. In [CM99] the data is split into three blocks when performing *deterministic* identification, namely when it is assumes that the noise terms $w(t)$ and $v(t)$ are absent. The third block is labelled *current*, and it allows one to estimate part of the system's input-output behaviour. (This idea is inherited from [CM98]; for linear systems a finite sequence of Markov parameters would be estimated in this way). This estimated behaviour is then used in a second step to estimate the state dimension of the system being identified, and two consecutive state sequences. In a third step these state sequences are used to estimate the matrices of the bilinear model.

In the *stochastic* case, when the noises $w(t)$ and $v(t)$ are assumed to be present, a fourth block is introduced, labelled *remote future*, which allows the random effects of these noises to be averaged out, before applying the same 'three-block' strategy as for the deterministic case. (This parallels the difference between the 'deterministic' and 'stochastic' cases in subspace identification of linear systems). The function `bilinid` implements this algorithm, the details of which can be found in [CM99]. The algorithms implemented in `bilinid` were introduced in [CM00a, CM00b]. Note that the *deterministic* 3-block algorithm (for the case when the w and v terms are absent), is not implemented in Version 1.0 of the *CUEDSID Toolbox*.

Alternative subspace algorithms for bilinear systems have been published in [FDV99, VV99]. The algorithm in [FDV99] assumes that the measured input is white. The algorithm in [VV99] is a two-stage method which employs hill-climbing optimization in the second stage.

4.1 Overview

The *CUEDSID Toolbox* provides a number of easy-to-use functions that allow the user to identify and validate discrete-time, bilinear state-space models. The main function is `bilinid`, which implements the four-block algorithm described in [CM99]. A `bilin` class, which functions in a fashion similar to the `idss` and `ss` classes, has also been defined. The methods that have been overloaded for the `bilin` class, include the following:

- `compare`
- `isstable`
- `sim`
- `pe`

One can simulate a bilinear system by first creating a `bilin` object in the same fashion as one would create an `idss` or `ss` object. Once a `bilin` object has been created, one can generate input-output data and add noise to it by using the method `bilin/sim`; `bilin/sim` returns input-output data as an `iddata` object that can then be manipulated using the MATLAB *System Identification Toolbox.*

Given a set of input-output data as an `iddata` object, `bilinid` aims to identify a discrete-time, bilinear state-space model in the form:

$$\begin{aligned} x(t+T_s) &= Ax(t) + N(u(t) \otimes x(t)) + Bu(t) + w(t), \quad x(0) = x_0 \\ y(t) &= Cx(t) + Du(t) + v(t) \end{aligned}$$

where $\otimes$ is the Kronecker product operator, the matrices $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{p\times n}$, $D \in \mathbb{R}^{p\times m}$, $N := [N_1\ N_2\ \cdots N_m] \in \mathbb{R}^{n\times nm}$ and each $N_i \in \mathbb{R}^{n\times n}$. T_s is the sample time and x_0 is the initial state.

When calling `bilinid` to identify a model, the user has can choose whether or not to override the automatic selection of the following:

- System order n,
- Block size k,
- Whether D is to be estimated or set to zero, and
- The specific variant of the four-block algorithm that is used, which can be one of the following:
  - General,
  - Fast (for the case when the number of outputs $p < n$), or
  - Accurate (for the case when the number of outputs $p \geq n$).

`bilinid` returns the estimate of the bilinear system to the user as a `bilin` object. Once this `bilin` object has been obtained, one could use the methods `bilin/compare`, `bilin/isstable` and `bilin/pe` to validate the model.

Suppose that the true system, together with the data, satisfies the following assumption, which is a kind of stability condition:

$$\lambda = \max_t \ \overline{\sigma} \left(A + \sum_i u_i(t) N_i \right) < 1,$$

where $\overline{\sigma}(\cdot)$ denotes the greatest singular value of a matrix and $u_i(t)$ is the i'th element of $u(t)$. Then the systematic error (bias) inherent in `bilinid` reduces as $o(\lambda^k)$ (if $p < n$). However, the computational complexity increases exponentially with k, so in practice one is restricted to rather small values of k, and hence of n, since it is generally required that $k > n$. The condition $\lambda < 1$ is tested (for a model) by the function `bilin/isstable`.

The user should be aware that, even with small values of k, the computational complexity and memory requirements of the `bilinid` function are both high, and computation times are likely to be very large on low-performance computers.

For more details, the user is referred to the function reference and [CM99].

4.2 Example

This example will demonstrate how to create, manipulate and simulate a `bilin` object. Once some data has been generated, `bilinid` will be used to identify a bilinear model from this data. Finally, the estimated model will be validated using some of the functions supplied with the *CUEDSID Toolbox*.

The following commands create a `bilin` object called `sys`:

```
>> A = diag([0.5 0.5]);
>> B = [0 1; -1 0];
>> C = [1 0; 0 2];
>> D = [1 0; 1 1];
>> N1 = [0.6 0; 0 0.4];
>> N2 = [0.2 0; 0 0.5];
>> sys = bilin(A,B,C,D,[N1 N2])

Discrete-time bilinear state-space model:
    x(t+Ts) = A x(t) + N kron(u(t),x(t)) + B u(t);   x(0) = X0
       y(t) = C x(t) + D u(t)

A =
    0.5000         0
         0    0.5000


B =
     0     1
    -1     0


C =
     1     0
     0     2


D =
     1     0
     1     1


N =
    0.6000         0    0.2000         0
         0    0.4000         0    0.5000


Initial state X0 =
     0
     0

Sampling time Ts =
     1
```

One can extract or set the properties of `sys` in a similar way as with `idss` objects:

```
>> sys.n

ans =

    0.6000         0    0.2000         0
         0    0.4000         0    0.5000
```

```
>> set(sys,'Ts',2)

Discrete-time bilinear state-space model:
    x(t+Ts) = A x(t) + N kron(u(t),x(t)) + B u(t);   x(0) = X0
       y(t) = C x(t) + D u(t)

A =
    0.5000         0
         0    0.5000


B =
     0     1
    -1     0


C =
     1     0
     0     2


D =
     1     0
     1     1


N =
    0.6000         0    0.2000         0
         0    0.4000         0    0.5000


Initial state X0 =
     0
     0

Sampling time Ts =
     2
```

The following code generates some random input and noise sequences that will be used to generate some data for identification:

```
>> W = iddata([],idinput([600 2],'RGS',[],[-0.01 0.01]));
>> V = iddata([],idinput([600 2],'RGS',[],[-0.01 0.01]));
>> U = iddata([],idinput([600 2],'RGS',[],[-0.1 0.1]));
```

The input sequence generated can be tested to see whether it satisfied the stability assumption made in [CM98]:

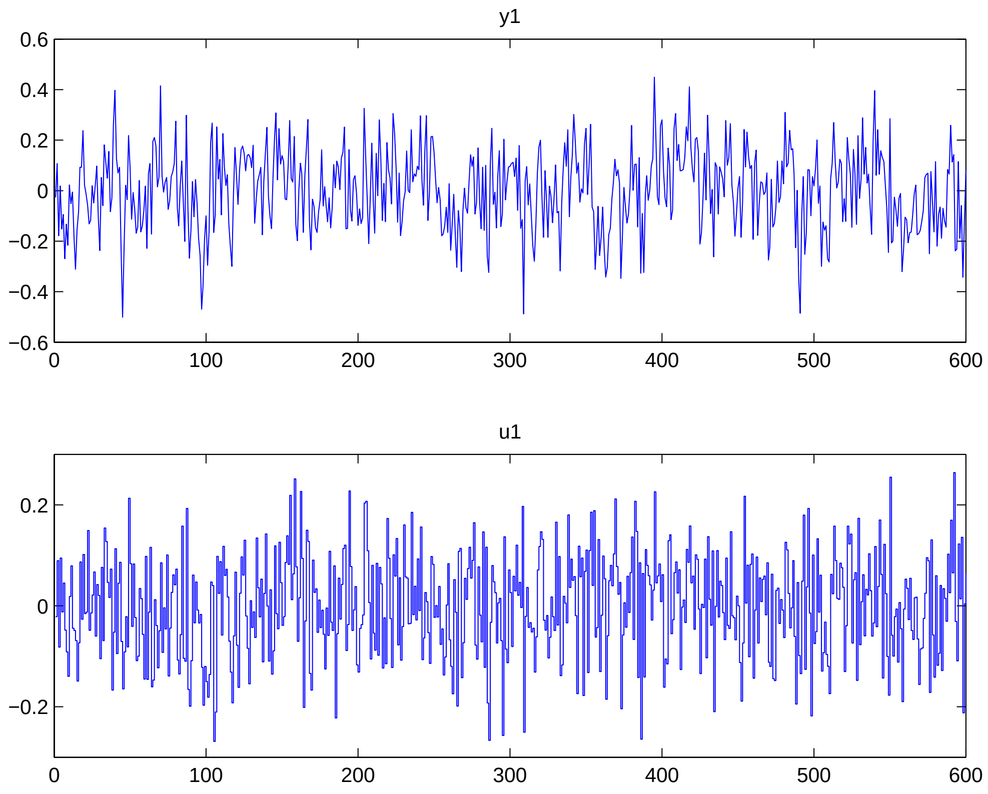

Figure 4.1: Input-output data generated from the bilinear system `sys`

```
>> isstable(sys,U)
The system satisfies the stability condition: Lambda = 6.91e-01 < 1.
```

One can now simulate the system with the given input sequence `U`, process noise `W` and measurement noise `V`:

```
>> [Y,X,YU] = sim(sys,[U W V]);
```

Figure 4.1 is a plot of the the first input and first output of `YU`, and was produced by:

```
>> plot(YU)
```

The following commands extract different parts of `YU` for identification and validation:

```
>> identdata = YU(1:550);
>> valdata = YU(551:600);
```

The function `bilinid` is now invoked to identify a bilinear system from the identification data.

```
>> M = bilinid(identdata,[1:4])
```

```
Using general four-block, deterministic-stochastic algorithm.
Constrained least squares will be used when estimating system matrices.

Block size k = 2.
Step 1/5. Decomposing the block equation...
...1/9...
...2/9...
...3/9...
...4/9...
...5/9...
...6/9...
...7/9...
...8/9...
...9/9...
Step 2/5. Computing the constant matrix via pseudo-inverse...
Garbage collection...
Step 3/5. Constructing matrices for SVD decomposition...
...1/9...
...2/9...
...3/9...
...4/9...
...5/9...
...6/9...
...7/9...
...8/9...
...9/9...
Step 4/5. Performing SVD decomposition...
Please select model order: 2
Garbage collection...
Step 5/5. Determining the system matrices using constrained least squares...

Done.

Discrete-time bilinear state-space model:
    x(t+Ts) = A x(t) + N kron(u(t),x(t)) + B u(t);   x(0) = X0
       y(t) = C x(t) + D u(t)

A =
    0.4916   -0.0068
   -0.0058    0.5079


B =
    0.7579    0.0004
   -0.0009    0.5311
```

```
C =
   -0.0095    1.8727
   -2.6179    0.0075


D =
    0.9870    0.0058
    1.0243    0.9910


N =
    0.4806   -0.0075    0.4995   -0.0767
    0.0549    0.6120    0.0911    0.0784


Initial state X0 =
     0
     0

Sampling time Ts =
     1
```

Figure 4.2 is a plot of the singular values that resulted from the SVD decomposition phase. As can be seen, the first two singular values are a significantly larger than the rest and the user correctly chose $n = 2$.

One can once again verify whether the estimated model satisfies the stability assumption of [CM99]:

```
>> isstable(M,U)
The system satisfies the stability condition: Lambda = 6.90e-01 < 1.
```

Finally, the estimated model can be validated using the functions `bilin/compare` or `bilin/pe`. Figure 4.3 results from the following call to the *CUEDSID Toolbox* function `bilin/compare`:

```
>> compare(M,valdata)

Percentage Fit:
   y1 - 93.41%
   y2 - 89.84%

>> compare(sys,valdata)

Percentage Fit:
   y1 - 93.45%
   y2 - 92.78%
```

As can be seen, `M` appears to be a good, initial estimate of `sys`.

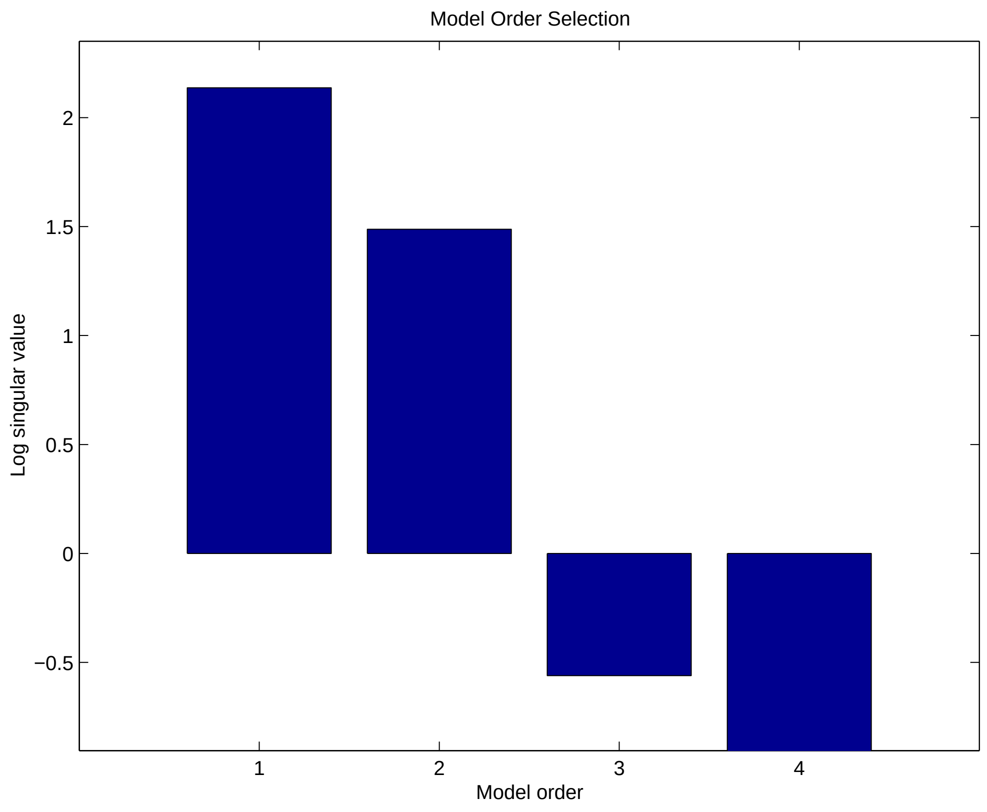

Figure 4.2: Plot of singular values resulting from the SVD decomposition phase in `bilinid`

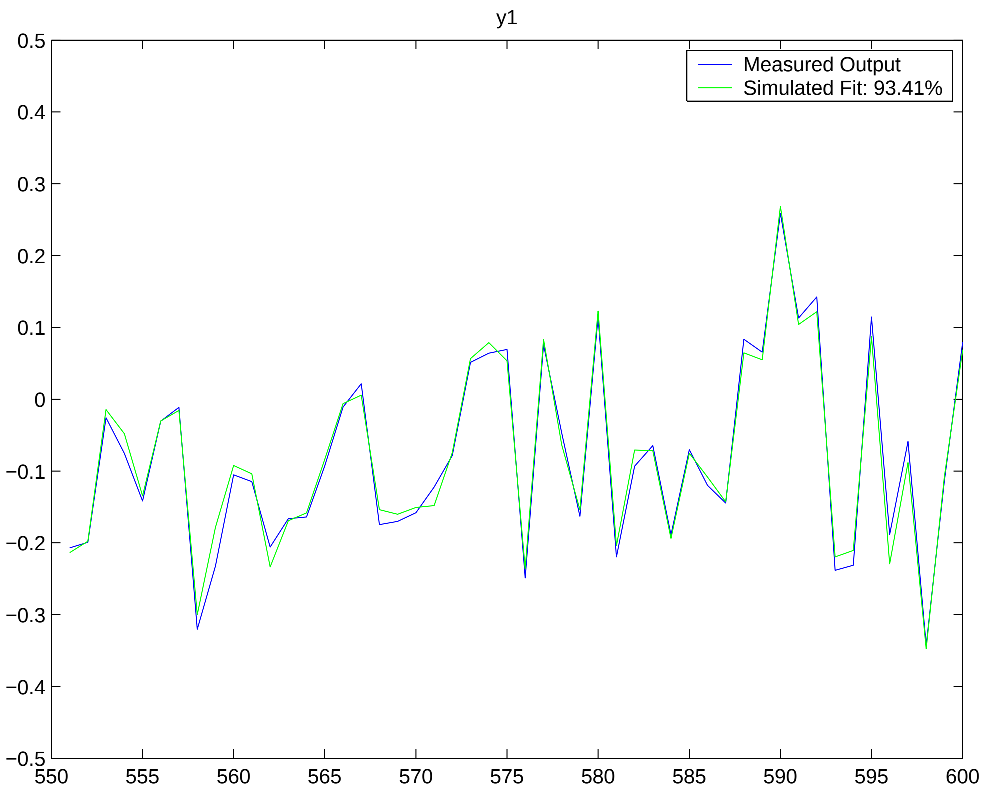

Figure 4.3: Comparison between the first output sequence in `valdata` and data that was simulated by using the estimated bilinear model `M`

Chapter 5

Function Reference

Subspace identification of linear systems

`subid3b`	Main function for linear subspace identification

Identification of linear systems using a balanced parameterization

`balpem`	Main function for identification using a balanced parameterization
`isbalanced`	Determines whether a state-space system is balanced
`ismpbalanced`	Determines whether a system is minimum-phase balanced
`mpbal`	Computes a minimum-phase balanced realization

Subspace identification of bilinear systems

`bilin`	Create a discrete-time, bilinear, state-space system
`bilin/compare`	Compare simulated data with measured data
`bilin/isstable`	Determine whether a given bilinear system is stable
`bilin/pe`	Compute prediction errors associated with a data set
`bilin/sim`	Simulate a given bilinear system (with noise)
`bilinid`	Main function for bilinear subspace identification

Other functions

`blochank`	Assemble a block Hankel matrix from a given block matrix
`bloctoep`	Assemble a block Toeplitz matrix from two given block matrices
`coorproj`	Orthogonal projection onto a complement subspace
`khatri`	Compute the Khatri-Rao product of two matrices
`obmat`	Construct the observability matrix with a given index
`orthproj`	Orthogonal projection onto a subspace
`soltritoep`	Solve for a lower-triangular, block Toeplitz matrix

balpem

Identifies a balanced state-space model from input-output data using a balanced parameterization prediction-error method.

Usage

```
M = balpem(DATA,Mi)
M = balpem(DATA,Mi,BD)
M = balpem(DATA,Mi,BD,ALG)
M = balpem(DATA,Mi,BD,ALG,Property_1,Value_1,...,Property_n,Value_n)
```

`DATA` is the input-output data given as an `iddata` object and `Mi` is an initial state-space estimate of the model, given as an `idss` object. The initial estimate `Mi` must be stable for the algorithm to work.

The estimated discrete-time, state-space model `M` is returned in innovation form as an `idss` object:

$$\begin{aligned} x(t+T_s) &= Ax(t) + Bu(t) + Ke(t), \quad x(0) = x_0 \\ y(t) &= Cx(t) + Du(t) + e(t) \end{aligned}$$

where T_s is the sample time and x_0 is the initial state. The estimated model `M` is stable, minimal and balanced in some sense (as determined by the choice of `ALG`).

See [CM97] for details of the algorithm.

Optional Inputs

- `BD` is used when estimating the B and D matrices and can be one of:

  'Estimate': Estimate B and D matrices (default).
  A warning is displayed if `Mi.nk` is not equal to $[0 \ \dots \ 0]$.

  'ZeroD': Estimate B and set $D = 0$.
  A warning is displayed if `Mi.nk` is not equal to $[1 \ \dots \ 1]$.

  'ZeroB': Set $B = 0$ and estimate D. Valid only if `ALG` is `'mp'`.
  A warning is displayed if `Mi.B` is not equal to 0 or `Mi.nk` is not equal to $[0 \ \dots \ 0]$.

  'ZeroBD': Set $B = 0$ and $D = 0$. Valid only if `ALG` is `'mp'`.
  A warning is displayed if `Mi.B` is not equal to 0 or `Mi.nk` is not equal to $[1 \ \dots \ 1]$.

- `ALG` determines the choice of balanced parameterization and can be one of:

  'sb': Stable, balanced parameterization (default). The estimated `M` is such that (A, B, C, D) is balanced and $K = 0$. The algorithm is described in [CM97, Sect. IV].
  A warning is displayed if `Mi.K` is not equal to zero.
  An error message is displayed if the initial estimate `Mi` is not minimal.

'mp': Minimum-phase balanced parameterization. The estimated `M` is such that the sub-system (A, K, C, I) of `M` is minimum-phase and balanced in the sense that the controllability gramian of (A, K, C, I) is diagonal and equal to the observability gramian of the *inverse* of (A, K, C, I). The algorithm is described in [CM97, Sect. V.A].

An error message is displayed if (A, K, C, I) of the initial estimate `Mi` is not minimum-phase and controllable, or the inverse of (A, K, C, I) is not observable.

- `Property,Value`: See `IDPROPS ALGORITHMS` or `IDPROPS IDGREY` for a list of possible Property/Value pairs. Typical properties that could be set include `'MaxIter'`, `'InitialState'` and `'DisturbanceModel'`. Setting the latter is sensible only when `ALG='sb'`.

Example

```
>> load examplebalpem
>> data
Data set with 300 samples.
Sampling interval: 1

Outputs      Unit (if specified)
   y1
   y2

Inputs       Unit (if specified)
   u1
   u2
   u3

>> idata = data(1:250);
>> valdata = data(251:300);
>> msb = balpem(idata,mi)
Warning: Mi.K is not allowed to be non-zero if ALG is 'sb'. Setting Mi.K = 0.
State-space model:  x(t+Ts) = A x(t) + B u(t) + K e(t)
                       y(t) = C x(t) + D u(t) + e(t)

A =
                       x1           x2           x3
           x1     0.95002     -0.08897    0.0010067
           x2   -0.029711     -0.62096      0.38682
           x3    0.058523     -0.35815     -0.33842


B =
                       u1           u2           u3
           x1    -0.24589     -0.70888      -1.6172
           x2      1.7191     -0.48856     -0.41401
           x3     -0.16323       0.51547        1.0581


C =
                        x1            x2            x3
           y1       1.7654        1.6652         0.254
           y2     -0.31006        0.6446        1.1529


D =
                        u1            u2            u3
           y1       1.1542      0.099871       0.99588
           y2      -1.1541    -0.0067296      -0.86999


K =
                        y1            y2
           x1            0             0
           x2            0             0
           x3            0             0


x(0) =

           x1            0
           x2            0
           x3            0

Estimated using BALPEM from data set data
Loss function 1.05851 and FPE 1.22276
Sampling interval: 1

>> mmp = balpem(idata,mi,[],'mp')
State-space model:  x(t+Ts) = A x(t) + B u(t) + K e(t)
                       y(t) = C x(t) + D u(t) + e(t)

A =
                        x1            x2            x3
           x1      0.93905     -0.047631        0.0216
           x2     -0.28762      -0.31094       0.33693
           x3    0.0099681      -0.42168      -0.64131


B =
                        u1            u2            u3
           x1      -6.9267       -12.855       -30.151
           x2      -11.043        4.1961        4.8219
           x3       19.958      -10.946      -16.077


C =
                        x1           x2           x3
           y1     0.056768     -0.16425     0.047706
           y2    -0.044647     -0.16137    -0.055189


D =
                        u1           u2           u3
           y1       1.1577     0.097444      0.98383
           y2      -1.1524    -0.016652     -0.87598


K =
                        y1           y2
           x1     0.062722    -0.076409
           x2      0.12789     -0.17769
           x3    -0.023432    0.0040451


x(0) =

           x1            0
           x2            0
           x3            0

Estimated using BALPEM from data set data
Loss function 1.12066 and FPE 1.35868
Sampling interval: 1

>> [yh,fit] = compare(valdata,mi,msb,mmp); fit

fit(:,:,1) =

   97.4080   97.4773   97.4719


fit(:,:,2) =

   92.3844   92.3163   92.3620

>> isbalanced(msb) % Check whether (A,B,C,D) of msb is balanced
System is balanced; norm(Wc-Wo) = 8.30e-15.
>> mpsys = ss(mmp.a,mmp.k,mmp.c,eye(2),mmp.Ts);
>> ismpbalanced(mpsys) % Check whether (A,K,C,I) of
```

```
                        % mmp is minimum-phase balanced
System is minimum-phase balanced; norm(Wc(SYS)-Wo(inv(SYS))) = 6.36e-17.
```

See Also

`PEM`, `SUBID3B`, `N4SID`, `ISBALANCED`, `ISMPBALANCED`, `MPBAL`.

bilin

Create a discrete-time, bilinear state-space model.

Usage

```
M = bilin(A,B,C,D,N)
M = bilin(A,B,C,D,N,X0)
M = bilin(A,B,C,D,N,X0,Ts)
```

The output `M` is a discrete-time bilinear, state-space model, returned as a `bilin` object. The model `M` is given by

$$\begin{aligned} x(t+T_s) &= Ax(t) + N(u(t) \otimes x(t)) + Bu(t), \quad x(0) = x_0 \\ y(t) &= Cx(t) + Du(t) \end{aligned}$$

where $\otimes$ is the Kronecker product operator, the matrices $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{p\times n}$, $D \in \mathbb{R}^{p\times m}$, $N := [N_1 \ N_2 \ \cdots N_m] \in \mathbb{R}^{n\times nm}$ and each $N_i \in \mathbb{R}^{n\times n}$.

Optional arguments are `X0`, which sets the initial state x_0 (default is `X0=zeros(n,1)`), and `Ts`, which sets the sample time T_s (default is `Ts=1`).

Example

```
>> A = [1 2; 3 4]; B = [5 6; 0 1]; C = [7 8]; D = [9 0];
>> N1 = [10 11; 12 13]; N2 = [14 15; 16 17];
>> M = bilin(A,B,C,D,[N1 N2],[0.1;0.2],3)

Discrete-time bilinear state-space model:
    x(t+Ts) = A x(t) + N kron(u(t),x(t)) + B u(t);   x(0) = X0
       y(t) = C x(t) + D u(t)

A =
     1     2
     3     4



B =
     5     6
     0     1



C =
     7     8



D =
     9     0


N =
    10    11    14    15
    12    13    16    17


Initial state X0 =
    0.1000
    0.2000

Sampling time Ts =
     3

>> M.D = [0 0]  % Equivalent to set(M,'D',[0 0])

Discrete-time bilinear state-space model:
    x(t+Ts) = A x(t) + N kron(u(t),x(t)) + B u(t);   x(0) = X0
       y(t) = C x(t) + D u(t)

A =
     1     2
     3     4


B =
     5     6
     0     1


C =
     7     8


D =
     0     0


N =
    10    11    14    15
    12    13    16    17


Initial state X0 =
    0.1000
    0.2000
```

```
Sampling time Ts =
     3

>> M.X0 % Equivalent to get(M,'X0')

ans =

    0.1000
    0.2000
```

See Also

BILINID, BILIN/COMPARE, BILIN/ISSTABLE, BILIN/PE, BILIN/SIM.

bilin/compare

Compares simulated output data for a bilinear model with the measured data.

Usage

```
compare(M,DATA)
[YH,FIT] = compare(M,DATA)
[YH,FIT] = compare(M,DATA,INIT)
```

`M` is the bilinear system given as a `bilin` object and `DATA` is the input-output data given as an `iddata` object.

In the absence of output arguments, `compare(M,DATA)` outputs the percentage fit to the workspace and produces plots comparing the measured and simulated outputs.

Optional Outputs

`YH` is the resulting simulated/predicted output returned as an `iddata` object.

`FIT` is a vector containing the percentage of the measured output that is explained by the model, with `FIT(1)` being the percentage fit of the first output, etc.

Optional Input

The argument `INIT` determines how to deal with initial conditions and can be one of:

'estimate': Results in the initial state being chosen so that the norm of the prediction error is minimized (default).

'model': Uses `M.X0` as the initial state.

'zero': Sets the initial state to zero.

A column vector: Uses `INIT` as the initial state.

Example

```
>> load example2i2s2o
>> [YH,FIT] = compare(m,valdata)
Data set with 50 samples.
Sampling interval: 1

Outputs      Unit (if specified)
   y1
   y2
```

```
FIT =

   93.4257
   92.3414
```

See Also

`BILINID`, `BILIN`, `BILIN/ISSTABLE`, `BILIN/PE`, `BILIN/SIM`, `IDMODEL/COMPARE`, `IDDATA/PLOT`.

bilin/isstable

Determines whether a bilinear system satisfies a kind of stability condition.

Usage

```
ISSTABLE(M,DATA)
[FVAL,LAMBDA] = ISSTABLE(M,DATA)
```

In [CM99] it is assumed that the discrete-time bilinear system

$$\begin{aligned} x(t+T_s) &= Ax(t) + N(u(t) \otimes x(t)) + Bu(t), \quad x(0) = x_0 \\ y(t) &= Cx(t) + Du(t) \end{aligned}$$

satisfies the following assumption, which is a kind of stability condition:

$$\lambda = \max_t \ \overline{\sigma} \left(A + \sum_{i=1}^{m} u_i(t) N_i \right) < 1, \quad \text{s.t. } t \in \{0, \ldots, \tilde{N} - 1\},$$

where $N := [N_1 \ \cdots \ N_m]$, $\overline{\sigma}(\cdot)$ denotes the greatest singular value of a matrix and $u_i(t)$ is the i'th element of $u(t)$ in the sequence $\{u(0), \ldots, u(\tilde{N}-1)\}$.

The system `M` to be tested should be given as a `bilin` object.

`DATA` should be an `iddata` object and the input sequence $\{u(0), \ldots, u(\tilde{N}-1)\}$ is taken from `DATA.InputData`.

`ISSTABLE(M,DATA)` computes λ and is true if $\lambda < 1$ and false if $\lambda \geq 1$.

Optional outputs

`FVAL` is returned as 1 if $\lambda < 1$ and `FVAL` is returned as 0 if $\lambda \geq 1$.

`LAMBDA` is the computed value of λ.

Example

```
if isstable(m,U)
  disp('Lambda is less than 1.')
else
  disp('Lambda is not less than 1.)
end
```

or

```
>> isstable(sys,U)
The system satisfies the stability condition: Lambda = 7.28e-01 < 1.
```

or

```
>> [fval,lambda]=isstable(sys,U)

fval =

     1


lambda =

    0.7277
```

See Also

`BILINID`, `BILIN`, `BILIN/COMPARE`, `BILIN/PE`, `BILIN/SIM`.

bilin/pe

Compute the prediction errors associated with a bilinear model and data set.

Usage

```
[E,X0] = pe(M,DATA)
[E,X0] = pe(M,DATA,INIT)
```

`M` is the bilinear system given as a `bilin` object and `DATA` is the input-output data given as an `iddata` object.

`E` is returned as an `iddata` object, so that `E.OutputData` contains the prediction errors that result when model `M` is applied to `DATA`.

`E.InputData` is set to `DATA.InputData`.

Optional Output

`X0` is the value that was used for the initial state.

Optional Input

The argument `INIT` determines how to deal with initial conditions and can be one of:

'estimate': Results in the initial state `X0` being chosen so that the norm of the prediction error is minimized (default).

'model': Uses `M.X0` as the initial state `X0`.

'zero': Sets the initial state `X0` to zero.

A column vector: Uses `INIT` as the initial state `X0`.

Example

```
>> load example2i2s2o
>> [E,X0] = pe(m,valdata); % estimate initial state X0
>> norm(E.y)

ans =

    0.1288

>> X0

X0 =
    0.0825
    0.1248

>> [E,X0] = pe(m,valdata,'zero'); % use initial state X0=0
>> norm(E.y)

ans =

    0.3599

>> X0

X0 =

     0
     0
```

See Also

BILINID, BILIN, BILIN/COMPARE, BILIN/ISSTABLE, BILIN/SIM, IDMODEL/PE, IDMODEL/RESID, IDDATA/PLOT

bilin/sim

Simulates a given bilinear system.

Usage

```
[Y,X,YU] = sim(M,UE)
[Y,X,YU] = sim(M,UE,INIT)
```

`M` is the bilinear system given as a `bilin` object and `UE` is an `iddata` object with the input and/or noise data contained in `UE.InputData`.

`UE` can be given as `U`, `[U W]` or `[U W V]` where `U`, `W` and `V` are `iddata` objects with compatible dimensions; `U.InputData` should contain the input sequence $\{u(t)\}$, `W.InputData` the process noise sequence $\{w(t)\}$ and `V.InputData` the measurement noise sequence $\{v(t)\}$ of the bilinear system:

$$\begin{aligned} x(t+T_s) &= Ax(t) + N(u(t)\otimes x(t)) + Bu(t) + w(t), \quad x(0) = x_0 \\ y(t) &= Cx(t) + Du(t) + v(t) \end{aligned}$$

where T_s is the sample time and x_0 is the initial state.

`Y`, `X` and `YU` are `iddata` objects. `Y.OutputData` contains the output sequence $\{y(t)\}$, `X.OutputData` contains the state sequence $\{x(t)\}$ and `YU` contains the input-output data sequence $\{(y(t), u(t))\}$. `YU.OutputData` is the output sequence $\{y(t)\}$ and `YU.InputData` is the input sequence $\{u(t)\}$, i.e. `YU.InputData=U.InputData`.

Optional Input

The argument `INIT` determines how to deal with initial conditions and can be one of:

'model': Uses `M.X0` as the initial state x_0 (default).

'zero': Sets the initial state x_0 to zero.

A column vector: Uses `INIT` as the initial state.

Example

```
>> load example2i2s2o
>> sys % system with 2 inputs, 2 states and 2 outputs

Discrete-time bilinear state-space model:
    x(t+Ts) = A x(t) + N kron(u(t),x(t)) + B u(t);   x(0) = X0
       y(t) = C x(t) + D u(t)

A =
    0.5000         0
```

```
         0    0.3000


B =
     0     1
    -1     0


C =
     1     0
     0     2


D =
     1     0
     0     1


N =
    0.6000         0    0.2000         0
         0    0.4000         0    0.5000


Initial state X0 =
     0
     0

Sampling time Ts =
     1

>> U = idinput([600 2],'RGS',[],[-0.1 0.1]); % U, W, V are random sequences
>> W = idinput([600 2],'RGS',[],[-0.01 0.01]);
>> V = idinput([600 2],'RGS',[],[-0.01 0.01]);
>> U = iddata([],U); % input data
>> W = iddata([],W); % process noise
>> V = iddata([],V); % measurement noise
>> [Y,X,YU] = sim(sys,[U W V]); % simulate system
>> isstable(sys,U) % check whether system is stable with given input data
The system satisfies the stability condition: Lambda = 6.86e-01 < 1.
>> data = YU(1:550); % identification data
>> valdata = YU(551:600); % validation data
```

See Also

BILINID, BILIN, BILIN/COMPARE, BILIN/ISSTABLE, BILIN/PE, IDINPUT, IDDATA, IDMODEL/SIM.

bilinid

Deterministic-stochastic subspace identification of bilinear systems using a four-block configuration.

Usage

```
[M,EXTRA] = bilinid(DATA)
[M,EXTRA] = bilinid(DATA,n)
[M,EXTRA] = bilinid(DATA,n,k)
[M,EXTRA] = bilinid(DATA,n,k,DMAT)
[M,EXTRA] = bilinid(DATA,n,k,DMAT,ALG)
[M,EXTRA] = bilinid(DATA,n,k,DMAT,ALG,LS)
```

`DATA` is the input-output data given as an `iddata` object.

`M` is the estimated discrete-time, bilinear state-space model returned as a `bilin` object:

$$\begin{aligned} x(t+T_s) &= Ax(t) + N(u(t)\otimes x(t)) + Bu(t) + w(t), \quad x(0) = x_0 \\ y(t) &= Cx(t) + Du(t) + v(t) \end{aligned}$$

where T_s is the sample time, x_0 the initial state, $\otimes$ is the Kronecker product operator, the matrices $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{p\times n}$, $D \in \mathbb{R}^{p\times m}$, $N := [N_1\ N_2\ \cdots N_m] \in \mathbb{R}^{n\times nm}$ and each $N_i \in \mathbb{R}^{n\times n}$.

See [CM99] for details of the algorithm.

Optional Output

`EXTRA` is a structure containing additional information about the model and data:

- `EXTRA.SV` is a vector containing the singular values resulting from the SVD decomposition. See [CM99] for details.
- `EXTRA.Q`, `EXTRA.R` and `EXTRA.S` are the matrices that form the joint noise covariance matrix `EXTRA.COV`, which is given by `EXTRA.COV = [Q S; S' R]`; `EXTRA.Q` is the process noise covariance matrix and `EXTRA.R` is the output noise covariance matrix. See [CM99] for details.
- If `LS='ols'`, then `EXTRA.ls` is a matrix containing information about the accuracy of the estimation and should be close to 0 for a good estimate. If `LS='cls'`, then `EXTRA.ls=0`. See [CM99] for details.

Optional Inputs

- `n` is the system order and can be one of the following:
  - If `n` is empty, then the algorithm will automatically select the order such that $\mathtt{n} \leq 10$ (default).

  - If `n` is a scalar, then the system order is equal to `n`.
  - If `n` is given as a row vector (e.g. [1 2 3 4 5]), a plot of singular values will be given and the user will be prompted to select an order.

- `k` specifies the block size, given as a positive integer. If `k` is not specified, the block size is chosen to be as large as possible while still trying to be compatible with the size of `DATA`. It is recommended that `k` $\geq$ `max(n)`.
- `DMAT` determines whether the D matrix of the system is to be estimated or set to zero. `DMAT` can be one of the following:

  'Estimate': Estimate the D matrix (default).

  'Zero': Set $D = 0$.

- `ALG` determines the specific algorithm to be used and can be one of the following:

  'general': General four-block deterministic-stochastic algorithm.

  'fast': Fast four-block deterministic-stochastic algorithm. Valid only if the number of outputs < `min(n)`.

  'accurate': Accurate four-block deterministic-stochastic algorithm. Valid only if the number of outputs $\geq$ `max(n)`.

  If `ALG` is not specified, then the most appropriate algorithm is automatically chosen, based on the number of outputs. If the number of outputs < `max(n)`, then `ALG='general'`. If the number of outputs $\geq$ `max(n)`, then `ALG='accurate'`.

- `LS` determines the choice of least squares method used in estimating the system matrices and can be one of:

  'cls': Constrained least squares (default).

  'ols': Ordinary least squares.

Example

```
>> load example2i2s2o
>> data
Data set with 550 samples.
Sampling interval: 1

Outputs      Unit (if specified)
   y1
   y2

Inputs       Unit (if specified)
   u1
   u2

>> [m,extra] = bilinid(data,2)
```

```
Number of outputs >= system order. Using accurate four-block algorithm.
Constrained least squares will be used when estimating system matrices.

Block size k = 3.
Step 1/5. Decomposing the block equation...
...1/9...
...2/9...
...3/9...
...4/9...
...5/9...
...6/9...
...7/9...
...8/9...
...9/9...
Step 2/5. Computing the constant matrix via pseudo-inverse...
Garbage collection...
Step 3/5. Constructing matrices for SVD decomposition...
...1/9...
...2/9...
...3/9...
...4/9...
...5/9...
...6/9...
...7/9...
...8/9...
...9/9...
Step 4/5. Performing SVD decomposition...
Garbage collection...
Step 5/5. Determining the system matrices using constrained least squares...
Garbage collection...
Finally, determining the noise covariance matrix...

Done.


Discrete-time bilinear state-space model:
    x(t+Ts) = A x(t) + N kron(u(t),x(t)) + B u(t);   x(0) = X0
       y(t) = C x(t) + D u(t)

A =
    0.3011   -0.0184
   -0.0098    0.4983


B =
    0.7763   -0.0188
    0.0399    0.5451


C =
   -0.1024    1.8279
   -2.5645   -0.1099


D =
    1.0000   -0.0141
    0.0215    1.0064


N =
    0.4256   -0.0235    0.4452   -0.0432
    0.0250    0.6812    0.0450    0.2123


Initial state X0 =
     0
     0

Sampling time Ts =
     1


extra =

     ls: [2x4 double]
     sv: [26x1 double]
    COV: [4x4 double]
      Q: [2x2 double]
      R: [2x2 double]
      S: [2x2 double]

>> extra.COV

ans =

   1.0e-03 *

    0.0776   -0.0055    0.0031   -0.0265
   -0.0055    0.0563    0.0114    0.0044
    0.0031    0.0114    0.1718    0.0206
   -0.0265    0.0044    0.0206    0.4013

>> [YH,FIT] = compare(m,valdata)
```

```
Data set with 50 samples.
Sampling interval: 1

Outputs      Unit (if specified)
   y1
   y2


FIT =

   93.4257
   92.3414
```

See Also

`BILIN`, `BILIN/COMPARE`, `BILIN/ISSTABLE`, `BILIN/SIM`, `BILIN/PE`.

blochank

Assembles a block Hankel matrix from a given block matrix.

Usage

```
Y = blochank(C,M,N)
```

C is a block matrix and given by

$$C = \begin{bmatrix} C_1 \\ C_2 \\ \vdots \\ C_N \end{bmatrix},$$

where all the C_i are matrices with the same dimensions. M and N are positive integers with $M \leq N$.

The resulting block Hankel matrix Y is given by

$$Y = \begin{bmatrix} C_1 & C_2 & C_3 & \cdots & C_{N-M+1} \\ C_2 & C_3 & C_4 & \cdots & C_{N-M+2} \\ C_3 & C_4 & C_5 & \cdots & C_{N-M+3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ C_M & C_{M+1} & C_{M+2} & \cdots & C_N \end{bmatrix}$$

Example

```
>> C = [1 2; 3 4; 5 6; 7 8; 9 10;
         11 12; 13 14; 15 16; 17 18; 19 20]

C =

     1     2
     3     4
     5     6
     7     8
     9    10
    11    12
    13    14
    15    16
    17    18
    19    20

>> Y = blochank(C,3,5)

Y =
     1     2     5     6     9    10
     3     4     7     8    11    12
     5     6     9    10    13    14
     7     8    11    12    15    16
     9    10    13    14    17    18
    11    12    15    16    19    20
```

See Also

`HANKEL`, `BLOCTOEP`, `TOEPLITZ`.

bloctoep

Assembles a block Toeplitz matrix from two given block matrices.

Usage

```
Y = bloctoep(C,R,N)
```

C and R are block matrices given by

$$C = \begin{bmatrix} C_1 \\ C_2 \\ \vdots \\ C_N \end{bmatrix}$$

and

$$R = \begin{bmatrix} R_1 \\ R_2 \\ \vdots \\ R_N \end{bmatrix}$$

where all the C_i and R_j are matrices with the same dimensions. It is assumed that $C_1 = R_1$. N is a positive integer and is the number of block matrices in C and R.

The resulting block Toeplitz matrix Y has C as its first block column and R as its first block row, i.e.

$$Y = \begin{bmatrix} C_1 & R_2 & R_3 & \cdots & R_N \\ C_2 & C_1 & R_2 & \cdots & R_{N-1} \\ C_3 & C_2 & C_1 & \cdots & R_{N-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ C_N & C_{N-1} & C_{N-2} & \cdots & C_1 \end{bmatrix}$$

Example

```
>> C = [1 2; 3 4; 5 6; 7 8; 9 10; 11 12; 13 14; 15 16]

C =

     1     2
     3     4
     5     6
     7     8
     9    10
    11    12
    13    14
    15    16
```

```
>> R = [1 2; 3 4; 17 18; 19 20; 21 22; 23 24; 25 26; 27 28]

R =

     1     2
     3     4
    17    18
    19    20
    21    22
    23    24
    25    26
    27    28

>> Y = bloctoep(C,R,4)

Y =

     1     2    17    18    21    22    25    26
     3     4    19    20    23    24    27    28
     5     6     1     2    17    18    21    22
     7     8     3     4    19    20    23    24
     9    10     5     6     1     2    17    18
    11    12     7     8     3     4    19    20
    13    14     9    10     5     6     1     2
    15    16    11    12     7     8     3     4
```

See Also

`TOEPLITZ`, `BLOCHANK`, `HANKEL`.

coorproj

Orthogonal projection onto a complement subspace.

Usage

```
Y = coorproj(A,B)
```

Projects the row vectors of matrix $A \in \mathbb{R}^{m\times p}$ onto the subspace that is orthogonal to the subspace spanned by the row vectors of matrix $B \in \mathbb{R}^{n\times p}$, i.e.

$$Y = \Pi_{\mathcal{B}^\perp} A := AC^T(CC^T)^\dagger C,$$

where Π is the orthogonal projection operator, $\mathcal{B} := \operatorname{span}\{\alpha^T B, \alpha \in \mathbb{R}^n\}$, $\mathcal{B}^\perp$ is the orthogonal complement of $\mathcal{B}$,

$$C := I - B^T(BB^T)^\dagger B$$

and $(\cdot)^\dagger$ denotes the Moore-Penrose (pseudo-inverse) of a matrix.

The number of columns of A and B must be the same.

Example

```
>> A = rand(3,5)

A =

    0.9153    0.9305    0.1339    0.1292    0.1953
    0.4045    0.6019    0.6317    0.3885    0.2160
    0.5885    0.3396    0.2573    0.1179    0.5965

>> B = [1 0 0 0 0; 1 1 0 0 0; 1 1 1 0 0]

B =

     1     0     0     0     0
     1     1     0     0     0
     1     1     1     0     0

>> Y = coorproj(A,B)

Y =

         0         0         0    0.1292    0.1953
         0         0         0    0.3885    0.2160
         0         0         0    0.1179    0.5965
```

See Also

`ORTHPROJ`, `PINV`.

isbalanced

Determines whether a given state-space system is balanced.

Usage

`isbalanced(sys)`

`isbalanced(sys)` is true if (A, B, C) of `sys` is balanced and false if it is not balanced.

The system is balanced if and only if the observability gramian W_o and the controllability gramian W_c of (A, B, C) are equal and diagonal.

`sys` has to be a stable state-space system, given as an `ss` or `idss` object.

Example

```
if isbalanced(sys)
  disp('System is balanced.')
else
  disp('System is not balanced.')
end
```

See Also

`GRAM`, `BALREAL`, `MPBAL`, `ISMPBALANCED`, `BALPEM`.

ismpbalanced

Determines whether a given state-space system is minimum-phase balanced.

Usage

`ismpbalanced(sys)`

`ismpbalanced(sys)` is true if (A, B, C, D) of `sys` is minimum-phase balanced and false if it is not minimum-phase balanced.

The system is minimum-phase balanced if and only if the controllability gramian W_c of (A, B, C, D) is equal to the observability gramian W_o of the *inverse* of (A, B, C, D).

`sys` has to be a square, stable and minimum-phase state-space system, given as an `ss` or `idss` object.

Example

```
if ismpbalanced(sys)
  disp('System is minimum-phase balanced.')
else
  disp('System is not minimum-phase balanced.')
end
```

See Also

`MPBAL`, `GRAM`, `ISBALANCED`, `BALREAL`, `BALPEM`.

khatri

Computes the Khatri-Rao product of two matrices.

Usage

```
Y = khatri(A,B)
```

Computes the Khatri-Rao product of two matrices $A \in \mathbb{R}^{m \times p}$ and $B \in \mathbb{R}^{n \times p}$ with the same number of columns, i.e. $Y \in \mathbb{R}^{mn \times p}$ is formed by taking the Kronecker tensor products between the respective columns of $A := [a_1\ a_2\ \cdots\ a_p]$ and $B := [b_1\ b_2\ \cdots\ b_p]$, i.e.

$$Y = A \odot B := \begin{bmatrix} a_1 \otimes b_1 & a_2 \otimes b_2 & \cdots & a_p \otimes b_p \end{bmatrix},$$

where $\otimes$ is the Kronecker product operator.

Example

```
>> A = [1 2; 3 4]

A =

     1     2
     3     4

>> B = [5 6; 7 8]

B =

     5     6
     7     8

>> Y = khatri(A,B)

Y =

     5    12
     7    16
    15    24
    21    32
```

See Also

KRON.

mpbal

Minimum-phase balancing of a minimum-phase state-space realization.

Usage

```
[balsys,G,T,Ti] = mpbal(sys)
```

`balsys` is a minimum-phase, balanced realization of the system `sys` in the sense that the controllability gramian of `balsys` and the observability gramian of the *inverse* of `balsys` are equal and diagonal [CM97, Sect. V.A].

`sys` has to be a *continuous-time*, controllable, stable, minimum-phase state-space system, given as an `ss` object. The inverse of `sys` must be observable.

Optional Outputs

`G` is a vector containing the diagonal of the gramian of the minimum-phase balanced realization. The matrix `T` is the state transformation $z = Tx$ that was used to convert `sys` to `balsys`, and `Ti` is its inverse.

Example

```
>> sys = rss(3)

a =
                       x1           x2           x3
           x1     -2.7113     -0.32309    -0.056596
           x2    -0.32309     -0.45836     0.022362
           x3   -0.056596     0.022362     -0.34326


b =
                       u1
           x1     -1.7509
           x2    -0.82862
           x3      1.3862


c =
                       x1           x2           x3
           y1     0.27187     -0.61306     0.021796


d =
                       u1
           y1       1.0392

Continuous-time model.

>> [balsys,G,T,Ti] = mpbal(sys)

a =
                       x1           x2           x3
           x1    -0.30374     -0.70872      0.10028
           x2     0.68176      -2.7244      0.75233
           x3   -0.096421      0.75196      -0.4848


b =
                       u1
           x1     0.41672
           x2    -0.41414
           x3    0.065091


c =
                       x1           x2           x3
           y1     0.57419      0.41703    -0.067346


d =
                       u1
           y1       1.0392

Continuous-time model.

G =

    0.2859
    0.0315
    0.0044


T =

    0.2290   -0.9776    0.0054
    0.3033   -0.0031    0.0824
   -0.2071    0.7487    0.2329


Ti =
   -0.8588    3.1894   -1.1081
   -1.2067    0.7493   -0.2369
    3.1149    0.4267    4.0692

>> Wc = gram(balsys,'c')

Wc =

    0.2859    0.0000   -0.0000
    0.0000    0.0315   -0.0000
   -0.0000   -0.0000    0.0044

>> Wo = gram(inv(balsys),'o')

Wo =

    0.2859   -0.0000    0.0000
   -0.0000    0.0315    0.0000
    0.0000    0.0000    0.0044

>> ismpbalanced(sys)
System is not minimum-phase balanced; norm(Wc(SYS)-Wo(inv(SYS))) = 3.47e+00.
>> ismpbalanced(balsys)
System is minimum-phase balanced; norm(Wc(SYS)-Wo(inv(SYS))) = 7.79e-16.
```

See Also

GRAM, ISBALANCED, BALREAL, BALPEM.

obmat

Constructs the observability matrix with a given index.

Usage

```
Y = obmat(A,C,k)
```

Given matrices $A \in \mathbb{R}^{n\times n}$ and $C \in \mathbb{R}^{p\times n}$ of a state-space system (A, B, C, D) with n states and p outputs, the truncated $(k < n)$, standard $(k = n)$ or extended $(k > n)$ observability matrix is constructed, i.e.

$$Y = \mathcal{O}_k := \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{k-1} \end{bmatrix}$$

Example

```
>> A = [1 -0.1; -0.1 1]

A =

    1.0000   -0.1000
   -0.1000    1.0000

>> C = [1 0; 1 1]

C =

     1     0
     1     1

>> Y = obmat(A,C,4)

Y =

    1.0000         0
    1.0000    1.0000
    1.0000   -0.1000
    0.9000    0.9000
    1.0100   -0.2000
    0.8100    0.8100
    1.0300   -0.3010
    0.7290    0.7290
```

See Also

`OBSV`.

orthproj

Orthogonal projection onto a subspace.

Usage

```
Y = orthproj(A,B)
```

Projects the row vectors of matrix $A \in \mathbb{R}^{m \times p}$ onto the subspace spanned by the row vectors of matrix $B \in \mathbb{R}^{n \times p}$, i.e.

$$Y = \Pi_{\mathcal{B}} A := AB^T(BB^T)^{\dagger}B$$

where Π is the orthogonal projection operator, $\mathcal{B} := \text{span}\{\alpha^T B, \alpha \in \mathbb{R}^n\}$ and $(\cdot)^{\dagger}$ denotes the Moore-Penrose (pseudo-inverse) of a matrix.

The number of columns of A and B must be the same.

Example

```
>> A = rand(3,5)

A =

    0.3468    0.1520    0.3879    0.8118    0.5601
    0.8625    0.9218    0.8235    0.5281    0.2114
    0.4751    0.4033    0.2914    0.2015    0.9282

>> B = [1 0 0 0 0; 1 1 0 0 0; 1 1 1 0 0]

B =

     1     0     0     0     0
     1     1     0     0     0
     1     1     1     0     0

>> Y = orthproj(A,B)

Y =

    0.3468    0.1520    0.3879         0         0
    0.8625    0.9218    0.8235         0         0
    0.4751    0.4033    0.2914         0         0
```

See Also

`COORPROJ`, `PINV`.

soltritoep

Solves for a least squares, lower triangular, block Toeplitz matrix.

Usage

```
[T,H] = soltritoep(S,P,k)
```

Given two matrices $S \in \mathbb{R}^{nk\times p}$ and $P \in \mathbb{R}^{mk\times p}$ with the same number of columns and with the number of rows divisible by k, the matrix

$$T = \begin{bmatrix} H_0 & 0 & \cdots & 0 & 0 \\ H_1 & H_0 & & & 0 \\ \vdots & & \ddots & & \vdots \\ H_{k-2} & H_{k-3} & \cdots & H_0 & 0 \\ H_{k-1} & H_{k-2} & \cdots & H_1 & H_0 \end{bmatrix}$$

is a least squares, lower triangular, block Toeplitz matrix solution to

$$S = TP.$$

The block matrix H is the first block column of T, i.e.

$$H = \begin{bmatrix} H_0 \\ H_1 \\ \vdots \\ H_{k-1} \end{bmatrix}$$

and each $H_i \in \mathbb{R}^{n\times m}$.

See [CM98, Sect. 6] for a description of the algorithm.

Example

```
>> S = rand(4,1)

S =

    0.5514
    0.4373
    0.3705
    0.1322

>> P = rand(6,1)

P =
    0.4460
    0.9884
    0.6820
    0.7749
    0.7643
    0.7025

>> [T,H] = soltritoep(S,P,2)

T =

         0    0.5578         0         0         0         0
         0    0.4424         0         0         0         0
         0   -0.0565         0         0    0.5578         0
         0   -0.2084         0         0    0.4424         0


H =

         0    0.5578         0
         0    0.4424         0
         0   -0.0565         0
         0   -0.2084         0
```

See Also

`BLOCTOEP`, `TOEPLITZ`.

subid3b

Deterministic-stochastic subspace identification of linear systems using a three-block configuration.

Usage

```
[M,P,SV] = subid3b(DATA)
[M,P,SV] = subid3b(DATA,n)
[M,P,SV] = subid3b(DATA,n,k)
[M,P,SV] = subid3b(DATA,n,k,DMAT)
[M,P,SV] = subid3b(DATA,n,k,DMAT,ALG)
```

`DATA` is the input-output data given as an `iddata` object and `M` is the estimated discrete-time, state-space model in innovation form, returned as an `idss` object:

$$\begin{aligned} x(t+T_s) &= Ax(t) + Bu(t) + Ke(t), \quad x(0) = x_0 \\ y(t) &= Cx(t) + Du(t) + e(t) \end{aligned}$$

where $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{p\times n}$, $D \in \mathbb{R}^{p\times m}$, $K \in \mathbb{R}^{n\times p}$, T_s is the sample time and x_0 is the initial state.

See [CM98] for details of the algorithm.

Optional Outputs

- `P` is the associated Kalman filter covariance matrix [CM98, Sect. 9].
- `SV` is the retained singular values from the SVD decomposition.

Optional Inputs

- `n` is the system order and can be one of the following:
  - If `n` is empty, then the algorithm will automatically select the order such that `n` ≤ 10 (default).
  - If `n` is a scalar, then the system order is equal to `n`.
  - If `n` is given as a row vector (e.g. [1 2 3 4 5]), a plot of singular values will be given and the user will be prompted to select an order.
- `k` specifies the block size, given as a positive integer. If `k` is not specified, the block size is chosen to be as large as possible while still trying to be compatible with the size of `DATA`.

  It is recommended that the user ensure that `k` $>$ `max(n)`, but that `k` is still a lot smaller than the size of `DATA`. For a more detailed discussion regarding the choice of block size, see [CM98, p. 24].

- `DMAT` determines whether the D matrix of the system is to be estimated or set to zero. `DMAT` can be one of the following:

  'Estimate': Estimate the D matrix (default).

  'Zero': Set $D = 0$.

- `ALG` is set to choose the specific identification algorithm and can be one of the following:

  'si': Shift invariance approach (default).
  See [CM98, Alg. 10.4] for details of the algorithm.

  'ss': State sequence approach.
  See [CM98, Alg. 10.5] for details of the algorithm.

  'mp': Markov parameter approach.
  See [CM98, Alg. 10.3] for details of the algorithm.

  The shift invariance approach is the most computationally demanding algorithm, but often results in obtaining the best estimate. The Markov parameter approach is often the least robust.

Example

```
>> load example3block
>> data
Data set with 300 samples.
Sampling interval: 1

Outputs      Unit (if specified)
   y1
   y2

Inputs       Unit (if specified)
   u1
   u2
   u3

>> idata = data(1:250);
>> valdata = data(251:300);
>> [m,p,sv]=subid3b(idata,3)
Block size k = 20.
State-space model:  x(t+Ts) = A x(t) + B u(t) + K e(t)
                       y(t) = C x(t) + D u(t) + e(t)

A =
                       x1           x2           x3
           x1     0.95181    -0.084857   0.00053211
           x2   0.0021275     -0.61431       0.4012
           x3    0.057737     -0.33869     -0.34948
```

```
B =
                     u1           u2           u3
          x1   0.025562     0.071337      0.16243
          x2   -0.17306     0.050237     0.042829
          x3    0.02156     -0.05271     -0.10584


C =
                     x1           x2           x3
          y1    -17.233      -16.594      -2.3172
          y2     3.1321      -6.7352      -11.604


D =
                     u1           u2           u3
          y1      1.158     0.097347      0.98285
          y2    -1.1522    -0.017377     -0.87642


K =
                     y1           y2
          x1 -0.00030967   0.00023064
          x2   0.0014415   -0.0015581
          x3   0.0013428   -0.0019277


x(0) =

          x1           0
          x2           0
          x3           0

Estimated using SUBID3B - Shift invariance approach
Loss function
Sampling interval: 1


p =

   1.0e-05 *

   -0.2832    0.0492    0.0182
    0.0492   -0.5308   -0.5354
    0.0182   -0.5354   -0.8602
```

```
sv =

   1.0e+03 *

    3.2066
    0.6023
    0.2707

>> [yh,fit] = compare(valdata,m,sys); fit % Compare estimated model to
                                          % original system
fit(:,:,1) =

   97.4080   97.4899


fit(:,:,2) =

   92.3844   92.2662
```

See Also

`IDDATA`, `IDSS`, `N4SID`, `BALPEM`, `PEM`.

Bibliography

[CM97] C.T. Chou and J.M. Maciejowski, "System identification using balanced parameterizations", *IEEE Trans. Auto. Contr.*, vol. 42, pp. 956–974, 1997.

[CM98] N.L.C. Chui and J.M. Maciejowski, "Subspace identification — a Markov parameter approach," Technical Report CUED/F-INFENG/TR.337, University of Cambridge, UK, December 1998. Submitted to *IEEE Trans. Auto. Contr.*

[CM99] H. Chen and J.M. Maciejowski, "A new subspace identification method for bilinear systems," Technical Report CUED/F-INFENG/TR.357, University of Cambridge, UK, May 2000. Submitted to *Automatica*.

[CM00a] H. Chen and J.M. Maciejowski, Subspace identification of combined deterministic-stochastic bilinear systems, *Proc. IFAC Symposium on System Identification SYSID 2000*, Santa Barbara, June 2000.

[CM00b] H. Chen and J.M. Maciejowski, An improved subspace identification method for bilinear systems, Proc. IEEE CDC Conference, Sydney, December 2000.

[FDV99] W. Favoreel, B. De Moor and P. Van Overschee, Subspace identification of bilinear systems subject to white inputs, *IEEE Trans. Auto. Contr.*, vol.44, no.6, 1157–1165, 1999.

[Lj99] L. Ljung, *System Identification: Theory for the User (2nd ed.)*, Prentice Hall, 1999.

[Ob87] R.J. Ober, Balanced realizations: Canonical form, parametrization, model reduction, *Int. Journal of Control*, vol.46, no.2, 643–670, 1987.

[Ob91] R.J. Ober, Balanced parametrization of classes of linear systems, *SIAM J. Contr. Optim.*, vol.29, 1251–1287, 1991.

[VODM96] P. Van Overschee and B. De Moor, *Subspace Identification for Linear Systems: Theory, Implementation, Applications*, Kluwer Academic Publishers, 1996.

[VV99] V. Verdult and M. Verhaegen, Subspace-based identification of MIMO bilinear systems, *Proc. European Control Conf.*, Karlsruhe, September 1999.

[Ve02] V. Verdult, *Nonlinear System Identification: A State-Space Approach*, Ph.D. Thesis, University of Twente, The Netherlands, 2002.