Matrix Canonical Structure Toolbox
User’s manual
version 0.7

Department of Computing Science, Umeå University, Sweden

September 13, 2019

Abstract
The Matrix Canonical Structure toolbox is a Matlab toolbox for computing and representing canonical structure information. This user’s manual is a summary of the help-texts for the functions and classes in the toolbox. The manual also includes the StratiGraph Matlab interface functions, which are distributed together with the Matlab plug-in for StratiGraph. For further information and downloads visit https://www.umu.se/en/stratigraph-mcs/, or contact Stefan Johansson, stefanj@cs.umu.se.

Contents
1 Introduction 4
  1.1 Installation ................................................. 4
  1.2 StratiGraph Connection ...................................... 5
  1.3 Compatibility .............................................. 5
  1.4 Contact ..................................................... 5
  1.5 Developers and Scientific Contributors ...................... 5
  1.6 Bibliography ............................................... 6
2 Toolbox summary ............................................. 7
3 Auxiliary functions .......................................... 10
  3.1 MCSFORMAT ................................................ 10
  3.2 MCSGETVALIDBLOCKS ...................................... 10
  3.3 MCSTOLERANCE ............................................. 11
  3.4 MCSEVCLMTH ............................................... 11
  3.5 SEGRE2WEYR ............................................... 12
  3.6 SIZES2SEGRE ............................................... 12
  3.7 SIZES2WEYR ............................................... 12
  3.8 WEYR2SEGRE ............................................... 13
  3.9 WEYR2SIZES ............................................... 13
4 Canonical Structure Objects .................................. 14
  4.1 MCSSTRUCT ................................................ 14
  4.2 CMSTRUCT ................................................ 20
  4.3 MPSTRUCT ................................................ 26
  4.4 MSTRUCT ................................................ 33
  4.5 PSTRUCT ................................................ 38
  4.6 SCMSTRUCT ................................................ 45
  4.7 SPSTRUCT ................................................ 51
9 StartiGraph public interface functions

9.1 SGCLOSE .......................................................... 136
9.2 SGGET ............................................................... 136
9.3 SGGETCONSTRAINTS .............................................. 136
9.4 SGLABELS .......................................................... 136
9.5 SGOPEN ............................................................ 137
9.6 SGSET .............................................................. 137

10 Software License .................................................. 139
1 Introduction

The Matrix Canonical Structure (MCS) Toolbox for Matlab\footnote{Matlab is a registered trademark of The MathWorks, Inc.} provides a framework with data type objects for representing canonical structures and computational functions related to canonical forms.

Current version supports canonical structures of

- matrices (under similarity, congruence, and *congruence),
- matrix pencils $G - sH$ (under equivalence, and symmetric and skew-symmetric pencils under congruence),
- matrix polynomials $P(s) = P_d s^d + \cdots + P_1 + P_0$, where $P_d \neq 0$,
- system pencils (under feedback-injection equivalence)

\[
\begin{bmatrix}
A & B \\
C & D
\end{bmatrix}
- s
\begin{bmatrix}
I & 0 \\
0 & 0
\end{bmatrix}
\]

associated with the state-space system

\[
\begin{aligned}
\dot{x} &= Ax(t) + Bu(t), \\
y &=Cx(t) + Du(t),
\end{aligned}
\]

and particular systems thereof.

The MCS toolbox includes routines for computing the canonical structure information using staircase algorithms. These are based on the GUPTRI (Generalized UPper TRIangular) algorithm [Demmel & Kågström, 1993] for matrix pencils. The GUPTRI form reveals the fine canonical structure information of a matrix pencil. For example, for a linearized model we can compute its canonical structure and then let StratiGraph determine and visualize nearby structures in the closure hierarchy.

The toolbox is based on the prototype Matrix Canonical Structure toolbox described in [Johansson, 2006], however, the majority of the code has been rewritten and many of the functions are modified or have been removed.

1.1 Installation

Download the package from


Unpack and save the code in desired location and add it (and subdirectories) to the Matlab search path.

To avoid problems together with StratiGraph, do not save MCS Toolbox in a path which has blank spaces in it.

For the function gsylve for solving the generalized Sylvester matrix equation, the toolbox includes pre-compiled mex-file for MS Windows. If needed or to run guptri on Linux or macOS, the included source file guptri/private/gsylve.cpp can be compiled as shown below. It requires to have a compatible C++ compiler and Matlab R2018a or higher. Depending on OS, run the appropriate call below in Matlab.
MS Windows:

> mex -llibmwlapack gsylve.cpp

Linux or macOS:

> mex -lmwlapack gsylve.cpp

1.2 StratiGraph Connection

There exist interface functions for communicating with the complementary software StratiGraph. These functions are distributed together with the Matlab plug-in for StratiGraph. For installation and usage see the README-file in the StratiGraph distribution (can be downloaded from https://www.umu.se/en/stratigraph-mcs/). For completeness, the interface functions to StratiGraph are also included in this user’s manual in Section 9.

1.3 Compatibility

For full functionality the toolbox requires Matlab version 9.4 (R2018a) or later. The requirements for individual parts are:

<table>
<thead>
<tr>
<th>Functions</th>
<th>Minimum Matlab version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core</td>
<td>7.6 (R2008a)</td>
</tr>
<tr>
<td>Guptri</td>
<td>9.4 (R2018a)</td>
</tr>
<tr>
<td>StratiGraph interface</td>
<td>8.2 (R2013b)</td>
</tr>
</tbody>
</table>

1.4 Contact

For questions and comments please contact:

Stefan Johansson, stefanj@cs.umu.se

Department of Computing Science
Umeå University
SE - 901 87 Umeå, Sweden

1.5 Developers and Scientific Contributors

MCS Toolbox is developed at the Department of Computing Science, Umeå University (Sweden) by the following people.


Scientific contribution: Bo Kågström, Erik Elmroth, Pedher Johansson, Stefan Johansson, and Andrii Dmytryshyn.

A Special thanks to our collaborators: Alan Edelman (MIT, Massachusetts) and Paul Van Dooren (UCL, Belgium).
1.6 Bibliography


2 Toolbox summary

Matrix Canonical Structure (MCS) Toolbox
Version 0.7

The following canonical structures are available
- mstruct - for matrices under similarity,
- cmstruct - for matrices under congruence,
- scmstruct - for matrices under congruence,
- pstruct - for matrix pencils under equivalence,
- spstruct - for symmetric matrix pencils,
- ssstruct - for skew-symmetric matrix pencils,
- mpstruct - for matrix polynomials,
- ssstruct - for state-space system pencils.

A canonical structure object is created by calling the constructor of the canonical structure class.

Common methods for canonical structure objects:
- codim - Compute the codimension of a canonical structure object.
- size - Total size of the represented structure.
- numblblk - Number of canonical block objects.
- isempty - True for an empty canonical structure object.
- char - Convert a structure object to a string.
- exist - Check if a canonical block of a specified type already exist in the canonical structure object.
- copy - Return a copy of the canonical structure object.
- compare - Compare two canonical structure objects.
- set - Set the canonical structure information (not implemented).
- get - Get the canonical structure information.
- getvalidblocks - Return a list of valid canonical blocks.

Available canonical blocks are:
- fjblock - Jordan block associated with a finite eigenvalue.
- zjblock - Jordan block associated with the zero eigenvalue.
- ijblock - Jordan block associated with the infinite eigenvalue.
- rsblock - Right singular block.
- lsblock - Left singular block.
- gblock - Gamma block.
- sgblock - Gamma block associated with a complex parameter.
- wblock - W block associated with a specified eigenvalue.
- swblock - #W block associated with a specified eigenvalue.
- mblock - Singular symmetric M block.
- smblock - Singular skew-symmetric SM block.
- hbblock - Symmetric H block associated with a finite or unspecified eigenvalue.
- shblock - Skew-symmetric SH block associated with a finite or unspecified eigenvalue.
- kblock - Symmetric K block associated with the infinite eigenvalue.
- skblock - Skew-symmetric K block associated with the infinite eigenvalue.
The two tables below show which blocks are available for each canonical structure.

<table>
<thead>
<tr>
<th>mstruct</th>
<th>cmstruct</th>
<th>scmstruct</th>
</tr>
</thead>
<tbody>
<tr>
<td>fjblock</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>zjblock</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>gbblock</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>sbblock</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>wblock</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>swblock</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pstruct</th>
<th>spstruct</th>
<th>sspstruct</th>
<th>mpstruct</th>
<th>ssstruct</th>
</tr>
</thead>
<tbody>
<tr>
<td>fjblock</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>zjblock</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ijblock</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rsblock</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lsblock</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mblock</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sblock</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hblock</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>shblock</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>kblock</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>skbblock</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Available canonical forms are:
- jcf/jnf - Jordan Canonical Form
- ccf - Congruent Canonical Form
- kcf - Kronecker Canonical Form
- bcf - (generalized) Brunovsky Canonical Form

The table below shows which form(s) is(are) available for each canonical structure.

<table>
<thead>
<tr>
<th>jcf</th>
<th>ccf</th>
<th>kcf</th>
<th>bcf</th>
</tr>
</thead>
<tbody>
<tr>
<td>mstruct</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmstruct</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>scmstruct</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pstruct</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>spstruct</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sspstruct</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mpstruct</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ssstruct</td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Computational functions
- pguptri - Compute the canonical structure information of a matrix pencil.
- pcluster - Compute and cluster generalized eigenvalues of a matrix pencil.
- pggallery - Test matrix pencils for staircase computation and bounds.

[x] below can be one of the prefixes:
- m - matrices under similarity,
- cm - matrices under congruence,
scm - matrices under congruence,
p - matrix pencils under equivalence,
sp - symmetric matrix pencils,
ssp - skew-symmetric matrix pencils,
s2 - controllability or observability system pencils.

[x]codim - Computes the codimension.
[x]tanspace - Computes the tangent space.

Notation conversion
segre2weyr - Weyr from Segre characteristics.
weyr2segre - Segre from weyr characteristics.
weyr2sizes - Block sizes from weyr characteristics.
sizes2weyr - Weyr characteristics from block sizes.
sizes2segre - Segre characteristics from block sizes.

Auxiliary functions
mcstolerance - Set tolerance parameters used by MCS Toolbox.
mcsformat - Set format to display structure and block objects.
mcsgetvalidblocks - Return a list of available canonical blocks.
mcevclmth - Set default method used to cluster eigenvalues in MCS Toolbox.
3 Auxiliary functions

3.1 MCSFORMAT

Set format to display structure and block objects.

**mcsformat** with no argument sets the output format to display canonical structure and block objects in MCS Toolbox to the default **block** format.

The output format can be changed to different formats as follows:

- **mcsformat block**  Canonical block format.
- **mcsformat segre**  Segre-type characteristics, where the sizes of the canonical blocks are in non-increasing order.
- **mcsformat segrec**  A compact variant of **segre** where multiple blocks of the same size is written as n*,... , where n is the number of blocks.
- **mcsformat weyr**  Weyr-type characteristics representation of the canonical blocks, where the integer in position k is the number of blocks greater than or equal to k for regular blocks or k-1 for singular blocks.
- **mcsformat sizes**  Same as **segre** but where the sizes of the canonical blocks may be unordered (displayed in the order they where created).

**mcsformat** can also be called as a function:

```
mcsformat(Fmt)
```

where Fmt is the output format as a string. For example

```
mcsformat('segre')
```

**mcsformat(Fmt, Persistent)** also set if the used format shall be persistent between Matlab sessions (Persistent = true) or not (Persistent = false). Default is false.

Fmt = **mcsformat** returns the current output format as a string.

```
[Fmt,IsPersistent] = mcsformat also returns if the used format is persistent between Matlab sessions or not.
```

3.2 MCSGETVALIDBLOCKS

Return a list of available canonical blocks.

**mcsgetvalidblocks** returns a list of all available canonical block objects.

**mcsgetvalidblocks(StructHandle)** returns the list of valid canonical block objects for the specified canonical structure **StructHandle**, where **StructHandle** can be a canonical structure object, the name of the canonical structure class as a string, or the class given as a **meta.class** object.

See also **mcsstruct/getvalidblocks, meta.class**.
3.3 MCSTOLERANCE

Default tolerance parameters used by MCS Toolbox.

\[ T = \text{mcstolerance} \]
returns the two-element row vector \( T = [\text{Tol}, \text{Gap}] \) containing the default tolerance parameters used for the functions in the toolbox. The default values are \( \text{Tol} = 1\text{e}-12 \) and \( \text{Gap} = 1000 \), if they have not been changed (see below). The tolerances can also usually be overridden by giving them as parameters to the individual routines.

\[ [\text{Tol},\text{Gap}] = \text{mcstolerance} \]
returns the tolerance parameters as separate output variables.

\[ [\text{Tol},\text{Gap},\text{IsPersistent}] = \text{mcstolerance} \]
also returns if the used tolerance parameters are persistent between sessions or not.

\text{mcstolerance}(\text{Tol},\text{Gap}) \] The positive parameters Tol and Gap is set as default tolerance parameters in all toolbox routines. Tol should reflect the relative uncertainty in the data and should be at least about \( \text{eps} \) (nominally between \( 1\text{e}-8 \) and \( 1\text{e}-12 \)). Gap determines the ratio between the smallest preserved singular value and the largest singular value imposed to zero. Gap should be at least 1 (nominally 1000).

\text{mcstolerance}(\text{Tol},\text{Gap},\text{Persistent}) \] The values of Tol and Gap can be made persistent between Matlab sessions by setting the parameter Persistent to non-zero (true). Default is false.

If the value of Gap is set to zero or if the value of Tol is less then eps a warning is issued. This warning message can be suppressed by typing

\text{warning('off','MCS:mcstolerance:ZeroTolerance')}\]

See also \text{mcsevclmth}.

3.4 MCSEVCLMTH

Default method used to cluster eigenvalues in MCS Toolbox.

\[ \text{CLMethod} = \text{mcsevclmth} \]
returns the name of the used method to cluster eigenvalues in the toolbox as a string. If no default method is previously set, the 'var' method is returned (see below). The default method can usually be overridden by a parameter to the individual functions.

\[ [\text{CLMethod},\text{IsPersistant}] = \text{mcsevclmth} \]
also returns if the used method is persistent between sessions or not.

\text{mcsevclmth}(\text{Method}) \] sets the default method to cluster eigenvalues.

Valid methods are the following:

- 'norm' - Use an "over-simple" method where the tolerance parameter is used to separate different clusters.

See also \text{private/cenorm}.
'var', 'min', 'max', 'avg', 'mean'
   - Use a hierarchical clustering algorithm where the
     clustering is determined by a distance function.
     See also private/cehierarchy.
'gersh'
   - Use a method based on Gershgorin circles to cluster
     the eigenvalues.
     See also private/mcegershgorin, private/pcegershgorin.

mcsevclmth(Method,Persistent) also set if the used cluster method shall be
persistent between sessions (Persistent = true) or not (Persistent = false).
Default is false.
See also pguptri, mguptri, pcluster, mcluster.

3.5 SEGRE2WEYR

Weyr from Segre characteristics.

segre2weyr(Segre,Type) converts an array with the block sizes given in
Segre characteristics to an array with their sizes presented in Weyr
characteristics. The parameter Type gives the type of the blocks and
can be:
   'regular' for regular (Jordan-type) blocks,
   'singular' for singular blocks.

See also weyr2sizes, weyr2segre, sizes2weyr, sizes2segre.

3.6 SIZES2SEGRE

Segre characteristics from block sizes.

sizes2segre(Sizes) converts an array with the block sizes to an array
with their sizes represented in Segre characteristics, i.e., the block
sizes sorted in a non-increasing order.

See also weyr2sizes, weyr2segre, segre2weyr, sizes2weyr.

3.7 SIZES2WEYR

Weyr characteristics from block sizes.

sizes2weyr(Sizes,Type) converts an array with the block sizes to an
array with their sizes represented in Weyr characteristics. The
parameter Type gives the type of the blocks and can be:
   'regular' for regular (Jordan-type) blocks,
   'singular' for singular blocks.

See also weyr2sizes, weyr2segre, segre2weyr, sizes2segre.
3.8 WEYR2SEGRE

Segre from Weyr characteristics.

`weyr2segre(Weyr,Type)` converts an array with the block sizes given in Weyr characteristics to an array with the blocks' sizes in Segre characteristics, i.e., the sizes of the blocks sorted in non-increasing order. The parameter Type gives the type of the blocks and can be:
- 'regular' for regular (Jordan-type) blocks,
- 'singular' for singular blocks.

See also `weyr2sizes`, `segre2weyr`, `sizes2weyr`, `sizes2segre`.

3.9 WEYR2SIZES

Block sizes from Weyr characteristics.

`weyr2sizes(Weyr,Type)` converts an array with the block sizes given in Weyr characteristics to an array with the blocks' sizes. The parameter Type gives the type of the blocks and can be:
- 'regular' for regular (Jordan-type) blocks,
- 'singular' for singular blocks.

See also `weyr2segre`, `segre2weyr`, `sizes2weyr`, `sizes2segre`. 
4 Canonical Structure Objects

4.1 MCSSTRUCT

Abstract class for generic canonical structures.

`mcsstruct` provides templates and generic methods for canonical structure objects.

```plaintext
StructObj = mcsstruct('BlockName1',StructInt1,'BlockName2',StructInt2,...,'Notation',Notation)
```
creates a new structure object with the specified blocks.

`mcsstruct` Methods:
- `codim` - Compute the codimension of a canonical structure object.
- `size` - Total size of the represented structure.
- `numblk` - Number of canonical block objects.
- `isempty` - True for an empty canonical structure object.
- `char` - Convert a structure object to a string.
- `exist` - Check if a canonical block of a specified type already exist in the canonical structure object.
- `copy` - Return a copy of the canonical structure object.
- `compare` - Compare two canonical structure objects.
- `set` - Set the canonical structure information (not implemented).
- `get` - Get the canonical structure information.
- `getvalidblocks` - Return a list of valid canonical blocks.
- `getsgconstraints` - Return a list of available StratiGraph constraints.

`mcsstruct` Operators:
- `==, ~=` (eq, ne) - Check if two structure objects are equal.
- `()`, `{}` (subsref) - Index reference for canonical structure objects.

See also `mcsblock`.

Reference page in Doc Center

doc mcsstruct

Class methods

getsgconstraints Return a list of available StratiGraph constraints.

getsgconstraints returns a list of all available constraints for the corresponding setup in StratiGraph. The returned list is a cell-array of strings.

See also `sggetconstraints`.

getvalidblocks Return a list of valid canonical blocks.

getvalidblocks(StructObj) returns a list of all valid canonical
block objects for StructObj.

See also mcsgetvalidblocks.

**subsref** Subscripted reference for canonical structure objects.

The canonical blocks in a structure object StructObj can be accessed by the subscripts 'StructObj(k)' and 'StructObj{k}', which have different meanings. The subscript '()' returns a new canonical structure object with the same canonical structure as the specified block(s). Consequently, 'StructObj(:)' returns a copy of the structure object StructObj. The subscript '{}' returns the specified block objects in separate outputs. Note that for '{} the returned block objects refers to the same objects as in the original canonical structure (for an example see below).

Examples (exemplified with matrix pencils):

```matlab
>> pstr = pstruct([0 1], 0, 1, 1e-4i)
>> pstr([1 3])
ans =
    R = (1 0) (1x3)
    J(0+0.0001i) = (1) (1x1)
```

Class methods may be applied directly to the subobject (by chaining call):

```matlab
>> [G,H] = pstr([1 3]).kcf
G =
    0+0i  0+0i  1+0i  0+0i
    0+0i  0+0i  0+0i  0+0.0001i
H =
    0  1  0  0
    0  0  0  1
```

```matlab
>> [b1,b2] = pstr{1:2}
b1 =
    R = (1 0) (1x3)
b2 =
    L = (0) (1x0)
```

Changing the block object b1 or b2 will also change the canonical structure of pstr:

```matlab
>> b1.set('StructInt',[2 1]);
>> pstr
pstr =
    R = (2 1) (3x5)
    L = (0) (1x0)
    J(0+0.0001i) = (1) (1x1)
```
Chaining call with '{}' is only possible if one single subscript is given, for example:

```plaintext
>> [G,H] = pstr{1}.kcf
G =
  0  0  1
H =
  0  1  0
```

See also `get`.

**get** Get the canonical structure information.

```plaintext
V = get(StructObj,'PropertyName') returns the value of the specified property for the canonical structure object StructObj.
V = get(StructObj,'PropertyName1', 'PropertyName2', ...) may be used to refine which property to return.

The property name can be in any order (described below):
1) Any valid canonical block name.
2) 'StructInt', 'Parameter', or 'Eigenvalue'.
3) Any valid notation.
4) 'Unroll'
5) 'Object' (properties of type 2, 3, and 4 are ignored).

By specifying any valid block name `get` returns the structure integer partition and any parameter associated with the canonical blocks of type `PropertyName`. Multiple block names can be specified. If no block names are given, properties for all existing blocks are returned. `PropertyName` is a string and can, e.g., be one of:
- 'sblock' - Singular blocks (both right and left)
- 'rsblock' - Right singular blocks
- 'lsblock' - Left singular blocks
- 'jblock' - Jordan blocks (both finite and infinite eigenvalues)
- 'fjblock' - Jordan blocks associated with finite eigenvalues

For a complete list of all valid canonical blocks for specific structure use the method `getvalidblocks`, see also `mcsgetvalidblocks`. Calling `mcsgetvalidblocks` without any arguments will return all implemented canonical block classes.

The returned `V` is a cell-array, where each element in `V` is a vector with the sizes of corresponding canonical block. For blocks associated with a parameter (e.g. an eigenvalue) the size vector and the parameter are combined in a cell-array tuple, one for each parameter. If more than one block name is given, then all structure integer partitions corresponding to the same block class are gathered in a cell-array. `V` will then have the same
length as number of specified blocks.

Example 1:
>> pstr = pstruct([3 1],[2 0],[[2 1] [3]],[3 0]);
>> V = pstr.get('lsblock','fjblock');

Then V = {[2 0], {{[2 1], 3}, {[3], 0}}}, i.e., the matrix pencil object pstr has two left singular blocks of sizes 3x2 and 1x0, two Jordan blocks of sizes 2x2 and 1x1 both associated with the eigenvalue 3, and one Jordan block of size 3x3 associated with the zero eigenvalue.

[V,Param] = get(StructObj,'Blockname') returns the structure integer partitions in the cell-array V and the associated parameters in the vector Param, where V(k) has the parameter Param(k).

[V,Param] = get(StructObj,'Blockname1','Blockname2',...) returns the structure integer partitions and the parameters in two separate cell-arrays. If the canonical block(s) in V(k) has/have an associated parameter, then the associated parameter of the blocks V(k){j} is Param(k){j}. If no parameter exist for block k, Param(k) is empty.

If StructObj is a vector of length M of structure objects, then V (and Param) will be M-by-X cell-array of values (some may be empty), where X is the number of requested canonical blocks in the structure.

2) It is also possible to specify which information to return. 'StructInt' only returns the structure integer partition, and 'Parameter' or 'Eigenvalue' the associated parameter (e.g. eigenvalue).

3) The notation of the structure integer partition can be specified by setting any property name to one of:
   - 'sizes' Sizes may be unordered. (Default)
   - 'segre' Sizes are ordered in a non-increasing order.
   - 'weyr' Weyr characteristics.

4) By passing 'Unroll' as a property, multiple blocks of same class are not encapsulated in a struct. Instead they are appended to the cell-array V as separate vectors. This option is only available if one of 'StructInt' or 'Parameter'/'Eigenvalue' is specified, not both.

Example 2 (compare with Example 1):
>> pstr = pstruct([3 1],[2 0],[[2 1] [3]],[3 0]);
>> V = pstr.get('lsblock','fjblock','StructInt','Unroll');

Returns the cell-array \( V = \{ [3 \ 1], [2 \ 1], [3] \} \).

5) By passing 'Object' as a property, the associated canonical block objects will be returned in the cell-array \( V \). In this case, any parameter specifying notation and/or which information to return is neglected.

See also \texttt{set}, \texttt{mcsblock}, \texttt{getvalidblocks}, \texttt{subsref}.

\textbf{set} Set the canonical structure information (not implemented).

\texttt{set} is not implemented for canonical structure objects. Instead use the following procedure to change any canonical structure information.

1. Obtain the canonical block object(s) which is/are going to be modified. Example (where \texttt{pstr} is a \texttt{pstruct} object):

   \[
   \texttt{blkobj} = \texttt{pstr.get('lsblock','object');}
   \]

2. Modify the block object by using \texttt{set()}. Example:

   \[
   \texttt{blkobj{1}.set('StructInt',[3 \ 1]);}
   \]

3. Done!

See also \texttt{get}, \texttt{mcsblock/set}

\textbf{copy} Return a copy of the canonical structure object.

\texttt{StructObjNew = copy(StructObj)} returns a copy of the canonical structure object \texttt{StructObj}.

\textbf{exist} Check if a canonical block of a specified type already exists.

\[
\texttt{B = exist(StructObj,BlockClass)} \quad \text{returns} \quad 1 \quad \text{(true)} \quad \text{if} \quad \text{a canonical block object of type BlockClass already exist in the structure object} \quad \texttt{StructObj} \quad \text{and} \quad 0 \quad \text{(false)} \quad \text{otherwise}. \quad \texttt{BlockClass} \quad \text{can either be as a string or a canonical block object.}
\]

\[
\texttt{B = exist(StructObj,BlockClass,Param)} \quad \text{is used when the canonical block object also has a parameter, e.g., an eigenvalue.}
\]

\textbf{disp} Display a canonical structure object.

\texttt{disp(StructObj)} is called for the structure object \texttt{StructObj} when the semicolon is not used to terminate a statement. Returns the canonical form represented as a string.

\textbf{char} Convert a structure object to a string.
S = char(StructObj) returns the canonical structure information of the structure object StructObj as a string in canonical block notation.

See also mcsblock/char.

**compare** Compare two canonical structure objects.

C = compare(S1,S2) compares the two canonical structure objects S1 and S2 of the same class and returns a vector C with the same length as number of existing canonical blocks. The two canonical structure objects must have the same number of canonical blocks.

The comparison is done by comparing the sizes of the canonical blocks for each pair of blocks (B1,B2) of the same class from S1 and S2, respectively. For each type of canonical block; first the largest block from each block object are compared then the second largest until one is larger than the other or no more blocks exist. The corresponding element in C is 0 if the blocks B1 and B2 are equal, 1 if B1 > B2, and -1 if B1 < B2. Consequently, if all elements in C are 0 then S1 and S2 have the same block structure. Any parameters are not compared, but have an impact on which block objects is compared to which block object. A block in S1 which does not exist in S2 always returns 1.

C = compare(S1,S2,'weyr') uses the Weyr characteristics resulting in that the comparison is done by comparing the number of blocks. First the number of all blocks are compared then the number of second smallest and larger blocks until one quantity is larger than the other.

[C2, D] = compare(S1,S2,Tol) or [C2, D] = compare(S1,S2,'weyr',Tol) returns the 2-column matrix C2 = [C,P], where C is the result from above and the P(k) is the result from comparing the corresponding parameter (eigenvalue) of the block C(k). It uses the tolerance Tol for comparing the parameters. P(k) = 0 if abs(p1-p2) <= Tol for all parameters p1 and p2 from the same block class in S1 and S2, respectively, otherwise P returns 1. For blocks with no parameter, Tol is ignored and the corresponding element in P is 0. Consequently, if both S and P are 0 then the two structures are equal with respect to Tol. The optional D returns the difference abs(p1-p2) between the two parameters, or NaN if the blocks have no parameter.

C = compare(S1,S2,'size') only compares the total sizes (m*n) of structure objects. Returns 0 if equal, 1 if S1 > S2, and -1 if S1 < S2.

See also eq.

**isempty** True for empty canonical structure.

isempty(StructureObj) returns 1 if StructureObj is an empty
canonical structure object and 0 otherwise. An empty structure object has no canonical blocks or all are empty.

**numblk** Number of canonical block objects.

\[ N = \text{numblk}(\text{StructObj}) \]

returns the number of canonical block objects of different types in the structure object StructObj. Canonical blocks with equal parameters (eigenvalues) are of the same type, while blocks with different parameters are not.

See also **size**, **mcsblock**, **numblk**.

**size** Total size of the represented structure.

\[ D = \text{size}(\text{StructObj}) \]

returns a two-element row vector \( D = [M, N] \) containing the number of rows \( M \) and columns \( N \) of the structure object in matrix or matrix pencil form, i.e., the sum of the sizes of all the canonical blocks.

\[ [M,N] = \text{size}(\text{StructObj}) \]

returns the number of rows and columns in separate output variables.

size(StructObj,Dim) returns the length of the dimension specified by the scalar Dim. For example, size(StructObj,1) returns the number of rows.

**eq** (==) Equal relation between two structure objects.

\[ \text{StructObj1} == \text{StructObj2} \]

checks if the two canonical structure objects are equal.

See also **ne**, **compare**.

**ne** (~=) Not equal relation between two structure objects.

\[ \text{StructObj1} \sim= \text{StructObj2} \]

checks if the two canonical structure objects are not equal.

See also **eq**, **compare**.

### 4.2 CMSTRUCT

Create a congruence matrix structure object.

**cmstruct** creates a canonical structure object representing a matrix \( A \) in its canonical form under congruence.

A **cmstruct** object can consist of the following three canonical blocks:

- **gblock** - Gamma block defined as:

\[ \begin{bmatrix} 0 & \cdots \\ \cdots & 0 \end{bmatrix} \]
\[
\gamma_n :=\begin{pmatrix} 1 & 1 \\ -1 & -1 \\ 1 & 1 & 0 \end{pmatrix}
\]

\text{wb}block - W block associated with a specified and admissible eigenvalue \( \mu \), defined as:
\[
W_n := \begin{pmatrix} 0 & I_n \\ J_n(\mu) & 0 \end{pmatrix}
\]
where \( \mu \sim \{0, (-1)^{(n+1)}\} \) and \( J_n(\mu) \) is an \( n \times n \) Jordan block associated with the eigenvalue \( \mu \).

\text{zj}block - Jordan block associated with the zero eigenvalue, defined as:
\[
J_n(0) := \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}
\]

StructObj = \text{cmstruct}(\text{GBlocks}, \text{WBlocksv}, \text{Eigv}) returns a new canonical structure object \text{StructObj} representing a matrix \( A \) under congruence. The canonical structure information can be specified using one of the three forms '
Size vector form
-----------------
StructObj = \text{cmstruct}(\text{GBlocks}, \text{WBlocksv}, \text{Eigv}) returns a new canonical structure object \text{StructObj} representing a matrix \( A \) under congruence. The parameter \text{GBlocks} defines the Gamma blocks of the structure and is given as row-vectors with the sizes of the blocks. The parameter \text{WBlocksv} defines the W blocks associated with the finite Jordan blocks of the structure and must be a cell-array of row-vectors where each vector contains the sizes of blocks associated with the same eigenvalue. \text{Eigv} is a row-vector with the values of each associated eigenvalue corresponding to the row-vectors in \text{WBlocksv}, where the eigenvalues must be non-zero complex scalars not equal to \((-1)^{(\text{size of the block}/2 + 1)}\) for all blocks associated with the same eigenvalue.

StructObj = \text{cmstruct}(\text{GBlocks}, \text{WBlocksv}, \text{Eigv}, \text{ZJBlocks}) also sets the sizes of the Jordan blocks with an associated zero eigenvalue.

Property-value form
---------------------
StructObj = \text{cmstruct}('BlockName1', \text{StructInt1}, 'BlockName2', \text{StructInt2}, ... ) specifies the block types \text{BlockName} and the sizes \text{StructInt} in property-value form. \text{BlockName} is a string specifying the canonical block to be
created, and can be one of:
'gblock' - Gamma blocks
'wblock' - W blocks associated with finite eigenvalues
'zjblock' - Jordan blocks associated with the zero eigenvalue.

W blocks associated with a specified finite eigenvalue are given as a cell-array tuple in the form {StructInt, Eigenvalue}.

Example:
>> cmstr = cmstruct('wblock',[3 2], 'wblock', [1], 4)
creates a structure object with one W block of size 6x6 and one of size 4x4 both with the associated eigenvalue 2, and one W block of size 2x2 with associated the eigenvalue 4.

cmstruct('BlockName1',StructInt1,...,'Notation',Notation) also specifies the notation used for StructInt. Valid notations are:
'segre' Sizes are ordered in a non-increasing order.
'weyr' Weyr characteristics.
'sizes' Sizes may be unordered. (default)

Object form
------------
StructObj = cmstruct(BlockObj1,BlockObj2,...) creates a structure object from the listed canonical block objects. The block objects must be valid blocks for the structure.

Examples:
>> cmstr = cmstruct([], [3 1], 4, [1])
returns a matrix structure with one W block of size 6x6 and one of size 2x2 both with the associated eigenvalue 4, and one Jordan block of size 1x1 with the zero eigenvalue.

Alternatively, the same structure can be created with
>> cmstr = cmstruct('zjblock', 1, 'wblock', [[3 1], 4])

>> cmstr = cmstruct([], [[2], [1]], [1 3], [2])
returns a matrix of the following form:
| W2(1) 0 0 |
| 0 W1(3) 0 |
| 0 0 J2(0) |

Subscripting
------------
To access the canonical blocks in the structure object it is possible to use subscripts. See subsref for detail and examples. If more control is needed of what should be retrieved, use the method get instead.

The class cmstruct provides the following methods for extracting information and modifying the canonical structure object.
cmstruct Methods:
- **codim** - Compute the codimension of a matrix under congruence.
- **ccf** - Return the matrix in the congruence canonical form.
- **size** - Total size of the represented structure.
- **numblk** - Number of canonical block objects.
- **isempty** - True for empty canonical structure.
- **char** - Convert a structure object to a string.
- **exist** - Check if a canonical block of a specified type already exist.
- **copy** - Return a copy of the structure object.
- **compare** - Compare two canonical structure objects.
- **set** - Set the canonical structure information (not implemented).
- **get** - Get the canonical structure information.
- **getvalidblocks** - Return a list of available canonical blocks.

cmstruct Operators:
- **==, ~= (eq, ne)** - Check if two structure objects are equal.
- **( ), {} (subsref)** - Index reference for canonical structure objects.

See also **gblock, wbloc, zjblock, scmstruct, mstruct**.

Reference page in Doc Center
doc cmstruct

Class methods

codim Compute the codimension of a matrix under congruence.

codim(StructObj) determines the codimension of the congruence orbit of the matrix structure object StructObj. The codimension is determined with respect to the represented canonical structure not the tangent space.

codim(StructObj,Strata) where the optional argument Strata can be
'orbit' Determines the codimension of the orbit.
Assumes that all eigenvalues are specified.
(default)
'bundle' Not implemented!

See also **cmcodim**.

cmstruct Create a congruence matrix structure object.

cmstruct creates a canonical structure object representing a matrix A in its canonical form under congruence.

A cmstruct object can consist of the following three canonical blocks:
- **gblock** - Gamma block defined as:
  | 0   . . |
  |     . |
Gamma_n := | 1 1 | n-by-n
| -1 -1 |
| 1 1 0 |

wblock - W block associated with a specified and admissible eigenvalue \( \mu \), defined as:

\[
W_n := \begin{bmatrix}
0 & I_n \\
J_n(\mu) & 0
\end{bmatrix} \quad 2n-by-2n
\]

where \( \mu \sim \{0, (-1)^{(n+1)}\} \) and \( J_n(\mu) \) is an \( n \)-by-\( n \) Jordan block associated with the eigenvalue \( \mu \).

zjblock - Jordan block associated with the zero eigenvalue, defined as:

\[
J_n(0) := \begin{bmatrix}
0 & 1 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0
\end{bmatrix} \quad n-by-n
\]

StructObj = \textit{cmstruct}() returns an empty canonical structure object StructObj representing a matrix \( A \) under congruence. The canonical structure information can be specified using one of the three forms 'Size vector form', Property-value form', or 'Object form', which are explained below.

Size vector form
-----------------
StructObj = \textit{cmstruct}(GBlocks,WBlocksv,Eigv) returns a new canonical structure object StructObj representing a matrix \( A \) under congruence. The parameter GBlocks defines the Gamma blocks of the structure and is given as row-vectors with the sizes of the blocks. The parameter WBlocksv defines the W blocks associated with the finite Jordan blocks of the structure and must be a cell-array of row-vectors where each vector contains the sizes of blocks associated with the same eigenvalue. Eigv is a row-vector with the values of each associated eigenvalue. Eigv is a row-vector with the values of each associated eigenvalue corresponding to the row-vectors in WBlocksv, where the eigenvalues must be non-zero complex scalars not equal to \((-1)^{\text{size of the block}/2 + 1}\) for all blocks associated with the same eigenvalue.

StructObj = \textit{cmstruct}(GBlocks,WBlocksv,Eigv,ZJBlocks) also sets the sizes of the Jordan blocks with an associated zero eigenvalue.

Property-value form
---------------------
StructObj = \textit{cmstruct}(BlockName1,StructInt1,BlockName2,StructInt2,...) specifies the block types BlockName and the sizes StructInt in property-value form. BlockName is a string specifying the canonical block to be created, and can be one of:
'gblock' - Gamma blocks
'wblock' - W blocks associated with finite eigenvalues
'zjblock' - Jordan blocks associated with the zero eigenvalue.

W blocks associated with a specified finite eigenvalue are given as a
cell-array tuple in the form (StructInt Eigenvalue).

Example:
>> cmstr = cmstruct('wblock',{[3 2],2},'wblock',{[1],4})
creates a structure object with one W block of size 6x6 and
one of size 4x4 both with the associated eigenvalue 2, and one
W block of size 2x2 with associated the eigenvalue 4.

cmstruct('BlockName1',StructInt1,...,'Notation',Notation) also
specifies the notation used for StructInt. Valid notations are:
'segre' Sizes are ordered in a non-increasing order.
'weyr' Weyr characteristics.
'sizes' Sizes may be unordered. (default)

Object form
-----------
StructObj = cmstruct(BlockObj1,BlockObj2,...) creates a structure
object from the listed canonical block objects. The block objects must
be valid blocks for the structure.

Examples:
>> cmstr = cmstruct([],{[3 1],4},{[1]})
returns a matrix structure with one W block of size 6x6 and
one of size 2x2 both with the associated eigenvalue 4, and one
Jordan block of size 1x1 with the zero eigenvalue.

Alternatively, the same structure can be created with
>> cmstr = cmstruct('zjblock',1,'wblock',{[3 1],4})

>> cmstr = cmstruct([], {[2] [1]}, [1 3], [2])
returns a matrix of the following form:
\[
\begin{bmatrix}
W2(1) & 0 & 0 \\
0 & W1(3) & 0 \\
0 & 0 & J2(0)
\end{bmatrix}
\]

Subscripting
------------
To access the canonical blocks in the structure object it is possible
to use subscript. See subsref for detail and examples. If more control
is needed of what should be retrieved, use the method get instead.

The class cmstruct provides the following methods for extracting
information and modifying the canonical structure object.

cmstruct Methods:
4 CANONICAL STRUCTURE OBJECTS

- **codim** - Compute the codimension of a matrix under congruence.
- **ccf** - Return the matrix in the congruence canonical form.
- **size** - Total size of the represented structure.
- **numblk** - Number of canonical block objects.
- **isempty** - True for empty canonical structure.
- **char** - Convert a structure object to a string.
- **exist** - Check if a canonical block of a specified type already exist.
- **copy** - Return a copy of the structure object.
- **compare** - Compare two canonical structure objects.
- **set** - Set the canonical structure information (not implemented).
- **get** - Get the canonical structure information.
- **getvalidblocks** - Return a list of available canonical blocks.

**cmstruct** Operators:
- **==, ~==** (eq, ne) - Check if two structure objects are equal.
- **(), {}** (subsref) - Index reference for canonical structure objects.

See also **gblock, wblock, zjblock, scmstruct, mstruct**.

### 4.3 MPSTRUCT

Create a matrix polynomial structure object.

**mpstruct** creates a canonical structure object representing a matrix polynomial \( P(s) = P_d s^d + \ldots + P_1 s + P_0 \), where \( P_d \approx 0 \). The invariants are represented as Kronecker canonical blocks of a matrix pencil with the same invariants as \( P(s) \).

An **mpstruct** object can consist of the following canonical blocks:

- **rsblock** - Right singular block defined as:
  
  \[
  L_n := \begin{bmatrix}
  0 & 1 & 0 \\
  \vdots & s & \vdots \\
  0 & 0 & 1
  \end{bmatrix}
  \]
  
  n-by-(n+1)

- **lsblock** - Left singular block defined as:
  
  \[
  L_n^T := \begin{bmatrix}
  0 & 0 \\
  1 & -s & 0 \\
  \vdots & \vdots & \vdots \\
  0 & 0 & 1
  \end{bmatrix}
  \]
  
  (n+1)-by-n

- **fjblock** - Jordan block associated with a finite eigenvalue \( \mu \), defined as:
  
  \[
  J_n(\mu) - s*I_n := \begin{bmatrix}
  \mu & 1 & 0 \\
  \vdots & \mu & \vdots \\
  0 & \mu
  \end{bmatrix}
  \]
  
  n-by-n

- **zjblock** - Jordan block associated with the zero eigenvalue (\( \mu=0 \)).

- **ijblock** - Jordan block associated with the infinite eigenvalue.
defined as:
\[
N_n := \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
- s & 0 & 0 \\
0 & 1 & 0
\end{bmatrix}
\]
n-by-n

StructObj = mpstruct() returns an empty canonical structure object
StructObj representing a matrix polynomial
\[ P(s) = P_d s^d + \ldots + P_1 s + P_0, \text{ where } P_d \sim 0, \]
of degree d. To follow the declarations of the other canonical
structure objects, the invariants of \( P(s) \) are represented in the form
of canonical blocks associated with the Kronecker canonical form of a
matrix pencil with the same invariants as \( P(s) \). The degree d ensures
that the matrix polynomial is unique. The canonical structure
information can be specified using one of the three forms 'Size vector
form', Property-value form', or 'Object form', which are explained
below.

Size vector form
----------------
StructObj = mpstruct(d,RSBlocks,LSBlocks,FJBlocksv) returns a new
canonical structure object StructObj representing a matrix polynomial.
The parameters RSBlocks and LSBlocks define the right and left singular
blocks (associated with the column and row minimal indices),
respectively, of the structure and are given as row-vectors with the
sizes of the blocks. The parameter FJBlocksv defines the Jordan blocks
(associated with the finite elementary divisors) of the structure and
must be either a row-vector with the sizes of Jordan blocks, all
associated with the same eigenvalue, or a cell-array of row-vectors
where each vector contains the sizes of blocks associated with the same
eigenvalue. By default the associated eigenvalues are set to
unspecified (NaN).

StructObj = mpstruct(d,RSBlocks,LSBlocks,FJBlocksv,Eigv) also sets the
eigenvalues, where Eigv is a row-vector with the values of each
associated eigenvalue corresponding to the row-vectors in FJBlocksv.
Jordan blocks with an unspecified associated eigenvalue are defined by
setting the corresponding eigenvalue in Eigv to NaN.

StructObj = mpstruct(d,RSBlocks,LSBlocks,FJBlocksv,Eigv,IJBlocks) also
sets the sizes of the Jordan blocks with an associated infinite
eigenvalue (associated with the infinite elementary divisors).

Property-value form
---------------------
StructObj =
mpstruct(d,'BlockName1',StructInt1,'BlockName2',StructInt2,...)
specifies the block types BlockName and the sizes StructInt in
property-value form. BlockName is a string specifying the canonical
block to be created, and can be one of:
'rsblock' - Right singular blocks
'lsblock' - Left singular blocks
'fjblock' - Jordan blocks associated with finite eigenvalues
'zjblock' - Jordan blocks associated with the zero eigenvalue
'ijblock' - Jordan blocks associated with the infinite eigenvalue

Jordan blocks associated with a specified finite eigenvalue are given as a cell-array tuple in the form {StructInt Eigenvalue}.

Example:
>> mpstr = mpstruct(3,'fjblock',[[3 2],0],'fjblock',[[1], -3])
creates a matrix polynomial structure object of degree 3 with one Jordan block of size 3x3 and one of size 2x2 both with eigenvalue 0, and one Jordan block of size 1x1 with eigenvalue -3.

mpstruct(d,'BlockName1',StructInt1,...,'Notation',Notation) also specifies the notation used for StructInt. Valid notations are:
'segre' Sizes are ordered in a non-increasing order.
'weyr' Weyr characteristics.
'sizes' Sizes may be unordered. (default)

Object form
-------------
StructObj = mpstruct(d,BlockObj1,BlockObj2,...) creates a structure object from the listed canonical block objects. The block objects must be valid blocks for the structure.

Examples:
>> mpstr = mpstruct([], [2], [3 1], 4, [1])
returns a matrix polynomial structure with one left singular block of size 3x2, one Jordan block of size 3x3 and one of size 1x1 both with eigenvalue 4, and one Jordan block of size 1x1 with an infinite eigenvalue.

Alternatively, the same structure can be created with
>> mpstr = mpstruct('lsblock',2,'ijblock',1,'fjblock',[[3 1],4])

>> mpstr = mpstruct([], [], [2] [1]), [1 3], [2])
returns a matrix polynomial of the following form:
| J2(1) 0 0 |
| 0 J1(3) 0 |
| 0 0 N2 |

Subscripting
-------------
To access the canonical blocks in the structure object it is possible to use subscripts. See subsref for detail and examples. If more control is needed of what should be retrieved, use the method get instead.

The class mpstruct provides the following methods for extracting
information and modifying the canonical structure object.

**mpstruct** Methods:
- **codim** - Compute the codimension of a matrix polynomial.
- **size** - Total size of the represented structure.
- **numblk** - Number of canonical block objects.
- **rank** - Compute the normal rank.
- **hasfullrank** - True for canonical structure with full normal rank.
- **isempty** - True for empty canonical structure.
- **char** - Convert a structure object to a string.
- **exist** - Check if a canonical block of a specified type already exist.
- **copy** - Return a copy of the structure object.
- **compare** - Compare two canonical structure objects.
- **set** - Set the canonical structure information (not implemented).
- **get** - Get the canonical structure information.
- **getvalidblocks** - Return a list of valid canonical blocks.
- **getsgconstraints** - Return a list of available StratiGraph constraints.

**mpstruct** Operators:
- **==, ~==** (eq, ne) - Check if two structure objects are equal.
- **(), {}** (subsref) - Index reference for canonical structure objects.

See also **rsblock, lsblock, fjblock, zjblock, ijblock, pstruct**.

Reference page in Doc Center
  doc mpstruct

**Class methods**

**size** Total size of the represented structure.

\[ S = \text{size}(\text{StructObj}) \] returns a three-element row vector \([M \, N \, D]\) containing the size \(M\)-by-\(N\) of the coefficient matrices and degree \(D\) of the matrix polynomial.

\[ [M, N, D] = \text{size}(\text{StructObj}) \] returns the number of rows, columns, and degree in separate output variables.

\[ \text{size}(\text{StructObj}, \text{Dim}) \] returns the length of the dimension specified by the scalar \(\text{Dim}\), where \(\text{Dim}\) maps as 1 → \(M\), 2 → \(N\), and 3 → \(D\).

**rank** Compute the normal rank.

\[ R = \text{rank}(\text{StructObj}) \] determines the normal rank \(R\) of the matrix polynomial structure object \(\text{StructObj}\) from its canonical structure information. The normal rank is equal to the rank of \(P(s)\) at any complex value \(s\) which is not a zero of \(P(s)\).

**hasfullrank** True for canonical structure with full normal rank.
hasfullrank(StructObj) returns logical 1 (true) if StructObj has the normal rank \( R = \min(m,n) \) of the matrix polynomial structure object StructObj, otherwise logical 0 (false).

**codim** Compute the codimension of a matrix polynomial.

**codim**\( (\text{StructObj}) \) determines the codimension of the orbit of the matrix polynomial structure object StructObj. The codimension is computed from the invariants of a Fiedler linearization and is determined with respect to the represented canonical structure not the tangent space.

**codim**\( (\text{StructObj}, \text{Strata}) \) where the optional argument Strata can be
- `'orbit'` Determines the codimension of the orbit. Assumes that all eigenvalues are specified. (default)
- `'bundle'` Determines the codimension of the bundle:
  \[ \text{codim(bundle)} = \text{codim(orbit)} \]
  Assumes that all (finite and infinite) eigenvalues are unspecified.
- `'semibundle'` Determines the codimension of the specified structure, i.e. the codimension is computed as
  \[ \text{codim(bundle)} = \text{codim(orbit)} \]
  \( \text{-(number of distinct unspecified eigenvalues)} \)

**mpstruct** Create a matrix polynomial structure object.

**mpstruct** creates a canonical structure object representing a matrix polynomial \( P(s) = P_d s^d + \ldots + P_1 s + P_0 \), where \( P_d \approx 0 \). The invariants are represented as Kronecker canonical blocks of a matrix pencil with the same invariants as \( P(s) \).

An **mpstruct** object can consist of the following canonical blocks:
- **rsblock** - Right singular block defined as:
  \[
  L_n := \begin{bmatrix} 0 & 1 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 1 \end{bmatrix} - s \begin{bmatrix} 1 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 1 & 0 \end{bmatrix} \quad \text{n-by-(n+1)}
  \]

- **lsblock** - Left singular block defined as:
  \[
  L_n^T := \begin{bmatrix} 1 & \vdots & \vdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 1 & \vdots & 0 \end{bmatrix} - s \begin{bmatrix} 0 & \vdots & \vdots & 1 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \vdots & \vdots & 0 \end{bmatrix} \quad \text{(n+1)-by-n}
  \]

- **fjblock** - Jordan block associated with a finite eigenvalue \( \mu \), defined as:
  \[
  J_n(\mu) = s I_n := \begin{bmatrix} \mu & 1 & 0 \\ \vdots & \ddots & \vdots \\ 0 & \vdots & \mu \end{bmatrix} - s \begin{bmatrix} 1 & \vdots & \vdots & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \vdots & \mu \end{bmatrix} \quad \text{n-by-n}
  \]
zjblock - Jordan block associated with the zero eigenvalue ($\mu=0$).

ijblock - Jordan block associated with the infinite eigenvalue, defined as:

\[
N_n := \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \quad n\text{-by-}n
\]

StructObj = mpstruct() returns an empty canonical structure object representing a matrix polynomial $P(s) = P_d s^d + \ldots + P_1 s + P_0$, where $P_d \sim 0$, of degree $d$. To follow the declarations of the other canonical structure objects, the invariants of $P(s)$ are represented in the form of canonical blocks associated with the Kronecker canonical form of a matrix pencil with the same invariants as $P(s)$. The degree $d$ ensures that the matrix polynomial is unique. The canonical structure information can be specified using one of the three forms 'Size vector form', 'Property-value form', or 'Object form', which are explained below.

Size vector form
---------------
StructObj = mpstruct(d,RSBlocks,LSBlocks,FJBlocksv) returns a new canonical structure object StructObj representing a matrix polynomial. The parameters RSBlocks and LSBlocks define the right and left singular blocks (associated with the column and row minimal indices), respectively, of the structure and are given as row-vectors with the sizes of the blocks. The parameter FJBlocksv defines the Jordan blocks (associated with the finite elementary divisors) of the structure and must be either a row-vector with the sizes of Jordan blocks, all associated with the same eigenvalue, or a cell-array of row-vectors where each vector contains the sizes of blocks associated with the same eigenvalue. By default the associated eigenvalues are set to unspecified (NaN).

StructObj = mpstruct(d,RSBlocks,LSBlocks,FJBlocksv,Eigv) also sets the eigenvalues, where Eigv is a row-vector with the values of each associated eigenvalue corresponding to the row-vectors in FJBlocksv. Jordan blocks with an unspecified associated eigenvalue are defined by setting the corresponding eigenvalue in Eigv to NaN.

StructObj = mpstruct(d,RSBlocks,LSBlocks,FJBlocksv,Eigv,IJBlocks) also sets the sizes of the Jordan blocks with an associated infinite eigenvalue (associated with the infinite elementary divisors).

Property-value form
---------------------
StructObj = mpstruct(d,'BlockName1',StructInt1,'BlockName2',StructInt2,...)
specifies the block types BlockName and the sizes StructInt in property-value form. BlockName is a string specifying the canonical block to be created, and can be one of:

- `'rsblock'` - Right singular blocks
- `'lsblock'` - Left singular blocks
- `'fjblock'` - Jordan blocks associated with finite eigenvalues
- `'zjblock'` - Jordan blocks associated with the zero eigenvalue
- `'ijblock'` - Jordan blocks associated with the infinite eigenvalue

Jordan blocks associated with a specified finite eigenvalue are given as a cell-array tuple in the form `{StructInt Eigenvalue}`.

Example:
```matlab
g >> mpstr = mpstruct(3,'fjblock',{[3 2],0},'fjblock',{[1], -3})
```
creates a matrix polynomial structure object of degree 3 with one Jordan block of size 3x3 and one of size 2x2 both with eigenvalue 0, and one Jordan block of size 1x1 with eigenvalue -3.

`mpstruct(d,BlockName1,StructInt1,...,'Notation',Notation)` also specifies the notation used for StructInt. Valid notations are:

- `'segre'` - Sizes are ordered in a non-increasing order.
- `'weyr'` - Weyr characteristics.
- `'sizes'` - Sizes may be unordered. (default)

Object form
-------------
StructObj = `mpstruct`(d,BlockObj1,BlockObj2,...) creates a structure object from the listed canonical block objects. The block objects must be valid blocks for the structure.

Examples:
```matlab
g >> mpstr = mpstruct([],[],[3 1],4,[1])
```
returns a matrix polynomial structure with one left singular block of size 3x2, one Jordan block of size 3x3 and one of size 1x1 both with eigenvalue 4, and one Jordan block of size 1x1 with an infinite eigenvalue.

Alternatively, the same structure can be created with
```matlab
g >> mpstr = mpstruct('lsblock',2,'ijblock',1,'fjblock',{[3 1],4})
```
returns a matrix polynomial of the following form:

```
| J2(1)  0  0 |
| 0  J1(3) 0 |
| 0  0  N2 |
```

Subscripting
-------------
To access the canonical blocks in the structure object it is possible to use subscripts. See `subsref` for detail and examples. If more control is needed of what should be retrieved, use the method `get` instead.
The class `mpstruct` provides the following methods for extracting information and modifying the canonical structure object.

**mpstruct Methods:**
- **codim** - Compute the codimension of a matrix polynomial.
- **size** - Total size of the represented structure.
- **numblk** - Number of canonical block objects.
- **rank** - Compute the normal rank.
- **hasfullrank** - True for canonical structure with full normal rank.
- **isempty** - True for empty canonical structure.
- **char** - Convert a structure object to a string.
- **exist** - Check if a canonical block of a specified type already exist.
- **copy** - Return a copy of the structure object.
- **compare** - Compare two canonical structure objects.
- **set** - Set the canonical structure information (not implemented).
- **get** - Get the canonical structure information.
- **getvalidblocks** - Return a list of valid canonical blocks.
- **getsgconstraints** - Return a list of available StratiGraph constraints.

**mpstruct Operators:**
- `==`, `~=` (eq, ne) - Check if two structure objects are equal.
- `()`, `{}` (subsref) - Index reference for canonical structure objects.

See also `rsblock`, `lsblock`, `fjblock`, `zjblock`, `ijblock`, `pstruct`.

### 4.4 MSTRUCT

Create a matrix structure object.

`mstruct` creates a canonical structure object representing a matrix A in its canonical form.

An `mstruct` object can only consist of the canonical blocks:
- **fjblock** - Jordan block associated with a finite eigenvalue `mu`, defined as:

  \[
  J_n(\mu) := \begin{bmatrix} 
  \mu & 1 & 0 \\
  & \mu & . \\
  & & \ddots & 1 \\
  & & & \mu 
  \end{bmatrix} \quad \text{n-by-n}
  \]

- **zjblock** - Jordan block associated with the zero eigenvalue.

StructObj = `mstruct()` returns an empty canonical structure object
StructObj representing a matrix A. The canonical structure information can be specified using one of the three forms 'Size vector form', 'Property-value form', or 'Object form', which are explained below.
Size vector form
--------------
StructObj = mstruct(JBlocksv) returns a new canonical structure object representing a matrix. The parameter JBlocksv defines the Jordan blocks of the structure and must be either a row-vector with the sizes of Jordan blocks, all with the same associated eigenvalue, or a cell-array of row-vectors where each vector contains the sizes of blocks with the same associated eigenvalue. By default the associated eigenvalues are set to unspecified (NaN).

StructObj = mstruct(JBlocksv, Eigv) also sets the eigenvalues, where Eigv is a row-vector with the values of each associated eigenvalue corresponding to the row-vectors in JBlocksv. Jordan blocks with an unspecified associated eigenvalue are defined by setting the corresponding eigenvalue in Eigv to NaN.

Property-value form
-------------------
StructObj = mstruct('fjblock',StructInt1,'fjblock',StructInt2,...) specifies the Jordan blocks in property-value form, where each StructInt is the sizes of the blocks associated with the same eigenvalue. Jordan blocks associated with a specified finite eigenvalue are given as a cell-array tuple in the form {StructInt Eigenvalue}. The block 'zjblock' for a Jordan block with zero eigenvalue is also allowed, but then StructInt can only be a vector with sizes. Note that an 'fjblock' with a zero eigenvalue is not.

StructObj = mstruct('fjblock',StructInt1,...,'Notation',Notation) also specifies the notation used for StructInt. Valid notations are:
   'segre' Sizes are ordered in a non-increasing order.
   'weyr' Weyr characteristics.
   'sizes' Sizes may be unordered. (default)

Object form
------------
StructObj = mstruct(BlockObj1,BlockObj2,...) creates a structure object from the listed canonical block objects. The block objects must be valid blocks for the structure.

Examples:
   >> mstr = mstruct([3 1])
returns a matrix structure with two Jordan blocks of sizes 3x3 and 1x1, both with the same but unspecified eigenvalue.

Alternatively, the same structure can be created with
   >> mstr = mstruct('fjblock',[3 1])

   >> mstr = mstruct({[3 2] [1]},[0 -3])
A matrix structure with one Jordan block of size 3x3 and one of size 2x2 both with the eigenvalue 0, and one Jordan block of size 1x1 with the eigenvalue -3.
Alternatively
>> mstr = mstruct('fjblock',[3 2],0,'fjblock',[1, -3])

In the property-value form, Jordan blocks associated with the same finite eigenvalue (not equal to NaN) are joined together, i.e.,
>> mstruct('fjblock',{3,4},'fjblock',{1 2})
returns
J(4) = (3 2 1) (6x6)

However, observe that an 'fjblock' associated with a zero eigenvalue is not the same as a 'zjblock', e.i.,
>> mstruct('fjblock',{3,0},'zjblock',[1 2])
returns
J(0) = (3) (3x3) <- an fjblock
J(0) = (2 1) (3x3) <- two zjblocks

Subscripting
----------
To access the canonical blocks in the structure object it is possible to use subscripts. See subsref for detail and examples. If more control is needed of what should be retrieved, use the method get instead.

The class mstruct provides the following methods for extracting information and modifying the canonical structure object.

mstruct Methods:
codim - Compute the codimension of a matrix.
jcf,JNF - Return the matrix in the Jordan Canonical Form.
size - Total size of the represented structure.
numblk - Number of canonical block objects.
isempty - True for empty canonical structure.
char - Convert a structure object to a string.
exist - Check if a canonical block of a specified type already exist.
copy - Return a copy of the structure object.
compare - Compare two canonical structure objects.
set - Set the canonical structure information (not implemented).
get - Get the canonical structure information.
getvalidblocks - Return a list of valid canonical blocks.
getsgconstraints - Return a list of available StratiGraph constraints.

mstruct Operators:
==, ~= (eq, ne) - Check if two structure objects are equal.
(), {} (subsref) - Index reference for canonical structure objects.

See also fjblock, zjblock, pstruct.

Reference page in Doc Center
doc mstruct
### Class methods

**eig** Return the eigenvalues of the canonical structure.

**codim** Compute the codimension of a matrix.

\[
\text{codim(StructObj)} \text{ determines the codimension of the orbit of the matrix structure object StructObj. The codimension is determined with respect to the represented canonical structure.}
\]

\[
\text{codim(StructObj,Strata)} \text{ where the optional argument Strata can be 'orbit' Determines the codimension of the orbit. Assumes that all eigenvalues are specified. (default) 'bundle' Determines the codimension of the bundle: codim(bundle) = codim(orbit) - (number of distinct eigenvalues) Assumes that all eigenvalues are unspecified. 'semibundle' Determines the codimension of the specified structure, i.e. the codimension is computed as codim(bundle) = codim(orbit) - (number of distinct unspecified eigenvalues) See also mcodim.}
\]

**mstruct** Create a matrix structure object.

\[
\text{mstruct creates a canonical structure object representing a matrix A in its canonical form.}
\]

An **mstruct** object can only consist of the canonical blocks:

- **fjblock** - Jordan block associated with a finite eigenvalue \(\mu\), defined as:
  \[
  J_n(\mu) := \begin{pmatrix}
  \mu & 1 & 0 \\
  \vdots & \ddots & \vdots \\
  0 & \cdots & \mu
  \end{pmatrix}, \quad n\text{-by}-n
  \]

- **zjblock** - Jordan block associated with the zero eigenvalue.

\[
\text{StructObj = mstruct() returns an empty canonical structure object StructObj representing a matrix A. The canonical structure information can be specified using one of the three forms 'Size vector form', Property-value form', or 'Object form', which are explained below.}
\]

**Size vector form**

--------------
StructObj = **mstruct**(JBlocksv) returns a new canonical structure object StructObj representing a matrix. The parameter JBlocksv defines the Jordan blocks of the structure and must be either a row-vector with the sizes of Jordan blocks, all with the same associated eigenvalue, or a cell-array of row-vectors where each vector contains the sizes of blocks with the same associated eigenvalue. By default the associated eigenvalues are set to unspecified (NaN).

StructObj = **mstruct**(JBlocksv, Eigv) also sets the eigenvalues, where Eigv is a row-vector with the values of each associated eigenvalue corresponding to the row-vectors in JBlocksv. Jordan blocks with an unspecified associated eigenvalue are defined by setting the corresponding eigenvalue in Eigv to NaN.

**Property-value form**

-------------------

StructObj = **mstruct**('fjblock',StructInt1,'fjblock',StructInt2,...)

specifies the Jordan blocks in property-value form, where each StructInt is the sizes of the blocks associates with the same eigenvalue. Jordan blocks associated with a specified finite eigenvalue are given as a cell-array tuple in the form {StructInt Eigenvalue}. The block 'zjblock' for a Jordan block with zero eigenvalue is also allowed, but then StructInt can only be a vector with sizes. Note that an 'fjblock' with a zero eigenvalue is not

StructObj = **mstruct**('fjblock',StructInt1,...,'Notation',Notation) also specifies the notation used for StructInt. Valid notations are:

'segre' Sizes are ordered in a non-increasing order.
 'weyr' Weyr characteristics.
'sizes' Sizes may be unordered. (default)

**Object form**

-------------------

StructObj = **mstruct**(BlockObj1,BlockObj2,...) creates a structure object from the listed canonical block objects. The block objects must be valid blocks for the structure.

**Examples:**

```matlab
>> mstr = mstruct([3 1])
returns a matrix structure with two Jordan blocks of sizes 3x3 and 1x1, both with the same but unspecified eigenvalue.

Alternatively, the same structure can be created with
>> mstr = mstruct('fjblock',[3 1])

>> mstr = mstruct({[3 2] [1]},[0 -3])
A matrix structure with one Jordan block of size 3x3 and one of size 2x2 both with the eigenvalue 0, and one Jordan block of size 1x1 with the eigenvalue -3.

Alternatively
```
>> mstr = mstruct('fjblock',[[3 2],0],'fjblock',[[1], -3])

In the property-value form, Jordan blocks associated with the same finite eigenvalue (not equal to NaN) are joined together, i.e.,
>> mstruct('fjblock',[3,4],'fjblock',[1 2])
returns
J(4) = (3 2 1) (6x6)

However, observe that an 'fjblock' associated with a zero eigenvalue is not the same as a 'zjblock', e.i.,
>> mstruct('fjblock',[3,0],'zjblock',[1 2])
returns
J(0) = (3) (3x3) <- an fjblock
J(0) = (2 1) (3x3) <- two zjblocks

Subscripting
------------
To access the canonical blocks in the structure object it is possible to use subscripts. See subsref for detail and examples. If more control is needed of what should be retrieved, use the method get instead.

The class mstruct provides the following methods for extracting information and modifying the canonical structure object.

mstruct Methods:
- codim - Compute the codimension of a matrix.
- jcf, JNF - Return the matrix in the Jordan Canonical Form.
- size - Total size of the represented structure.
- numblk - Number of canonical block objects.
- isempty - True for empty canonical structure.
- char - Convert a structure object to a string.
- exist - Check if a canonical block of a specified type already exist.
- copy - Return a copy of the structure object.
- compare - Compare two canonical structure objects.
- set - Set the canonical structure information (not implemented).
- get - Get the canonical structure information.
- getvalidblocks - Return a list of valid canonical blocks.
- getsgconstraints - Return a list of available StratiGraph constraints.

mstruct Operators:
- ==, ~= (eq, ne) - Check if two structure objects are equal.
- (), {} (subsref) - Index reference for canonical structure objects.

See also fjblock, zjblock, pstruct.

4.5 PSTRUCT

Create a matrix pencil canonical structure object.
**pstruct** creates a canonical structure object representing a matrix pencil $G - sH$ in its canonical form.

A **pstruct** object can consist of the following canonical blocks:

- **rsblock** - Right singular block defined as:
  \[
  L_n := \begin{bmatrix}
  0 & 1 & 0 \\
  0 & 0 & 1 \\
  \vdots & \ddots & \ddots \\
  0 & 0 & 1
  \end{bmatrix}
  \]
  n-by-(n+1)

- **lsblock** - Left singular block defined as:
  \[
  L_n^T := \begin{bmatrix}
  1 & \cdots & 0 \\
  0 & \ddots & \ddots \\
  \vdots & \ddots & \ddots \\
  0 & \cdots & 1
  \end{bmatrix}
  \]
  (n+1)-by-n

- **fjblock** - Jordan block associated with a finite eigenvalue $\mu$, defined as:
  \[
  J_n(\mu) - sI_n := \begin{bmatrix}
  \mu & 1 & 0 \\
  0 & \ddots & \ddots \\
  \vdots & \ddots & \ddots \\
  0 & 0 & \mu
  \end{bmatrix}
  \]
  n-by-n

- **zjblock** - Jordan block associated with the zero eigenvalue ($\mu=0$).

- **ijblock** - Jordan block associated with the infinite eigenvalue, defined as:
  \[
  N_n := \begin{bmatrix}
  1 & 0 & 0 \\
  0 & 1 & 0 \\
  \vdots & \ddots & \ddots \\
  0 & 0 & 1
  \end{bmatrix}
  \]
  n-by-n

**StructObj = pstruct** returns an empty canonical structure object **StructObj** representing a matrix pencil $G - sH$. The canonical structure information can be specified using one of the three forms 'Size vector form', 'Property-value form', or 'Object form', which are explained below.

**Size vector form**

-------------------

**StructObj = pstruct(RSBlocks,LSBlocks,FJBlocksv)** returns a new canonical structure object **StructObj** representing a matrix pencil. The parameters **RSBlocks** and **LSBlocks** define the right and left singular blocks (associated with the column and row minimal indices), respectively, of the structure and are given as row-vectors with the sizes of the blocks. The parameter **FJBlocksv** defines the Jordan blocks (associated with the finite elementary divisors) of the structure and must be either a row-vector with the sizes of Jordan blocks, all associated with the same eigenvalue, or a cell-array of row-vectors where each vector contains the sizes of blocks associated with the same
eigenvalue. By default the associated eigenvalues are set to unspecified (NaN).

StructObj = `pstruct(RSBlocks,LSBlocks,FJBlocksv,Eigv)` also sets the eigenvalues, where `Eigv` is a row-vector with the values of each associated eigenvalue corresponding to the row-vectors in `FJBlocksv`. Jordan blocks with an unspecified associated eigenvalue are defined by setting the corresponding eigenvalue in `Eigv` to NaN.

StructObj = `pstruct(RSBlocks,LSBlocks,FJBlocksv,Eigv,IJBlocks)` also sets the sizes of the Jordan blocks with an associated infinite eigenvalue (associated with the infinite elementary divisors).

Property-value form
-------------------
StructObj = `pstruct('BlockName1',StructInt1,'BlockName2',StructInt2,...)` specifies the block types `BlockName` and the sizes `StructInt` in property-value form. `BlockName` is a string specifying the canonical block to be created, and can be one of:

- `'rsblock'` - Right singular blocks
- `'lsblock'` - Left singular blocks
- `'fjblock'` - Jordan blocks associated with finite eigenvalues
- `'zjblock'` - Jordan blocks associated with the zero eigenvalue
- `'ijblock'` - Jordan blocks associated with the infinite eigenvalue

Jordan blocks associated with a specified finite eigenvalue are given as a cell-array tuple in the form `{StructInt Eigenvalue}`.

Example:
```matlab
>> pstr = pstruct('fjblock',{[3 2],0},'fjblock',{[1], -3})
```
creates a pencil structure object with one Jordan block of size 3x3 and one of size 2x2 both with eigenvalue 0, and one Jordan block of size 1x1 with eigenvalue -3.

`pstruct('BlockName1',StructInt1,...,'Notation',Notation)` also specifies the notation used for `StructInt`. Valid notations are:

- `'segre'` Sizes are ordered in a non-increasing order.
- `'weyr'` Weyr characteristics.
- `'sizes'` Sizes may be unordered. (default)

Object form
-----------
StructObj = `pstruct(BlockObj1,BlockObj2,...)` creates a structure object from the listed canonical block objects. The block objects must be valid blocks for the structure.

StructObj = `pstruct(Xstr)` converts the structure object `Xstr` to a matrix pencil structure object. `Xstr` can be a canonical structure object of the type: (skew-)symmetric matrix pencil, matrix, or state-space system.
Examples:

```matlab
>> pstr = pstruct([], [2], [3 1], 4, [1])
returns a matrix pencil structure with one left singular block of
size 3x2, one Jordan block of size 3x3 and one of size 1x1 both with
eigenvalue 4, and one Jordan block of size 1x1 with an infinite
eigenvalue.
```

Alternatively, the same structure can be created with

```matlab
>> pstr = pstruct('lsblock', 2, 'ijblock', 1, 'fjblock', {[3 1], 4})
```

```matlab
>> pstr = pstruct([], [], {[2] [1]}, [1 3], [2])
returns a matrix pencil of the following form:
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
Class methods

codim Compute the codimension of a matrix pencil.

codim(StructObj) determines the codimension of the orbit of the matrix pencil structure object StructObj. The codimension is determined with respect to the represented canonical structure not the tangent space.

codim(StructObj,Strata) where the optional argument Strata can be
'orbit' Determines the codimension of the orbit. Assumes that all eigenvalues are specified. (default)
'bUNDLE' Determines the codimension of the bundle:
codim(bundle) = codim(orbit)
- (number of distinct eigenvalues)
Assumes that all (finite and infinite) eigenvalues are unspecified.
'semibundle' Determines the codimension of the specified structure, i.e. the codimension is computed as
codim(bundle) = codim(orbit)
- (number of distinct unspecified eigenvalues)

See also pcodim.

pstruct Create a matrix pencil canonical structure object.

pstruct creates a canonical structure object representing a matrix pencil G-sH in its canonical form.

A pstruct object can consist of the following canonical blocks:
rsblock - Right singular block defined as:
| 0 1 0 |
| 1 0 0 |
L_n := | . . | - s | . . | n-by-(n+1)
| 0 0 1 |
| 0 1 0 |

lsblock - Left singular block defined as:
| 0 0 |
| 1 0 |
L_n^T := | 1 . | - s 0 . | (n+1)-by-n
| . 0 |
| . 1 |
| 0 1 |
| 0 0 |

fjblock - Jordan block associated with a finite eigenvalue mu, defined as:
| mu 1 0 |
J_n(mu) - s*I_n := | mu . | - s*I_n n-by-n
| . 1 |
| 0 mu |

zjblock - Jordan block associated with the zero eigenvalue (mu=0).
ijb的根本 - 与无限特征值相关的Jordan块，定义为：

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{bmatrix}
\]

\[
N_n := \begin{bmatrix}
1 \\
- s \\
0 \\
\end{bmatrix}
\]

n-by-n

StructObj = pstruct returns an empty canonical structure object
StructObj representing a matrix pencil G-sH. The canonical structure
information can be specified using one of the three forms 'Size vector
form', Property-value form', or 'Object form', which are explained
below.

Size vector form
----------------
StructObj = pstruct(RSBlocks,LSBlocks,FJBlocksv) returns a new
canonical structure object StructObj representing a matrix pencil. The
parameters RSBlocks and LSBlocks define the right and left singular
blocks (associated with the column and row minimal indices),
respectively, of the structure and are given as row-vectors with the
sizes of the blocks. The parameter FJBlocksv defines the Jordan blocks
(associated with the finite elementary divisors) of the structure and
must be either a row-vector with the sizes of Jordan blocks, all
associated with the same eigenvalue, or a cell-array of row-vectors
where each vector contains the sizes of blocks associated with the same
eigenvalue. By default the associated eigenvalues are set to
unspecified (NaN).

StructObj = pstruct(RSBlocks,LSBlocks,FJBlocksv,Eigv) also sets the
eigenvalues, where Eigv is a row-vector with the values of each
associated eigenvalue corresponding to the row-vectors in FJBlocksv.
Jordan blocks with an unspecified associated eigenvalue are defined by
setting the corresponding eigenvalue in Eigv to NaN.

StructObj = pstruct(RSBlocks,LSBlocks,FJBlocksv,Eigv,IJBlocks) also
sets the sizes of the Jordan blocks with an associated infinite
eigenvalue (associated with the infinite elementary divisors).

Property-value form
---------------------
StructObj =
pstruct('BlockName1',StructInt1,'BlockName2',StructInt2,...) specifies
the block types BlockName and the sizes StructInt in property-value
form. BlockName is a string specifying the canonical block to be
created, and can be one of:
'rsblock' - Right singular blocks
'lsblock' - Left singular blocks
'fjblock' - Jordan blocks associated with finite eigenvalues
'zjblock' - Jordan blocks associated with the zero eigenvalue
'ijblock' - Jordan blocks associated with the infinite eigenvalue

Jordan blocks associated with a specified finite eigenvalue are given as a cell-array tuple in the form {StructInt Eigenvalue}.

Example:
>> pstr = pstruct('fjblock', {[3 2], 0}, 'fjblock', {[1], -3})
creates a pencil structure object with one Jordan block of size 3x3 and one of size 2x2 both with eigenvalue 0, and one Jordan block of size 1x1 with eigenvalue -3.

`pstruct('BlockName1',StructInt1,...,'Notation',Notation)` also specifies the notation used for StructInt. Valid notations are:
'segre' Sizes are ordered in a non-increasing order.
'weyr' Weyr characteristics.
'sizes' Sizes may be unordered. (default)

Object form
-----------

StructObj = `pstruct`(BlockObj1, BlockObj2,...) creates a structure object from the listed canonical block objects. The block objects must be valid blocks for the structure.

StructObj = `pstruct`(Xstr) converts the structure object Xstr to a matrix pencil structure object. Xstr can be a canonical structure object of the type: (skew-)symmetric matrix pencil, matrix, or state-space system.

Examples:
>> pstr = pstruct([], [2], [3 1], 4, [1])
returns a matrix pencil structure with one left singular block of size 3x2, one Jordan block of size 3x3 and one of size 1x1 both with eigenvalue 4, and one Jordan block of size 1x1 with an infinite eigenvalue.

Alternatively, the same structure can be created with
>> pstr = pstruct('lsblock', 2, 'ijblock', 1, 'fjblock', {[3 1], 4})

>> pstr = pstruct([], [], {[2] [1]}, {[1 3], [2]})
returns a matrix pencil of the following form:

| J2(1) 0 0 |
| 0 J1(3) 0 |
| 0 0 N2 |

Subscripting
------------

To access the canonical blocks in the structure object it is possible to use subscripts. See `subsref` for detail and examples. If more control is needed of what should be retrieved, use the method `get` instead.
The class **pstruct** provides the following methods for extracting information and modifying the canonical structure object.

**pstruct Methods:**
- **codim** - Compute the codimension of a matrix pencil.
- **kcf** - Return the matrix pencil in the Kronecker Canonical Form.
- **size** - Total size of the represented structure.
- **numblk** - Number of canonical block objects.
- **isempty** - True for empty canonical structure.
- **char** - Convert a structure object to a string.
- **exist** - Check if a canonical block of a specified type already exist.
- **copy** - Return a copy of the structure object.
- **compare** - Compare two canonical structure objects.
- **set** - Set the canonical structure information (not implemented).
- **get** - Get the canonical structure information.
- **getvalidblocks** - Return a list of valid canonical blocks.
- **getsgconstraints** - Return a list of available StratiGraph constraints.

**pstruct Operators:**
- **==, ~= (eq, ne)** - Check if two structure objects are equal.
- **(), {} (subsref)** - Index reference for canonical structure objects.

See also **rsblock**, **lsblock**, **fjblock**, **zjblock**, **ijblock**, **spstruct**, **sspstruct**, **mstruct**.

### 4.6 SCMSTRUCT

Create a *congruence matrix structure object.

**scmstruct** creates a canonical structure object representing a matrix \( A \) in its canonical form under *congruence.

A **scmstruct** object can consist of the following canonical blocks:
- **sgblock** - *\( \Gamma \) block associated with a complex parameter \( \mu \), defined as:
  \[
  \begin{bmatrix}
  0 & \ldots \\
  \vdots & \ddots \\
  1 & 1 \\
  -1 & -1 \\
  1 & 1 & 0
  \end{bmatrix}_{n \times n}
  \]
  where \( |\mu| = 1 \).

- **swblock** - *W block associated with a specified and admissible eigenvalue \( \mu \), defined as:
  \[
  \begin{bmatrix}
  0 & I_m \\
  \vdots & \vdots \\
  J_m(\mu) & 0
  \end{bmatrix}_{2n \times 2n}
  \]
  where \( |\mu| > 1 \) and \( J_m(\mu) \) is an \( m \times m \) Jordan block associated with the eigenvalue \( \mu \).
zjblock - Jordan block associated with the zero eigenvalue, defined as:

\[
J_n(0) := \begin{bmatrix}
0 & 1 & 0 \\
0 & . & 1 \\
0 & 0 & 1 
\end{bmatrix}
\]

n-by-n

StructObj = \texttt{scmstruct()} returns an empty canonical structure object StructObj representing a matrix \( A \) under *congruence. The canonical structure information can be specified using one of the three forms 'Size vector form', Property-value form, or 'Object form', which are explained below.

Size vector form
----------------
StructObj = \texttt{scmstruct}(SGBlocksv,Muv,SWBlocksv,Eigv) returns a new canonical structure object StructObj representing a matrix under *congruence. The parameter SGBlocksv defines the *Gamma blocks of the structure and is given as a cell-array of row-vectors where each vector contains the sizes of blocks associated with the same parameter \( \mu \). Muv is a row-vector with the values of each associated parameters \( \mu \) corresponding to the row-vectors in SGBlocksv. The parameter SWBlocksv defines the *W blocks associated with Jordan blocks of the structure and must be a cell-array of row-vectors where each vector contains the sizes of the blocks associated with the same eigenvalue. Eigv is a row-vector with the values of each associated eigenvalue corresponding to the row-vectors in SWBlocksv.

StructObj = \texttt{scmstruct}(SGBlocks,Muv,SWBlocksv,Eigv,ZJBlocks) also sets the sizes of the Jordan blocks with an associated zero eigenvalue.

Property-value form
---------------------
StructObj = \texttt{scmstruct('BlockName1',StructInt1,'BlockName2',StructInt2,...)} specifies the block types BlockName and the sizes StructInt in property-value form. BlockName is a string specifying the canonical block to be created, and can be one of:

'SGblock' - *Gamma blocks
'SWblock' - *W blocks associated with Jordan blocks (associated with finite eigenvalues)
'Zjblock' - Jordan blocks associated with the zero eigenvalue.

*Gamma and *W blocks associated with a specified parameter are given as a cell-array tuple in the form \{StructInt Parameter\}.

Example:

>> scmstr = scmstruct('swblock',[[3 2],2],'sgblock',[[1], 1i]) creates a structure object with one *W block of size 6x6 and one of size 4x4 both with the associated eigenvalue 2, and one
*Gamma block of size 2x2 with the associated complex parameter 1i.

\texttt{scmstruct('BlockName1',StructInt1,...,'Notation',Notation)} also specifies the notation used for StructInt. Valid notations are:
- 'segre' Sizes are ordered in a non-increasing order.
- 'weyr' Weyr characteristics.
- 'sizes' Sizes may be unordered. (default)

Object form
-------------
\texttt{StructObj = scmstruct(BlockObj1,BlockObj2,...)} creates a structure object from the listed canonical block objects. The block objects must be valid blocks for the structure.

Examples:

\begin{verbatim}
>> scmstr = scmstruct([],[],[3,1],4,[1])
\end{verbatim}
returns a matrix structure with one *W block of size 6x6 and one of size 2x2 both with eigenvalue 4, and one Jordan block of size 1x1 with the zero eigenvalue.

Alternatively, the same structure can be created with

\begin{verbatim}
>> scmstr = scmstruct('zjblock',1,'swblock',[3,1],4)
\end{verbatim}

\begin{verbatim}
>> scmstr = scmstruct([], [], {[2] [1]}, [5,3], [2])
\end{verbatim}
returns a matrix of the following form:

\begin{verbatim}
|   SW2(5) 0 0 |
|     0   SW1(3) 0 |
|     0     0   J2(0) |
\end{verbatim}

Subscripting
-------------
To access the canonical blocks in the structure object it is possible to use subscripts. See \texttt{subsref} for detail and examples. If more control is needed of what should be retrieved, use the method \texttt{get} instead.

The class \texttt{scmstruct} provides the following methods for extracting information and modifying the canonical structure object.

\texttt{scmstruct} Methods:
- \texttt{codim} - Compute the codimension of a matrix under *congruence.
- \texttt{ccf} - Return the matrix in the congruence canonical form.
- \texttt{size} - Total size of the represented structure.
- \texttt{numblk} - Number of canonical block objects.
- \texttt{isempty} - True for empty canonical structure.
- \texttt{char} - Convert a structure object to a string.
- \texttt{exist} - Check if a canonical block of a specified type already exist.
- \texttt{copy} - Return a copy of the structure object.
- \texttt{compare} - Compare two canonical structure objects.
set - Set the canonical structure information (not implemented).

get - Get the canonical structure information.

getvalidblocks - Return a list of available canonical blocks.

pstruct Operators:

==, ~= (eq, ne) - Check if two structure objects are equal.

(), {} (subsref) - Index reference for canonical structure objects.

See also sgblock, swblock, zjblock, cmstruct, mstruct.

Reference page in Doc Center
doc scmstruct

Class methods

codim Compute the codimension of a matrix under *congruence.

codim(StructObj) determines the codimension of the *congruence orbit of the matrix structure object StructObj. The codimension is determined with respect to the represented canonical structure not the tangent space.

codim(StructObj,Strata) where the optional argument Strata can be 'orbit' Determines the codimension of the orbit.

Assumes that all eigenvalues are specified. (default)

'bundle' Not implemented!

See also scmcodim.

scmstruct Create a *congruence matrix structure object.

scmstruct creates a canonical structure object representing a matrix A in its canonical form under *congruence.

A scmstruct object can consist of the following canonical blocks:

sgblock - *Gamma block associated with a complex parameter mu, defined as:

\[
\begin{pmatrix}
0 & \cdots \\
\vdots & \ddots & \ddots \\
-1 & -1 & \cdots \\
1 & 1 & 0 \\
mu & 1 & 1 & \cdots & n-by-n
\end{pmatrix}
\]

where \(|\mu| = 1\).

swblock - *W block associated with a specified and admissible eigenvalue mu, defined as:

\[
\begin{pmatrix}
0 & I_m \\
\vdots & \ddots & \ddots \\
J_m(\mu) & 0 \\
J_m(\mu) & 0
\end{pmatrix}
\]

where \(|\mu| > 1\) and \(J_m(\mu)\) is an m-by-m Jordan
block associated with the eigenvalue mu.

\[
\text{zjblock} - \text{Jordan block associated with the zero eigenvalue, defined as:} \\
\begin{bmatrix} 
0 & 1 & 0 \\
0 & . & 1 \\
0 & 0 \\
\end{bmatrix} \\
J_n(0) := \begin{bmatrix} 
0 & 1 \\
0 \\
\end{bmatrix} \text{n-by-n}
\]

\text{StructObj} = \text{scmstruct()} \text{returns an empty canonical structure object StructObj representing a matrix } A \text{ under *congruence. The canonical structure information can be specified using one of the three forms 'Size vector form', Property-value form', or 'Object form', which are explained below.}

\text{Size vector form}

-------------------

\text{StructObj} = \text{scmstruct(SGBlocksv,Muv,SWBlocksv,Eigv)} \text{returns a new canonical structure object StructObj representing a matrix under *congruence. The parameter SGBlocksv defines the *Gamma blocks of the structure and is given as a cell-array of row-vectors where each vector contains the sizes of blocks associated with the same parameter mu. Muv is a row-vector with the values of each associated parameters mu corresponding to the row-vectors in SGBlocksv. The parameter SWBlocksv defines the *W blocks associated with Jordan blocks of the structure and must be a cell-array of row-vectors where each vector contains the sizes of the blocks associated with the same eigenvalue. Eigv is a row-vector with the values of each associated eigenvalue corresponding to the row-vectors in SWBlocksv.}

\text{StructObj} = \text{scmstruct(SGBlocks,Muv,SWBlocksv,Eigv,ZJBlocks)} \text{also sets the sizes of the Jordan blocks with an associated zero eigenvalue.}

\text{Property-value form}

-------------------

\text{StructObj} = \text{scmstruct('BlockName1',StructInt1,'BlockName2',StructInt2,...)} \text{specifies the block types BlockName and the sizes StructInt in property-value form. BlockName is a string specifying the canonical block to be created, and can be one of: 'sgblock' - *Gamma blocks 'swblock' - *W blocks associated with Jordan blocks (associated with finite eigenvalues) 'zjblock' - Jordan blocks associated with the zero eigenvalue. *Gamma and *W blocks associated with a specified parameter are given as a cell-array tuple in the form (StructInt Parameter).}

Example:

\text{>> scmstr = scmstruct('swblock',{[3 2],2},'sgblock',{[1], 1i})}
creates a structure object with one *W block of size 6x6 and
one of size 4x4 both with the associated eigenvalue 2, and one
*Gamma block of size 2x2 with the associated complex parameter
1i.

`scmstruct('BlockName1',StructInt1,...,'Notation',Notation)` also
specifies the notation used for StructInt. Valid notations are:
- 'segre' Sizes are ordered in a non-increasing order.
- 'weyr' Weyr characteristics.
- 'sizes' Sizes may be unordered. (default)

Object form
-------------
`StructObj = scmstruct(BlockObj1,BlockObj2,...)` creates a structure
object from the listed canonical block objects. The block objects must
be valid blocks for the structure.

Examples:
```
>> scmstr = scmstruct([],[],[3 1],4,[1])
returns a matrix structure with one *W block of size 6x6 and one of
size 2x2 both with eigenvalue 4, and one Jordan block of size 1x1
with the zero eigenvalue.
```
Alternatively, the same structure can be created with
```
>> scmstr = scmstruct('zjblock',1,'swblock',{[3 1],4})
```
```
>> scmstr = scmstruct([], [], {[2] [1]}, [5 3], [2])
returns a matrix of the following form:
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
copy - Return a copy of the structure object.
copy - Compare two canonical structure objects.
set - Set the canonical structure information (not implemented).
get - Get the canonical structure information.
getvalidblocks - Return a list of available canonical blocks.

pstruct Operators:
  ==, ~= (eq, ne) - Check if two structure objects are equal.
  (), {} (subsref) - Index reference for canonical structure objects.

See also sgblock, swblock, zjblock, cmstruct, mstruct.

4.7 SPSTRUCT

Create a symmetric matrix pencil structure object.

spstruct creates a canonical structure object representing a symmetric matrix pencil G-sH in its canonical form.

A spstruct object can consist of the following canonical blocks:

mblock - Singular M block defined as:
\[ M_n := \begin{bmatrix} 0 & G_n^T \\ G_n & 0 \end{bmatrix} \begin{bmatrix} 0 & F_n^T \\ F_n & 0 \end{bmatrix} \]
where
\[
G_n := \begin{bmatrix} \vdots & \vdots \\ 0 & 1 & 0 \end{bmatrix}, \quad F_n := \begin{bmatrix} \vdots & \vdots \\ 0 & 0 & 1 \end{bmatrix}
\]

hblock - H block, associated with a finite or unspecified eigenvalues mu, defined as:
\[ H_n(\mu) := \begin{bmatrix} 0 & \mu \\ \mu & 1 \end{bmatrix} - s \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \]

kblock - K block, associated with the infinite eigenvalue, defined as:
\[ K_n := \begin{bmatrix} 0 & 1 \\ 1 & -s \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \]

StructObj = spstruct() returns an empty canonical structure object
StructObj representing a symmetric matrix pencil G-sH. The canonical structure information can be specified using one of the three forms 'Size vector form', Property-value form', or 'Object form', which are explained below.
Size vector form

StructObj = \texttt{spstruct}(MBlocks,HBlocksv) returns a new canonical structure object StructObj representing a symmetric matrix pencil $G-H$. The parameter $MBlocks$ defines the singular $M$ blocks of the structure and is given as a row-vector with the sizes of the blocks. The parameter $HBlocksv$ defines the $H$ blocks associated with finite or unspecified eigenvalues and must either be a row-vector with the indices of blocks, all associated with the same eigenvalue, or a cell-array of row-vectors where each vector contains the sizes of blocks associated with the same eigenvalue. By default the associated eigenvalues are set to unspecified (NaN).

StructObj = \texttt{spstruct}(MBlocks,HBlocksv,Eigv) sets the eigenvalues, where $Eigv$ is a row-vector with the values of each associated eigenvalue corresponding to the row-vectors in $HBlocksv$. The blocks with an unspecified associated eigenvalue are defined by setting the corresponding eigenvalue in $Eigv$ to NaN.

StructObj = \texttt{spstruct}(MBlocks,HBlocksv,Eigv,KBlocks) also sets the sizes of the $K$ blocks associated with the infinite eigenvalue.

Property-value form

StructObj = \texttt{spstruct}('BlockName1',StructInt1,'BlockName2',StructInt2,...) specifies the block types $BlockName$ and the sizes $StructInt$ in property-value form. $BlockName$ is a string specifying the canonical block to be created, and can be one of:

- 'mblock' - Singular $M$ blocks.
- 'hblock' - $H$ blocks associated finite eigenvalues.
- 'kblock' - $K$ blocks associated the infinite eigenvalue.

$H$ blocks associated with a specified finite eigenvalue are given as a cell-array tuple in the form $\{StructInt, Eigenvalue\}$.

Example:

\texttt{
\begin{verbatim}
>> spstr = spstruct('hblock',[[10 3],[7]],'mblock',[3,2]) creates a pencil structure object with two $H$ blocks of the sizes 10x10 and 3x3 both with eigenvalue 7, and two singular $M$ block of the sizes 7x7 and 5x5.
\end{verbatim}
}\texttt{
\end{verbatim}
}

\texttt{spstruct}('BlockName1',StructInt1,...,'Notation',Notation) also specifies the notation used for $StructInt$. Valid notations are:

- 'segre' Indices are ordered in a non-increasing order.
- 'weyr' Weyr characteristics.
- 'sizes' Indices may be unordered. (default)

Object form

StructObj = \texttt{spstruct}(BlockObj1,BlockObj2,...) creates a structure object from the listed canonical block objects. The block objects must
be valid blocks for the structure.

Example:
```matlab
>> spstr = spstruct([2], [3 1], 4, [1])
returns a symmetric matrix pencil structure with one M block of size 5, one H block of size 3 and one of size 1, both with the eigenvalue 4, and one K block of size 1 with the infinite eigenvalue.

Alternatively, the same structure can be created with
```matlab
>> spstr = spstruct('mblock',2,'kblock',1,'hblock',{[3 1],4})
```matlab
```matlab
>> spstr = spstruct([], {[2] [1]}, [1 3], [2])
returns a symmetric matrix pencil of the following form:
```
| H2(1) 0 0 |
| 0   H1(3) 0 |
| 0 0 K2 |
```

Subscripting
-------------
To access the canonical blocks in the structure object it is possible to use subscripts. See `subsref` for detail and examples. If more control is needed of what should be retrieved, use the method `get` instead.

The class `spstruct` provides the following methods for extracting information and modifying the canonical structure object.

**spstruct** Methods:
- `codim` - Compute the codimension of a symmetric matrix pencil.
- `kcf` - Return the symmetric matrix pencil in a Kronecker-like canonical form (described above).
- `size` - Total size of the represented structure.
- `numblk` - Number of canonical block objects.
- `isempty` - True for empty canonical structure.
- `char` - Convert a structure object to a string.
- `exist` - Check if a canonical block of a specified type already exist.
- `copy` - Return a copy of the structure object.
- `compare` - Compare two canonical structure objects.
- `set` - Set the canonical structure information (not implemented).
- `get` - Get the canonical structure information.
- `getvalidblocks` - Return a list of available canonical blocks.

**spstruct** Operators:
- `==, ~= (eq, ne)` - Check if two structure objects are equal.
- `(), {} (subsref)` - Index reference for canonical structure objects.

See also `mblock`, `hblock`, `kblock`, `sspstruct`, `pstruct`.

Reference page in Doc Center
```matlab
doc spstruct
```
Class methods

**codim** Compute the codimension of a symmetric matrix pencil.

**codim**(*StructObj*) determines the codimension of the orbit of the symmetric matrix pencil structure object *StructObj*. The codimension is determined with respect to the represented structure not the tangent space.

**codim**(*StructObj*, *Strata*) where the optional argument *Strata* can be

- 'orbit' Determines the codimension of the orbit.
  Assumes that all eigenvalues are specified. (default)
- 'bundle' Determines the codimension of the bundle.
  Assumes that all (finite and infinite) eigenvalues are unspecified.
- 'semibundle' Determines the codimension of the specified structure, i.e., the codimension is computed as **codim**(orbit) - (number of distinct unspecified eigenvalues).

Remark:
**codim**(bundle) = **codim**(orbit) - (number of distinct eigenvalues)

See also **spcodim**.

**spstruct** Create a symmetric matrix pencil structure object.

**spstruct** creates a canonical structure object representing a symmetric matrix pencil G-sH in its canonical form.

A **spstruct** object can consist of the following canonical blocks:

- **mblock** - Singular M block defined as:
  \[
  M_n := \begin{bmatrix}
  0 & G_n^T & 0 & F_n^T \\
  G_n & 0 & F_n & 0 \\
  \end{bmatrix}
  \]
  
  where
  \[
  G_n := \begin{bmatrix}
  . & . & . & . \\
  0 & 0 & 1 & . \\
  \end{bmatrix}, \text{ and } F_n := \begin{bmatrix}
  . & . & . \\
  0 & 0 & 1 & . \\
  \end{bmatrix}
  \]
  
  \[
  (2n+1)-\text{by-}(2n+1)
  \]

- **hblock** - H block, associated with a finite or unspecified eigenvalues \( \mu \) defined as:
  \[
  H_n(\mu) := \begin{bmatrix}
  0 & \mu & 0 & 1 \\
  \mu & 1 & -s & 1 \\
  . & . & . & . \\
  \mu & 1 & 0 & 1 \\
  \end{bmatrix}
  \]
  
  \[
  n-\text{by-}n
  \]

- **kblock** - K block, associated with the infinite eigenvalue, defined as:
StructObj = spstruct() returns an empty canonical structure object
StructObj representing a symmetric matrix pencil G-sH. The canonical
structure information can be specified using one of the three forms
'Size vector form', 'Property-value form', or 'Object form', which are
explained below.

Size vector form
---------------
StructObj = spstruct(MBlocks, HBlocksv) returns a new canonical
structure object StructObj representing a symmetric matrix pencil G-sH.
The parameter MBlocks defines the singular M blocks of the structure
and is given as a row-vector with the sizes of the blocks. The
parameter HBlocksv defines the H blocks associated with finite or
unspecified eigenvalues and must either be a row-vector with the
indices of blocks, all associated with the same eigenvalue, or a
cell-array of row-vectors where each vector contains the sizes of
blocks associated with the same eigenvalue. By default the associated
eigenvalues are set to unspecified (NaN).

StructObj = spstruct(MBlocks, HBlocksv, Eigv) sets the eigenvalues, where
Eigv is a row-vector with the values of each associated eigenvalue
corresponding to the row-vectors in HBlocksv. The blocks with an
unspecified associated eigenvalue are defined by setting the
corresponding eigenvalue in Eigv to NaN.

StructObj = spstruct(MBlocks, HBlocksv, Eigv, KBlocks) also sets the sizes
of the K blocks associated with the infinite eigenvalue.

Property-value form
---------------------
StructObj = spstruct('BlockName1', StructInt1, 'BlockName2', StructInt2,...) specifies
the block types BlockName and the sizes StructInt in property-value
form. BlockName is a string specifying the canonical block to be
created, and can be one of:
    'mblock' - Singular M blocks.
    'hblock' - H blocks associated finite eigenvalues.
    'kblock' - K blocks associated the infinite eigenvalue.

H blocks associated with a specified finite eigenvalue are given as a
cell-array tuple in the form {StructInt Eigenvalue}.

Example:
>> spstr = spstruct('hblock', {[10 3], [7]}, 'mblock', [3, 2]) creates a
pencil structure object with two H blocks of the sizes 10x10 and 3x3
both with eigenvalue 7, and two singular M block of the sizes 7x7 and 5x5.

\[ \text{spstruct('BlockName1',StructInt1,...,'Notation',Notation)} \]
also specifies the notation used for StructInt. Valid notations are:
- 'segre' Indices are ordered in a non-increasing order.
- 'weyr' Weyr characteristics.
- 'sizes' Indices may be unordered. (default)

Object form
-----------
StructObj = spstruct(BlockObj1,BlockObj2,...) creates a structure object from the listed canonical block objects. The block objects must be valid blocks for the structure.

Example:
```
>> spstr = spstruct([2], [3 1], 4, [1])
```
returns a symmetric matrix pencil structure with one M block of size 5, one H block of size 3 and one of size 1, both with the eigenvalue 4, and one K block of size 1 with the infinite eigenvalue.

Alternatively, the same structure can be created with
```
>> spstr = spstruct('mblock',2,'kblock',1,'hblock',{[3 1],4})
```
```
>> spstr = spstruct([], {[2] [1]}, [1 3], [2])
```
returns a symmetric matrix pencil of the following form:
```
| H2(1)  0   0 |
| 0      H1(3) 0 |
| 0      0     K2 |
```

Subscripting
------------
To access the canonical blocks in the structure object it is possible to use subscript. See \texttt{subsref} for detail and examples. If more control is needed of what should be retrieved, use the method \texttt{get} instead.

The class \texttt{spstruct} provides the following methods for extracting information and modifying the canonical structure object.

\texttt{spstruct} Methods:
- \texttt{codim} - Compute the codimension of a symmetric matrix pencil.
- \texttt{kcf} - Return the symmetric matrix pencil in a Kronecker-like canonical form (described above).
- \texttt{size} - Total size of the represented structure.
- \texttt{numblk} - Number of canonical block objects.
- \texttt{isempty} - True for empty canonical structure.
- \texttt{char} - Convert a structure object to a string.
- \texttt{exist} - Check if a canonical block of a specified type already exist.
- \texttt{copy} - Return a copy of the structure object.
compare - Compare two canonical structure objects.
set - Set the canonical structure information (not implemented).
get - Get the canonical structure information.
getvalidblocks - Return a list of available canonical blocks.

sspstruct Operators:
==, ~= (eq, ne) - Check if two structure objects are equal.
(), {} (subsref) - Index reference for canonical structure objects.

See also mblock, hblock, kblock, sspstruct, pstruct.

4.8 SSPSTRUCT

Create a skew-symmetric matrix pencil structure object.

sspstruct creates a canonical structure object representing a skew-symmetric matrix pencil G-sH in its canonical form.

A sspstruct object can consist of the following canonical blocks:
smblock - Singular SM block defined as:
\[
\begin{bmatrix}
0 & G_n & 0 \\
- & s & F_n \\
-G_n^T & 0 & -F_n^T & 0
\end{bmatrix}
\]

where
\[
\begin{bmatrix}
0 & 1 & 0 \\
1 & 0 & 0
\end{bmatrix}
\]

G_n := \ldots and F_n := \ldots

shblock - SH block associated with a finite or unspecified eigenvalues mu, defined as:
\[
\begin{bmatrix}
0 & J_n(mu) & 0 & I_n \\
- & s & 0 & -I_n \\
-J_n(mu)^T & 0 & -I_n & 0
\end{bmatrix}
\]

where J_n(mu) is an n-by-n Jordan block associated with the eigenvalue mu.

skblock - SK block associated with the infinite eigenvalue, defined as:
\[
\begin{bmatrix}
0 & I_n & 0 & J_n(0) \\
- & s & 0 & -J_n(0)^T & 0
\end{bmatrix}
\]

where J_n(0) is an n-by-n Jordan block associated with the zero eigenvalue.

StructObj = sspstruct() returns an empty canonical structure object StructObj representing a skew-symmetric matrix pencil G-sH. The canonical structure information can be specified using one of the three forms 'Size vector form', Property-value form', or 'Object form', which are explained below.
Size vector form

----------------
StructObj = \texttt{sspstruct}(\text{SMBlocks},\text{SHBlocksv}) \text{ returns a new canonical structure object StructObj representing a skew-symmetric matrix pencil.}

The parameter \text{SMBlocks} defines the singular SM blocks of the structure and is given as a row-vector with the sizes of the blocks. The parameter \text{SHBlocksv} defines the SH blocks associated with the finite or unspecified eigenvalues and must be either a row-vector with the indices of blocks, all associated with the same eigenvalue, or a cell-array of row-vectors where each vector contains the sizes of blocks associated with the same eigenvalue. By default the associated eigenvalues are set to unspecified (NaN).

StructObj = \texttt{sspstruct}(\text{SMBlocks},\text{SHBlocksv},\text{Eigv}) \text{ sets the eigenvalues,}

where \text{Eigv} is a row-vector with the values of each associated eigenvalue corresponding to the row-vectors in \text{SHBlocksv}. The SH blocks with an unspecified associated eigenvalue are defined by setting the corresponding eigenvalue in \text{Eigv} to NaN.

StructObj = \texttt{sspstruct}(\text{SMBlocks},\text{SHBlocksv},\text{Eigv},\text{SKBlocks}) \text{ also sets the sizes of the SK blocks associated with the infinite eigenvalue.}

Property-value form

---------------------
StructObj = \texttt{sspstruct}(\text{\'BlockName1\',\text{StructInt1},\text{\'BlockName2\',\text{StructInt2},\ldots}) \text{ specifies the block types BlockName and the sizes StructInt in property-value form. BlockName is a string specifying the canonical block to be created, and can be one of:}

\begin{itemize}
  \item \text{\textquoteleft smbloc\textquoteleft} - Singular SM blocks.
  \item \text{\textquoteleft shblock\textquoteleft} - SH blocks associated with finite eigenvalues.
  \item \text{\textquoteleft skbloc\textquoteleft} - SK blocks associated with the infinite eigenvalue.
\end{itemize}

SH blocks associated with a specified finite eigenvalue are given as a cell-array tuple in the form \{\text{StructInt Eigenvalue}\}.

Example:

\begin{verbatim}
>> sspstr = sspstruct('shblock',{
[10 3],[7],
'smbloc',[3,2])
\end{verbatim}

creates a pencil structure object with two SH blocks of the sizes 20x20 and 6x6 both with the eigenvalue 7, and two singular blocks of the sizes 7x7 and 5x5.

\texttt{sspstruct(\text{\'BlockName1\',\text{StructInt1},\ldots,'\text{\textquoteleft Notation\textquoteleft},\text{Notation}) also specifies the notation used for StructInt. Valid notations are:}

\begin{itemize}
  \item \text{\textquoteleft segre\textquoteleft} - Indices are ordered in a non-increasing order.
  \item \text{\textquoteleft weyr\textquoteleft} - Weyr characteristics.
  \item \text{\textquoteleft sizes\textquoteleft} - Indices may be unordered. (default)
\end{itemize}

Object form

----------
StructObj = \texttt{sspstruct(\text{BlockObj1},\text{BlockObj2},\ldots)} \text{ creates a structure object from the listed canonical block objects. The block objects must}
be valid blocks for the structure.

Example:

```matlab
t >> sspstr = sspstruct([2], [3 1], 4, [1])
returns a skew-symmetric matrix pencil structure with one SM
block of size 5, one SH block of size 6 and one of size 2,
both associated with eigenvalue 4, and one SK block of size 2
associated with the infinite eigenvalue.
```

Alternatively, the same structure can be created with
```matlab
>> sspstr = sspstruct('smblock',2,'skblock',1,'shblock',[3 1],4))
```

```matlab
>> sspstr = sspstruct([4], {[2] [1]}, [1 3], [2])
```

returns a skew-symmetric matrix pencil of the following form:

```
| SM4 0 0 0 |
| 0 SH2(1) 0 0 |
| 0 0 SH1(3) 0 |
| 0 0 0 SK2 |
```

Subscripting
------------
To access the canonical blocks in the structure object it is possible
to use subscript. See `subsref` for detail and examples. If more control
is needed of what should be retrieved, use the method `get` instead.

The class `ssppstruct` also provides methods for extracting information
and modifying the canonical structure object.

**ssppstruct** Methods:

- `codim` - Compute the codimension of a skew-symmetric matrix
  pencil.
- `kcf` - Return the skew-symmetric matrix pencil in a
  Kronecker-like canonical form (described above).
- `size` - Total size of the represented structure.
- `numblk` - Number of canonical block objects.
- `isempty` - True for empty canonical structure.
- `char` - Convert a structure object to a string.
- `exist` - Check if a canonical block of a specified type
  already exist.
- `copy` - Return a copy of the structure object.
- `compare` - Compare two canonical structure objects.
- `set` - Set the canonical structure information (not implemented).
- `get` - Get the canonical structure information.
- `getvalidblocks` - Return a list of available canonical blocks.

**ssppstruct** Operators:

- `==`, `~=` `(eq, ne)` - Check if two structure objects are equal.
- `(), {}` `(subsref)` - Index reference for canonical structure objects.
See also `smblock`, `shblock`, `skblock`, `spstrcut`, `pstruct`.

Reference page in Doc Center
doc ssppstruct

Class methods

codim Compute the codimension of a skew-symmetric matrix pencil.

codim(StructObj) determines the codimension of the orbit of the skew-symmetric matrix pencil structure object StructObj. The codimension is determined with respect to the represented structure not the tangent space.

codim(StructObj,Strata) where the optional argument Strata can be
'orbit' Determines the codimension of the orbit.
 Assumes that all eigenvalues are specified. (default)
 'bundle' Determines the codimension of the bundle.
 Assumes that all (finite and infinite) eigenvalues are unspecified.
 'semibundle' Determines the codimension of the specified structure, i.e., the codimension is computed as codim(orbit) - (number of distinct unspecified eigenvalues).

Remark:
  codim(bundle) = codim(orbit) - (number of distinct eigenvalues)

See also `sspcodim`.

sspstruct Create a skew-symmetric matrix pencil structure object.

sspstruct creates a canonical structure object representing a skew-symmetric matrix pencil $G - sH$ in its canonical form.

A sspstruct object can consist of the following canonical blocks:
smblock - Singular SM block defined as:
$$SM_n := \begin{pmatrix}
0 & G_n \\
0 & F_n \\
\end{pmatrix}$$
where
$$G_n := \begin{pmatrix}
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{pmatrix}$$

shblock - SH block associated with a finite or unspecified eigenvalues $\mu$, defined as:
$$SH_n := \begin{pmatrix}
0 & J_n(\mu) \\
0 & I_n \\
\end{pmatrix}$$
where
$$J_n(\mu) := \begin{pmatrix}
0 & 1 & 0 & \ldots \\
1 & 0 & 0 & \ldots \\
0 & 0 & 1 & \ldots \\
\end{pmatrix}$$

and
$$I_n := \begin{pmatrix}
1 & 0 & 0 & \ldots \\
0 & 1 & 0 & \ldots \\
\end{pmatrix}$$
where $J_n(\mu)$ is an $n$-by-$n$ Jordan block associated with the eigenvalue $\mu$.

$skblock$ - SK block associated with the infinite eigenvalue, defined as:

\[
\begin{pmatrix}
0 & I_n & 0 & J_n(0) & \cdots \\
-I_n & 0 & \cdots & J_n(0)^T & 0 \\
\end{pmatrix}
\]

where $J_n(0)$ is an $n$-by-$n$ Jordan block associated with the zero eigenvalue.

StructObj = `sspstruct()` returns an empty canonical structure object StructObj representing a skew-symmetric matrix pencil $G-sH$. The canonical structure information can be specified using one of the three forms 'Size vector form', Property-value form', or 'Object form', which are explained below.

**Size vector form**

StructObj = `sspstruct(SMBlocks,SHBlocksv)` returns a new canonical structure object StructObj representing a skew-symmetric matrix pencil. The parameter SMBlocks defines the singular SM blocks of the structure and is given as a row-vector with the sizes of the blocks. The parameter SHBlocksv defines the SH blocks associated with the finite or unspecified eigenvalues and must be either a row-vector with the indices of blocks, all associated with the same eigenvalue, or a cell-array of row-vectors where each vector contains the sizes of blocks associated with the same eigenvalue. By default the associated eigenvalues are set to unspecified (NaN).

StructObj = `sspstruct(SMBlocks,SHBlocksv,Eigv)` sets the eigenvalues, where Eigv is a row-vector with the values of each associated eigenvalue corresponding to the row-vectors in SHBlocksv. The SH blocks with an unspecified associated eigenvalue are defined by setting the corresponding eigenvalue in Eigv to NaN.

StructObj = `sspstruct(SMBlocks,SHBlocksv,Eigv,SKBlocks)` also sets the sizes of the SK blocks associated with the infinite eigenvalue.

**Property-value form**

StructObj =

`sspstruct('BlockName1',StructInt1,'BlockName2',StructInt2,...)` specifies the block types BlockName and the sizes StructInt in property-value form. BlockName is a string specifying the canonical block to be created, and can be one of:

- ‘smblock’ - Singular SM blocks.
- ‘shblock’ - SH blocks associated with finite eigenvalues.
- ‘skblock’ - SK blocks associated with the infinite eigenvalue.

SH blocks associated with a specified finite eigenvalue are given as a
cell-array tuple in the form \( \{\text{StructInt Eigenvalue}\} \).

Example:

```matlab
>> sspstr = sspstruct('shblock',[[10 3],[7]],'smblock',[3,2])
```

creates a pencil structure object with two SH blocks of the sizes 20x20 and 6x6 both with the eigenvalue 7, and two singular blocks of the sizes 7x7 and 5x5.

**SSPSTRUCT**('BlockName1',StructInt1,...,'Notation',Notation) also specifies the notation used for StructInt. Valid notations are:

- **'segre'** Indices are ordered in a non-increasing order.
- **'weyr'** Weyr characteristics.
- **'sizes'** Indices may be unordered. (default)

Object form
-------------

**StructObj = SSPSTRUCT**(BlockObj1,BlockObj2,...) creates a structure object from the listed canonical block objects. The block objects must be valid blocks for the structure.

Example:

```matlab
>> sspstr = sspstruct([2], [3 1], 4, [1])
```

returns a skew-symmetric matrix pencil structure with one SM block of size 5, one SH block of size 6 and one of size 2, both associated with eigenvalue 4, and one SK block of size 2 associated with the infinite eigenvalue.

Alternatively, the same structure can be created with

```matlab
>> sspstr = sspstruct('smblock',2,'skblock',1,'shblock',[3 1],4)
```

```matlab
>> sspstr = sspstruct([4], {[2] [1]}, [1 3], [2])
```

returns a skew-symmetric matrix pencil of the following form:

```
| SM4   0   0   0 |  
| 0   SH2(1)   0   0 |  
| 0   0   SH1(3)   0 |  
| 0   0   0   SK2 |
```

Subscripting
-------------

To access the canonical blocks in the structure object it is possible to use subscripts. See **subsref** for detail and examples. If more control is needed of what should be retrieved, use the method **get** instead.

The class **SSPSTRUCT** also provides methods for extracting information and modifying the canonical structure object.

**SSPSTRUCT** Methods:

- **codim** - Compute the codimension of a skew-symmetric matrix pencil.
kcf - Return the skew-symmetric matrix pencil in a
Kronecker-like canonical form (described above).
size - Total size of the represented structure.
numblk - Number of canonical block objects.
isempty - True for empty canonical structure.
char - Convert a structure object to a string.
exist - Check if a canonical block of a specified type
already exist.
copy - Return a copy of the structure object.
compare - Compare two canonical structure objects.
set - Set the canonical structure information (not implemented).
get - Get the canonical structure information.
getvalidblocks - Return a list of available canonical blocks.

**sspstruct Operators:**

==, ~= (eq, ne) - Check if two structure objects are equal.
(), {} (subsref) - Index reference for canonical structure objects.

See also smbloc, sbhblock, skblock, spstruct, pstruct.

### 4.9 SSSTRUCT

Create a state-space system structure object.

ssstruct creates a canonical structure object representing the system
pencil \( S-sT = [A B; C D] - s[I 0; 0 0] \) in its canonical form. The
system pencil \( S-sT \) is associated with the continuous-time state-space
model
\[
\frac{dx}{dt} = Ax(t) + Bu(t),
\]
\[
y = Cx(t) + Du(t).
\]

An ssstruct object can consist of the following canonical blocks:

- **rsblock** - Right singular block defined as:
  \[
  L_n := \begin{bmatrix}
  0 & 1 & 0 \\
  \vdots & \ddots & \ddots \\
  0 & 0 & 1 \\
  \end{bmatrix} - s
  \]
  \( n \times (n+1) \)

- **lsblock** - Left singular block defined as:
  \[
  L_n^T := \begin{bmatrix}
  1 & \vdots & 0 \\
  \vdots & \ddots & \vdots \\
  0 & \vdots & 1 \\
  \end{bmatrix} - s
  \]
  \( (n+1) \times n \)

- **fjblock** - Jordan block associated with a finite eigenvalue \( \mu \),
defined as:
  \[
  J_n(\mu) - sI_n := \begin{bmatrix}
  \mu & 1 & 0 \\
  \vdots & \ddots & \vdots \\
  0 & \vdots & \mu \\
  \end{bmatrix} - sI_n
  \]
  \( n \times n \)

- **zjblock** - Jordan block associated with the zero eigenvalue (\( \mu=0 \)).
ijblock - Jordan block associated with the infinite eigenvalue, defined as:

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
-1 & 0 & \ddots & 0 \\
0 & \ddots & \ddots & 1 \\
0 & \ddots & \ddots & 0 \\
\end{bmatrix}
\]

\( N_n := \begin{bmatrix} 1 & 0 & \cdots & 0 \\ -s & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & -s & 1 \end{bmatrix} \) \( n \times n \)

StructObj = \textit{ssstruct}() returns an empty canonical structure object \texttt{StructObj} representing the system pencil

\[
S-sT = [A \; B; \; C \; D] - s[I \; 0; \; 0 \; 0],
\]

associated with the continuous-time state-space model

\[
\frac{dx}{dt} = Ax(t) + Bu(t),
\]

\[
y(t) = Cx(t) + Du(t).
\]

The canonical structure information can be specified using one of the three forms 'Size vector form', Property-value form', or 'Object form', which are explained below.

Size vector form

-----------------

StructObj = \textit{ssstruct}(RSBlocks,LSBlocks,FJBlocksv) returns a new canonical structure object \texttt{StructObj} representing a system pencil. The parameter RSBlocks and LSBlocks define the right and left singular blocks (associated with the column and row minimal indices), respectively, of the structure and is given as row-vectors with the sizes of the blocks. The parameter FJBlocksv defines the Jordan blocks (associated with the finite elementary divisors) of the structure and must be either a row-vector with the sizes of Jordan blocks, all associated with the same eigenvalue, or a cell-array of row-vectors, each containing the sizes of blocks associated with the same eigenvalue. By default the associated eigenvalues are set to unspecified (NaN).

StructObj = \textit{ssstruct}(RSBlocks,LSBlocks,FJBlocksv,Eigv) also sets the eigenvalues, where \texttt{Eigv} is a row-vector with the values of each associated eigenvalue corresponding to the row-vectors in \texttt{FJBlocksv}. Jordan blocks with an unspecified associated eigenvalue are defined by setting the corresponding eigenvalue in \texttt{Eigv} to NaN.

StructObj = \textit{ssstruct}(RSBlocks,LSBlocks,FJBlocksv,Eigv,IJBlocks) also sets the sizes of the Jordan blocks with an associated infinite eigenvalue (associated with the infinite elementary divisors).

Property-value form

---------------

StructObj = \textit{ssstruct}('BlockName1',StructInt1,'BlockName2',StructInt2,...) specifies the block types \texttt{BlockName} and the sizes \texttt{StructInt} in property-value form. \texttt{BlockName} is a string specifying the canonical block to be created, and can be one of:
'rsblock' - Right singular blocks
'lsblock' - Left singular blocks
'fjblock' - Jordan blocks associated with finite eigenvalues
'zjblock' - Jordan blocks associated with the zero eigenvalue
'ijblock' - Jordan blocks associated with the infinite eigenvalue

Jordan blocks associated with a specified finite eigenvalue are given as a cell-array tuple in the form {StructInt Eigenvalue}.

Example:
>> ssstr = ssstruct('fjblock',[[3 2],[1]],'fjblock',[1,-3])
creates a state-space structure object with one Jordan block of size 3x3 and one of size 2x2 both with eigenvalue 0, and one Jordan block of size 1x1 with eigenvalue -3.

ssstruct('BlockName1',StructInt1,...,'Notation',Notation) also specifies the notation used for StructInt. Valid notations are:
'segre' Sizes are ordered in a non-increasing order.
'weyr' Weyr characteristics.
'sizes' Sizes may be unordered. (default)

Object form
------------
StructObj = ssstruct(BlockObj1,BlockObj2,...) creates a structure object from the listed canonical block objects. The block objects must be valid blocks for the structure.

StructObj = ssstruct(Pstr) converts the matrix pencil structure object Pstr to a system pencil structure object with the same canonical structure information.

Examples:
>> ssstr = ssstruct([2],[0 1],[2],4)
returns a state-space structure with one right singular block of size 2x3, two left singular blocks of sizes 1x0 and 2x1, and one Jordan block of size 2x2 with eigenvalue 4.

Alternatively, the same structure can be created with
>> ssstr = ...
    ssstruct('rsblock',2,'lsblock',[0 1],'fjblock',[2,4])

>> ssstr = ssstruct('rsblock',3,'fjblock',[3,-1])
returns a controllability pair structure with one right singular block of size 3x4 and one Jordan block of size 3x3 with eigenvalue -1.

Subscripting
------------
To access the canonical blocks in the structure object it is possible to use subscripts. See subsref for detail and examples. If more control is needed of what should be retrieved, use the method get instead.
The class **ssstruct** provides the following methods for extracting information and modifying the canonical structure object.

**ssstruct** Methods:
- **codim** - Compute the codimension of a state-space model.
- **oftype** - State-space model type.
- **ctrbpair** - Controllability pair.
- **obsvpair** - Observability pair.
- **bcf** - Return the system pencil in the Brunovsky Canonical Form.
- **kcf** - Return the matrix pencil in the Kronecker Canonical Form.
- **size** - Total size of the represented structure.
- **numblk** - Number of canonical block objects.
- **isempty** - True for empty canonical structure.
- **char** - Convert a structure object to a string.
- **exist** - Check if a canonical block of a specified type already exist.
- **copy** - Return a copy of the structure object.
- **compare** - Compare two canonical structure objects.
- **set** - Set the canonical structure information (not implemented).
- **get** - Get the canonical structure information.
- **getvalidblocks** - Return a list of valid canonical blocks.
- **getsgconstraints** - Return a list of available StratiGraph constraints.

**ssstruct** Operators:
- **==, ~= (eq, ne)** - Check if two structure objects are equal.
- **( ), {} (subsref)** - Index reference for canonical structure objects.

See also **rsblock, lsblock, fjblock, zjblock, ijblock**.

Reference page in Doc Center
  doc ssstruct

**Class methods**

**codim** Compute the codimension of a state-space system pencil.

**codim(StructObj)** determines the codimension of the orbit of the state-space structure object **StructObj**. The codimension is determined with respect to the represented canonical structure not the tangent space.

**codim(StructObj, Strata)** where the optional argument **Strata** can be

- **'orbit'** Determines the codimension of the orbit.
  Assumes that all eigenvalues are specified. (default)
- **'bundle'** Determines the codimension of the bundle:
  - **codim(bundle) = codim(orbit)**
  - (number of distinct finite eigenvalues)
  Assumes that all finite eigenvalues are unspecified.
Infinite eigenvalues are still treated as specified.
'semibundle' Determines the codimension of the specified structure, i.e. the codimension is computed as
\[ \text{codim}(\text{bundle}) = \text{codim}(\text{orbit}) - \text{(number of distinct unspecified eigenvalues)} \]

See also \texttt{s2codim}.

\textbf{obsvpair} Observability pair:

PairObj = \texttt{obsvpair}(StructObj) returns a new state-space system structure object PairObj representing the observability pair \((A,C)\), i.e., the matrices \(B\) and \(D\) in the state-space model are assumed to be empty matrices.

\[[A,C] = \texttt{obsvpair}(\text{StructObj})\] returns the observability pair \((A,C)\) associated with the particular system
\[
\begin{align*}
\frac{dx}{dt} &= Ax(t), \\
y(t) &= Cx(t),
\end{align*}
\]
of the state-space model represented by the structure object StructObj. The matrix pair \((A,C)\) is returned in Brunovsky canonical form (\texttt{bcf}).

\textbf{note}! The new canonical form is determined exactly from the \texttt{bcf}, i.e., the pair \((A,C)\) is assumed to be in \texttt{bcf} and no computational routines are called.

See also \texttt{ctrbpair}.

\textbf{ctrbpair} Controllability pair:

PairObj = \texttt{ctrbpair}(StructObj) returns a new state-space system structure object PairObj representing the controllability pair \((A,B)\), i.e., the matrices \(C\) and \(D\) in the state-space model are assumed to be empty matrices.

\[[A,B] = \texttt{ctrbpair}(\text{StructObj})\] returns the controllability pair \((A,B)\) associated with the particular system
\[
\begin{align*}
\frac{dx}{dt} &= Ax(t) + Bu(t), \\
y(t) &= Cx(t),
\end{align*}
\]
of the state-space model represented by the structure object StructObj. The matrix pair \((A,B)\) is returned in Brunovsky canonical form (\texttt{bcf}).

\textbf{note}! The new canonical form is determined exactly from the \texttt{bcf}, i.e., the pair \((A,B)\) is assumed to be in \texttt{bcf} and no computational routines are called.

See also \texttt{obsvpair}.

\textbf{oftype} State-space model type.

Type = \texttt{oftype}(StructObj) returns a string Type indicating the non-empty matrices \((A, B, C, \text{and } D)\) in the state-space model.
\[
\begin{align*}
\dot{x}(t) &= Ax(t) + Bu(t), \\
y(t) &= Cx(t) + Du(t).
\end{align*}
\] (1)

The type is determined from the canonical structure information.

Type can be:

'a' - StructObj represents a state-space model with only a state matrix \( A \). The state-space structure consists only of Jordan blocks with finite eigenvalues (associated with the finite elementary divisors).

'ab' - StructObj represents a controllability pair \((A,B)\) associated with the particular system
\[
\dot{x}(t) = Ax(t) + Bu(t),
\]
of (1). The state-space structure consists only of right singular blocks (associated with the column minimal indices) and Jordan blocks with finite eigenvalues (associated with the finite elementary divisors).

'ac' - StructObj represents an observability pair \((A,C)\) associated with the particular system
\[
\dot{x}(t) = Ax(t),
y(t) = Cx(t),
\]
of (1). The state-space structure consists only of left singular blocks (associated with the row minimal indices) and Jordan blocks with finite eigenvalues (associated with the finite elementary divisors).

'abc' - StructObj represents a state-space model without any feed-forward, i.e., \( D \) is a zero matrix. The state-space structure do not have any Jordan blocks associated with the infinite eigenvalue of size 1-by-1 (no infinite elementary divisors of order 1).

'abcd' - StructObj represents the full state-space model in (1).

' ' (empty string) - StructObj is an empty object.

**size** Total size of the represented structure.

\[D = \text{size}(\text{StructObj})\] returns a two-element row vector \( D = [M, N] \) containing the number of rows \( M \) and columns \( N \) of the system pencil structure, i.e., the sum of the sizes of all the canonical blocks.

\[[M,N] = \text{size}(\text{StructObj})\] returns the number of rows and columns in separate output variables.

**size**(StructObj,Dim) returns the length of the dimension specified by the scalar Dim. For example, **size**(StructObj,1) returns the number of rows.

\[[N,M,P] = \text{size}(\text{StructObj})\] returns the sizes of the matrices in the
state-space model. The returned sizes are the sizes of the N-by-N matrix A (number of states N), the N-by-M matrix B (number of inputs M), and the P-by-N matrix C (number of outputs P).

**ssstruct** Create a state-space system structure object.

**ssstruct** creates a canonical structure object representing the system pencil \( S-sT = [A \ B; \ C \ D] - s[I \ 0; \ 0 \ 0] \) in its canonical form. The system pencil \( S-sT \) is associated with the continuous-time state-space model

\[
\begin{align*}
\frac{dx}{dt} &= Ax(t) + Bu(t), \\
y &= Cx(t) + Du(t).
\end{align*}
\]

An **ssstruct** object can consist of the following canonical blocks:

- **rsblock** - Right singular block defined as:

\[
L_n := \begin{bmatrix}
0 & 1 & 0 \\
& \ddots & \ddots & \ddots \\
0 & 0 & 1
\end{bmatrix} \quad \text{n-by-(n+1)}
\]

- **lsblock** - Left singular block defined as:

\[
L_n^T := \begin{bmatrix}
1 & \ddots & \ddots & 0 \\
& \ddots & \ddots & \ddots \\
& & 0 & 1 \\
& & & 0 & 0
\end{bmatrix} \quad \text{(n+1)-by-n}
\]

- **fjblock** - Jordan block associated with a finite eigenvalue \( \mu \), defined as:

\[
J_n(\mu) - sI_n := \begin{bmatrix}
\mu & 1 & 0 \\
& \ddots & \ddots & \ddots \\
& & \mu & 1 \\
0 & & & \mu
\end{bmatrix} \quad \text{n-by-n}
\]

- **zjblock** - Jordan block associated with the zero eigenvalue (\( \mu=0 \)).

- **ijblock** - Jordan block associated with the infinite eigenvalue, defined as:

\[
N_n := \begin{bmatrix}
1 & \ddots & \ddots & 0 \\
& \ddots & \ddots & \ddots \\
& & 0 & 1 \\
& & & 0 & 0
\end{bmatrix} \quad \text{n-by-n}
\]

**StructObj = ssstruct()** returns an empty canonical structure object **StructObj** representing the system pencil

\[
S-sT = [A \ B; \ C \ D] - s[I \ 0; \ 0 \ 0],
\]

associated with the continuous-time state-space model

\[
\begin{align*}
\frac{dx}{dt} &= Ax(t) + Bu(t), \\
y(t) &= Cx(t) + Du(t).
\end{align*}
\]

The canonical structure information can be specified using one of the three forms 'Size vector form', 'Property-value form', or 'Object form', which are explained below.
Size vector form
----------------

StructObj = \texttt{ssstruct}(\text{RSBlocks}, \text{LSBlocks}, \text{FJBlocksv}) \text{ returns a new}
\text{canonical structure object} \text{StructObj} \text{ representing a system pencil. The}
\text{parameter RSBlocks and LSBlocks define the right and left singular}
\text{blocks (associated with the column and row minimal indices),}
\text{respectively, of the structure and is given as row-vectors with the}
\text{sizes of the blocks. The parameter FJBlocksv defines the Jordan blocks}
\text{(associated with the finite elementary divisors) of the structure and}
\text{must be either a row-vector with the sizes of Jordan blocks, all}
\text{associated with the same eigenvalue, or a cell-array of row-vectors,}
\text{each containing the sizes of blocks associated with the same}
eigenvalue. By default the associated eigenvalues are set to
\text{unspecified (NaN)}.}

StructObj = \texttt{ssstruct}(\text{RSBlocks}, \text{LSBlocks}, \text{FJBlocksv}, \text{Eigv}) \text{ also sets the}
eigenvalues, where Eigv is a row-vector with the values of each
\text{associated eigenvalue corresponding to the row-vectors in FJBlocksv.}
\text{Jordan blocks with an unspecified associated eigenvalue are defined by}
\text{setting the corresponding eigenvalue in Eigv to NaN.}

StructObj = \texttt{ssstruct}(\text{RSBlocks}, \text{LSBlocks}, \text{FJBlocksv}, \text{Eigv}, \text{IJBlocks}) \text{ also}
\text{sets the sizes of the Jordan blocks with an associated infinite}
eigenvalue (associated with the infinite elementary divisors).

Property-value form
---------------------

StructObj = \texttt{ssstruct}(\text{BlockName1}, \text{StructInt1}, \text{BlockName2}, \text{StructInt2},...)
\text{specifies}
\text{the block types BlockName and the sizes StructInt in property-value}
\text{form. BlockName is a string specifying the canonical block to be}
\text{created, and can be one of:}
\text{'rsblock' - Right singular blocks}
\text{'lsblock' - Left singular blocks}
\text{'fjblock' - Jordan blocks associated with finite eigenvalues}
\text{'zjblock' - Jordan blocks associated with the zero eigenvalue}
\text{'ijblock' - Jordan blocks associated with the infinite}
eigenvalue
\text{Jordan blocks associated with a specified finite eigenvalue are given}
as a cell-array tuple in the form \{StructInt, Eigenvalue\}.

Example:
>> \texttt{ssstr = ssstruct(}'fjblock',{[3 2],0},'fjblock',{[1], -3})
\text{creates a state-space structure object with one Jordan block of}
\text{size 3x3 and one of size 2x2 both with eigenvalue 0, and one Jordan}
\text{block of size 1x1 with eigenvalue -3.}

\texttt{ssstruct('BlockName1',StructInt1,...,'Notation',Notation)} \text{also}
specifies the notation used for StructInt. Valid notations are:
\text{'segre' \text{- Sizes are ordered in a non-increasing order.}}
'weyr'   Weyr characteristics.
'sizes'   Sizes may be unordered. (default)

Object form
------------
StructObj = ssstruct(BlockObj1,BlockObj2,...) creates a structure object from the listed canonical block objects. The block objects must be valid blocks for the structure.

StructObj = ssstruct(Pstr) converts the matrix pencil structure object Pstr to a system pencil structure object with the same canonical structure information.

Examples:

>> ssstr = ssstruct([2],[0 1],[2],4)
returns a state-space structure with one right singular block of size 2x3, two left singular blocks of sizes 1x0 and 2x1, and one Jordan block of size 2x2 with eigenvalue 4.

Alternatively, the same structure can be created with
>> ssstr = ...
    ssstruct('rsblock',2,'lsblock',[0 1],'fjblock',{2,4})

>> ssstr = ssstruct('rsblock',3,'fjblock',{3,-1})
returns a controllability pair structure with one right singular block of size 3x4 and one Jordan block of size 3x3 with eigenvalue -1.

Subscripting
------------
To access the canonical blocks in the structure object it is possible to use subscripts. See subsref for detail and examples. If more control is needed of what should be retrieved, use the method get instead.

The class ssstruct provides the following methods for extracting information and modifying the canonical structure object.

ssstruct Methods:
- codim - Compute the codimension of a state-space model.
- oftype - State-space model type.
- ctrbpair - Controllability pair.
- obsvpair - Observability pair.
- bcf - Return the system pencil in the Brunovsky Canonical Form.
- kcf - Return the matrix pencil in the Kronecker Canonical Form.
- size - Total size of the represented structure.
- numblk - Number of canonical block objects.
- isempty - True for empty canonical structure.
- char - Convert a structure object to a string.
- exist - Check if a canonical block of a specified type already exist.
copy - Return a copy of the structure object.
compare - Compare two canonical structure objects.
set - Set the canonical structure information (not implemented).
get - Get the canonical structure information.
getvalidblocks - Return a list of valid canonical blocks.
getsgconstraints - Return a list of available StratiGraph constraints.

ssstruct Operators:
==, ~= (eq, ne) - Check if two structure objects are equal.
(,), {} (subsref) - Index reference for canonical structure objects.

See also rsblock, lsblock, fjbloc, zjblock, ijblock.
5  Canonical Block Objects

5.1  MCSBLOCK

Abstract class for generic canonical blocks.

\texttt{mcsblock} provides a template and generic methods for canonical block objects.

\texttt{mcsblock} Methods:
- \texttt{size} - Total size of the represented canonical blocks.
- \texttt{numblk} - Number of canonical blocks.
- \texttt{copy} - Return a copy of the block object.
- \texttt{compare} - Compare two block objects.
- \texttt{isempty} - True for empty canonical block object.
- \texttt{issingular} - True for singular canonical block object.
- \texttt{isregular} - True for regular canonical block object.
- \texttt{set} - Set the canonical structure information.
- \texttt{get} - Get the canonical structure information.
- \texttt{sizes} - Canonical block sizes.
- \texttt{segre} - Segre characteristics.
- \texttt{weyr} - Weyr characteristics.
- \texttt{char} - Convert a block object to a string.
- \texttt{block2cell} - Convert a block object to a cell array of strings.

\texttt{mcsblock} Operators:
- \texttt{==, \sim (eq, ne)} - Check if two block objects are equal.

See also \texttt{mcsstruct}.

Reference page in Doc Center
\begin{verbatim}
doc mcsblock
\end{verbatim}

Class methods

\texttt{weyr} Weyr characteristics.

\texttt{weyr(BlockObj)} returns the canonical block sizes represented in Weyr characteristics.

See also \texttt{sizes, segre}.

\texttt{segre} Segre characteristics.

\texttt{segre(BlockObj)} returns the canonical block sizes represented in Segre characteristics (non-increasing order).

See also \texttt{sizes, weyr}.

\texttt{sizes} Canonical block sizes.
sizes(BlockObj) returns the canonical block sizes unordered (in the same order as created).

See also segre, weyr.

get Get the canonical structure information.

Blockv = get(BlockObj,'StructInt',Notation) or
Blockv = get(BlockObj,Notation) returns the structure integer partition Blockv. The optional Notation specify in which notation the structure integer partition Blockv is represented in. Valid values of Notation are 'sizes' (default), 'segre', or 'weyr'.

See also set.

set Set the canonical structure information.

set(BlockObj,'StructInt',Blockv) or
set(BlockObj,'StructInt',Blockv,Notation) sets the structure integer partition to Blockv. Notation specify in which notation the structure integer partition Blockv is represented in. Valid values of Notation are 'sizes' (default), 'segre', or 'weyr'.

set(BlockObj,Blockv,Notation) is a compact form which sets the canonical block structure to the structure integer partition Blockv in notation Notation, where Notation can be omitted.

See also get.

block2cell Convert a block object to a cell array of strings.

Out = block2cell(BlockObj,Format) returns the canonical structure information as strings in the cell array Out where Format can be 'block', 'segre', 'segrec', 'weyr', or 'sizes'. If Format is 'block' (for canonical block), Out is N-by-1, where N is the number of canonical blocks and Out{k} returns the canonical block structure, k = 1,...,N. Otherwise, Out is 1-by-3, where Out{1,1} returns the label, Out{1,2} the structure integer partition (as Segre-type, Weyr-type, or as unordered sizes), and Out{1,3} the total dimension.

If Format is 'segrec' a compact variant of 'segre' is used where multiple blocks of the same size is written as n*..., where n is the number of blocks.

See also char.

disp Display a canonical block object.

disp(BlockObj) is called for the canonical block object BlockObj when the semicolon is not used to terminate a statement. Returns the canonical form represented as a string.
**char** Convert a block object to a string.

\[ S = \text{char}(\text{BlockObj}) \]
returns the canonical structure information of the block object `BlockObj` as a string in canonical block notation.

See also `mcsstruct/char`.

**compare** Compare two block objects.

\[ C = \text{compare}(B1,B2) \]
compares the two block objects `B1` and `B2` of the same class, and returns 0 if equal, 1 if `B1` > `B2`, and -1 if `B1` < `B2`. The comparison is done by comparing the sizes of the canonical blocks. First the largest block from each object are compared then the second largest until one is larger than the other or no more blocks exist. Any parameters are ignored.

\[ C = \text{compare}(B1,B2,'weyr') \]
uses the Weyr characteristics resulting in that the comparison is done by comparing the number of blocks. First the number of all blocks are compared then the number of second smallest and larger blocks until one quantity is larger than the other.

\[ [C2, D] = \text{compare}(B1,B2,Tol) \]
or
\[ [C2, D] = \text{compare}(B1,B2,Tol,'weyr') \]
returns the row vector `C2 = [C,P]`, where `C` is the result from above and `P` is the result from comparing the parameters (eigenvalues). It uses the tolerance `Tol` for comparing the parameters. Parameter `p1` from `B1` is assumed to be equal to `p2` from `B2` if \( \text{abs}(p1-p2) \leq Tol \), larger if `p1` > `p2`, otherwise smaller. For blocks with no parameter `P` is always 0. The optional `D` returns the difference `abs(p1-p2)` between the two parameters, or `NaN` if the blocks have no parameter.

\[ C = \text{compare}(B1,B2,'size') \]
only compares the total sizes of the block objects. Returns 0 if equal, 1 if `S1` > `S2`, and -1 if `S1` < `S2`.

See also `eq`.

**isregular** True for regular canonical block object.

\[ \text{isregular}(\text{BlockObj}) \]
returns logical 1 (true) if `BlockObj` represents a regular canonical block and logical 0 (false) otherwise.

**issingular** True for singular canonical block object.

\[ \text{issingular}(\text{BlockObj}) \]
returns logical 1 (true) if `BlockObj` represents a singular canonical block and logical 0 (false) otherwise.

**isempty** True for empty canonical block object.

\[ \text{isempty}(\text{BlockObj}) \]
returns logical 1 (true) if `BlockObj` is an empty canonical block object and logical 0 (false) otherwise. An empty block object has an empty structure integer partition.
numblk Number of canonical blocks.

\[ N = \text{numblk}(\text{BlockObj}) \] returns the number of canonical blocks in the block object BlockObj.

See also size, mcsstruct/numblk.

size Total size of the represented canonical blocks.

\[ D = \text{size}(\text{BlockObj}) \] returns a two-element row vector \( D = [M, N] \) containing the number of rows and columns of the canonical block object BlockObj, i.e., the sum of the sizes of the represented canonical blocks.

\[ [M,N] = \text{size}(\text{BlockObj}) \] returns the number of rows and columns in separate output variables.

\[ \text{size}(\text{BlockObj}, \text{Dim}) \] returns the length of the dimension specified by the scalar Dim. For example, \( \text{size}(X,1) \) returns the number of rows.

ne (\( \sim \)) Not equal relation between two block objects.

\[ \text{Block1} \sim \text{Block2} \] checks if the two canonical block objects are not equal.

See also eq, compare.

eq (\( == \)) Equal relation between two block objects.

\[ \text{Block1} == \text{Block2} \] checks if the two canonical block objects are equal.

See also ne, compare.

5.2 FJBLOCK

Create a Jordan block object.

fjblock creates a canonical block object representing a Jordan block structure associated with a finite or unspecified eigenvalue.

Each n-by-n Jordan block associated with a finite eigenvalue \( \mu \) is defined as:

\[
\begin{bmatrix}
\mu & 1 & 0 \\
& \mu & \\
& & \ddots & 1 \\
& & & \mu
\end{bmatrix}
\]

for matrices and as \( J_n(\mu) - s*1_n \) for matrix pencils.

\[ \text{BlockObj} = \text{fjblock}(\text{JordanBlocks, Ev}) \] returns a new canonical block object BlockObj representing a structure of Jordan blocks. The Jordan
blocks are defined by the exponents \((h_n, \ldots, h_1)\) of the finite elementary divisors. The vector \(\text{JordanBlocks} = [h_n, \ldots, h_1]\) is the structure integer partition representing the (unordered) sizes \((h_k)\)-by-\((h_k)\) of the blocks and \(\text{Ev}\) is the associated eigenvalue. \(\text{Ev}\) can be a complex number or NaN (unspecified eigenvalue). If \(\text{Ev}\) is omitted, the eigenvalue is assumed to be unspecified.

\[
\text{BlockObj} = \text{fjblock}(\text{JordanBlocks}, \text{Ev}, \text{Notation})
\]
also specifies the notation used for \(\text{JordanBlocks}\). Valid notations are:

- 'segre' Sizes are ordered in a non-increasing order.
- 'weyr' Weyr characteristics.
- 'sizes' Sizes may be unordered. (default)

\[
\text{BlockObj} = \text{fjblock}
\]
returns an empty Jordan block object.

The class \text{fjblock} provides the following methods for extracting information and modifying the canonical block object.

\text{fjblock} Methods:

- \text{size} - Total size of the represented canonical blocks.
- \text{numblk} - Number of canonical blocks.
- \text{copy} - Return a copy of the block object.
- \text{compare} - Compare two block objects.
- \text{isempty} - True for empty canonical block object.
- \text{issingular} - True for singular canonical block object.
- \text{isregular} - True for regular canonical block object.
- \text{set} - Set the canonical structure information.
- \text{get} - Get the canonical structure information.
- \text{sizes} - Canonical block sizes.
- \text{segre} - Segre characteristics.
- \text{weyr} - Weyr characteristics.
- \text{jcf} - Jordan canonical form of the block object.
- \text{kcf} - Kronecker canonical form of the block object.
- \text{bcf} - Brunovsky canonical form of the block object.
- \text{char} - Convert a block object to a string.
- \text{block2cell} - Convert a block object to a cell array of strings.

\text{fjblock} Operators:

\[
==, \sim (\text{eq}, \text{ne}) \quad \text{Check if two block objects are equal.}
\]

See also \text{ijblock}, \text{zjblock}, \text{mstruct}, \text{pstruct}, \text{ssstruct}.

Reference page in Doc Center

doc fjblock

Class methods

\text{kcf} Kronecker canonical form of the block object.

\[
[G,H] = \text{kcf}(	ext{BlockObj})
\]
returns the square matrix pencil \(G-sH\) in
Kronecker Canonical Form (\texttt{kcf}) specified by the Jordan block object \texttt{BlockObj}. The matrix \( H \) is an identity matrix and \( G \) is block-diagonal where each block is in Jordan Canonical Form (\texttt{jcf}).

\[
J = \texttt{kcf}(\texttt{BlockObj}) \text{ returns the matrix } J \text{ in } \texttt{jcf}.
\]

\[
\ldots = \texttt{kcf}(\texttt{BlockObj},'\texttt{segre}') \text{ sorts the blocks in non-increasing block size order. By default are the blocks presented in the order they were created.}
\]

See also \texttt{jcf}, \texttt{ijblock/kcf}, \texttt{pstruct/kcf}.

\texttt{bcf} Brunovsky canonical form of the block object.

\[
A = \texttt{bcf}(\texttt{BlockObj}) \text{ returns the matrix } A \text{ of the system pencil } S-sT = [A; C D] - s[1 0; 0 0] \text{ in Brunovsky Canonical Form (\texttt{bcf}) specified by the Jordan block object \texttt{BlockObj} (the matrix } A \text{ is in Jordan canonical form). Consequently, the pencil } S-sT \text{ consists only of Jordan blocks associated with a finite eigenvalue (J blocks).}
\]

\[
[A,B,C,D] = \texttt{bcf}(\texttt{BlockObj}) \text{ also returns the empty matrices } B, C \text{ and } D \text{ of the system pencil } S-sT.
\]

\[
\ldots = \texttt{bcf}(\texttt{BlockObj},'\texttt{segre}') \text{ sorts the blocks in non-increasing block size order. By default are the blocks presented in the order they were created.}
\]

See also \texttt{kcf}, \texttt{jcf}.

\texttt{jcf} Jordan canonical form of the block object.

\[
J = \texttt{jcf}(\texttt{BlockObj}) \text{ returns the matrix } J \text{ in Jordan Canonical Form (\texttt{jcf}) specified by the Jordan block object \texttt{BlockObj}.}
\]

\[
\ldots = \texttt{jcf}(\texttt{BlockObj},'\texttt{segre}') \text{ sorts the blocks in non-increasing block size order. By default are the blocks presented in the order they were created.}
\]

See also \texttt{kcf}, \texttt{mstruct/jcf}.

\texttt{fjblock} Create a Jordan block object.

\texttt{fjblock} creates a canonical block object representing a Jordan block structure associated with a finite or unspecified eigenvalue.

Each \( n \)-by-\( n \) Jordan block associated with a finite eigenvalue \( \mu \) is defined as:

\[
J_n(\mu) := \begin{vmatrix}
\mu & 1 & 0 \\
\mu & . & 1 \\
0 & \mu
\end{vmatrix}
\]
for matrices and as $J_n(\mu) - sI_n$ for matrix pencils.

BlockObj = \texttt{fjblock}(\text{JordanBlocks, Ev})$ returns a new canonical block object \texttt{BlockObj} representing a structure of Jordan blocks. The Jordan blocks are defined by the exponents $(h_n, ..., h_1)$ of the finite elementary divisors. The vector \texttt{JordanBlocks} = $[h_n, ..., h_1]$ is the structure integer partition representing the (unordered) sizes $(h_k)$-by-$(h_k)$ of the blocks and \texttt{Ev} is the associated eigenvalue. \texttt{Ev} can be a complex number or NaN (unspecified eigenvalue). If \texttt{Ev} is omitted, the eigenvalue is assumed to be unspecified.

BlockObj = \texttt{fjblock}(\text{JordanBlocks, Ev, Notation}) also specifies the notation used for \texttt{JordanBlocks}. Valid notations are:
- \texttt{'segre'} Sizes are ordered in a non-increasing order.
- \texttt{'weyr'} Weyr characteristics.
- \texttt{'sizes'} Sizes may be unordered. (default)

BlockObj = \texttt{fjblock} returns an empty Jordan block object.

The class \texttt{fjblock} provides the following methods for extracting information and modifying the canonical block object.

\texttt{fjblock} Methods:
- \texttt{size} - Total size of the represented canonical blocks.
- \texttt{numblk} - Number of canonical blocks.
- \texttt{copy} - Return a copy of the block object.
- \texttt{compare} - Compare two block objects.
- \texttt{isempty} - True for empty canonical block object.
- \texttt{issingular} - True for singular canonical block object.
- \texttt{isregular} - True for regular canonical block object.
- \texttt{set} - Set the canonical structure information.
- \texttt{get} - Get the canonical structure information.
- \texttt{sizes} - Canonical block sizes.
- \texttt{segre} - Segre characteristics.
- \texttt{weyr} - Weyr characteristics.
- \texttt{jcf} - Jordan canonical form of the block object.
- \texttt{kcf} - Kronecker canonical form of the block object.
- \texttt{bcf} - Brunovsky canonical form of the block object.
- \texttt{char} - Convert a block object to a string.
- \texttt{block2cell} - Convert a block object to a cell array of strings.

\texttt{fjblock} Operators:
- \texttt{==, ~= (eq, ne)} - Check if two block objects are equal.

See also \texttt{ijblock}, \texttt{zjblock}, \texttt{mstruct}, \texttt{pstruct}, \texttt{ssstruct}.

5.3 GBLOCK

Create a Gamma block object.
**gblock** creates a canonical block object representing a Gamma block structure.

Each n-by-n Gamma block is defined as:

```
| 0   ... |
|        ... |
Gamma_n := | 1   1 |
| -1  -1 |
| 1   1  0 |
```

BlockObj = **gblock**(GBlocks) returns a new canonical block object BlockObj representing a structure of Gamma blocks. The vector GBlocks = [epsilon_n, ..., epsilon_1] is the structure integer partition representing the (unordered) sizes (epsilon_k)-by-(epsilon_k) of the blocks.

BlockObj = **gblock**(GBlocks, Notation) also specifies the notation used for GBlocks. Valid notations are:
- 'segre' Sizes are ordered in a non-increasing order.
- 'weyr' Weyr characteristics.
- 'sizes' Sizes may be unordered. (default)

BlockObj = **gblock** returns an empty Gamma block object.

The class **gblock** provides the following methods for extracting information and modifying the canonical block object.

**gblock** Methods:
- `size` - Total size of the represented canonical blocks.
- `numblk` - Number of canonical blocks.
- `copy` - Return a copy of the block object.
- `compare` - Compare two block objects.
- `isempty` - True for empty canonical block object.
- `issingular` - True for singular canonical block object.
- `isregular` - True for regular canonical block object.
- `set` - Set the canonical structure information.
- `get` - Get the canonical structure information.
- `sizes` - Canonical block sizes.
- `segre` - Segre characteristics.
- `weyr` - Weyr characteristics.
- `ccf` - Congruence canonical form of the block object.
- `char` - Convert a block object to a string.
- `block2cell` - Convert a block object to a cell array of strings.

**gblock** Operators:
- `==, ~= (eq, ne)` - Check if two block objects are equal.

See also **sgblock**, **cmstruct**.
Class methods

**ccf** Congruence canonical form of the block object.

\[ H = \text{ccf}(\text{BlockObj}) \]

returns the matrix \( H \) in Congruence Canonical Form (ccf) specified by the Gamma block object `BlockObj`. The matrix \( H \) only consists of Gamma blocks.

\[ \ldots = \text{ccf}(\text{BlockObj}, \text{'segre'}) \]

sorts the blocks in non-increasing block size order. By default are the blocks presented in the order they were created.

See also **cmstruct/ccf**.

**gblock** Create a Gamma block object.

\[ \text{gblock} \]

creates a canonical block object representing a Gamma block structure.

Each \( n \)-by-\( n \) Gamma block is defined as:

\[
\begin{pmatrix}
0 & \cdots \\
\vdots & \ddots \\
Gamma_n := & 1 & 1 \\
& -1 & -1 \\
1 & 1 & 0
\end{pmatrix}
\]

\[ \text{BlockObj} = \text{gblock}(\text{GBlocks}) \]

returns a new canonical block object \( \text{BlockObj} \) representing a structure of Gamma blocks. The vector \( \text{GBlocks} = [\text{epsilon_n, ..., epsilon_1}] \) is the structure integer partition representing the (unordered) sizes \((\text{epsilon_k})\)-by-\((\text{epsilon_k})\) of the blocks.

\[ \text{BlockObj} = \text{gblock}(\text{GBlocks, Notation}) \]

also specifies the notation used for \( \text{GBlocks} \). Valid notations are:

- 'segre' Sizes are ordered in a non-increasing order.
- 'weyr' Weyr characteristics.
- 'sizes' Sizes may be unordered. (default)

\[ \text{BlockObj} = \text{gblock} \]

returns an empty Gamma block object.

The class **gblock** provides the following methods for extracting information and modifying the canonical block object.

**gblock** Methods:

- **size** - Total size of the represented canonical blocks.
- **numblk** - Number of canonical blocks.
copy - Return a copy of the block object.
compare - Compare two block objects.
isempty - True for empty canonical block object.
issingular - True for singular canonical block object.
isregular - True for regular canonical block object.
set - Set the canonical structure information.
get - Get the canonical structure information.
sizes - Canonical block sizes.
segre - Segre characteristics.
weyr - Weyr characteristics.
ccf - Congruence canonical form of the block object.
char - Convert a block object to a string.
block2cell - Convert a block object to a cell array of strings.

gblock Operators:
==, ~= (eq, ne) - Check if two block objects are equal.

See also sgblock, cmstruct.

5.4 HBLOCK

Create a H block object.

**hblock** creates a canonical block object representing a symmetric H block structure associated with a finite or unspecified eigenvalue.

Each n-by-n symmetric H block associated with an eigenvalue mu is defined as:

\[
H_n(\mu) := \begin{bmatrix}
0 & \mu \\
\mu & 1 \\
\end{bmatrix} - s
\begin{bmatrix}
0 & 1 \\
1 & 0 \\
\end{bmatrix}
\]

BlockObj = hblock(HBlocks, Ev) returns a new canonical block object BlockObj representing a structure of symmetric H blocks ("anti-diagonal" Jordan blocks) associated with a finite or unspecified eigenvalue. The vector HBlocks = [h_n, ..., h_1] is the structure integer partition representing the (unordered) sizes (h_k)-by-(h_k) of the blocks and Ev is the associated eigenvalue. Ev can be a complex number or NaN (unspecified eigenvalue). If Ev is omitted, the eigenvalue is assumed to be unspecified.

BlockObj = hblock(HBlocks, Ev, Notation) also specifies the notation used for HBlocks. Valid notations are:

- 'segre' Sizes are ordered in a non-increasing order.
- 'weyr' Weyr characteristics.
- 'sizes' Sizes may be unordered. (default)

BlockObj = hblock returns an empty H block object.
The class **hblock** provides the following methods for extracting information and modifying the canonical block object.

**hblock Methods:**
- **size** - Total size of the represented canonical blocks.
- **numblk** - Number of canonical blocks.
- **copy** - Return a copy of the block object.
- **compare** - Compare two block objects.
- **isempty** - True for empty canonical block object.
- **issingular** - True for singular canonical block object.
- **isregular** - True for regular canonical block object.
- **set** - Set the canonical structure information.
- **get** - Get the canonical structure information.
- **sizes** - Canonical block sizes.
- **segre** - Segre characteristics.
- **weyr** - Weyr characteristics.
- **kcf** - Kronecker-like canonical form of the block object.
- **char** - Convert a block object to a string.
- **block2cell** - Convert a block object to a cell array of strings.

**hblock Operators:**
- **==, ~=** (eq, ne) - Check if two block objects are equal.

See also **shblock**, **spstruct**.

Reference page in Doc Center

doc hblock

**Class methods**

**kcf** Kronecker-like canonical form of the block object.

\[
[G,H] = \text{kcf}(\text{BlockObj}) \text{ returns the square symmetric matrix pencil } G-sH \\
\text{in Kronecker-like canonical form specified by the H block object} \\
\text{BlockObj.}\n\]

\[
... = \text{kcf}(\text{BlockObj},'segre') \text{ sorts the blocks in non-increasing block} \\
\text{size order. By default are the blocks presented in the order they were} \\
\text{created.}\n\]

See also **spstruct/kcf**.

**hblock** Create a H block object.

**hblock** creates a canonical block object representing a symmetric H block structure associated with a finite or unspecified eigenvalue.

Each n-by-n symmetric H block associated with an eigenvalue mu is defined as:
\[
H_n(\mu) := \begin{vmatrix}
\mu & 1 & & \\
& \mu & 1 & -s \\
& & \ddots & \ddots \\
& & & 1 & 1 \\
\end{vmatrix}
\]

BlockObj = \texttt{hblock}(\texttt{HBlocks}, \texttt{Ev}) returns a new canonical block object \texttt{BlockObj} representing a structure of symmetric H blocks ("anti-diagonal" Jordan blocks) associated with a finite or unspecified eigenvalue. The vector \texttt{HBlocks} = [h_n, ..., h_1] is the structure integer partition representing the (unordered) sizes \((h_k)\)-by-\((h_k)\) of the blocks and \texttt{Ev} is the associated eigenvalue. \texttt{Ev} can be a complex number or NaN (unspecified eigenvalue). If \texttt{Ev} is omitted, the eigenvalue is assumed to be unspecified.

BlockObj = \texttt{hblock}(\texttt{HBlocks}, \texttt{Ev}, \texttt{Notation}) also specifies the notation used for \texttt{HBlocks}. Valid notations are:

- `'segre'` Sizes are ordered in a non-increasing order.
- `'weyr'` Weyr characteristics.
- `'sizes'` Sizes may be unordered. (default)

BlockObj = \texttt{hblock} returns an empty H block object.

The class \texttt{hblock} provides the following methods for extracting information and modifying the canonical block object.

\texttt{hblock} Methods:
- `size` - Total size of the represented canonical blocks.
- `numblk` - Number of canonical blocks.
- `copy` - Return a copy of the block object.
- `compare` - Compare two block objects.
- `isempty` - True for empty canonical block object.
- `issingular` - True for singular canonical block object.
- `isregular` - True for regular canonical block object.
- `set` - Set the canonical structure information.
- `get` - Get the canonical structure information.
- `sizes` - Canonical block sizes.
- `segre` - Segre characteristics.
- `weyr` - Weyr characteristics.
- `kcf` - Kronecker-like canonical form of the block object.
- `char` - Convert a block object to a string.
- `block2cell` - Convert a block object to a cell array of strings.

\texttt{hblock} Operators:
- `==, \sim` (eq, ne) - Check if two block objects are equal.

See also \texttt{shblock}, \texttt{spstruct}.
5.5 **IJBLOCK**

Create a Jordan block object associated with the infinite eigenvalue.

**ijblock** creates a canonical block object representing a Jordan block structure associated with the infinite eigenvalue.

Each n-by-n Jordan block associated with the infinite eigenvalue is defined as:

\[
N_n := \begin{bmatrix}
1 & 0 & \cdots & 0 \\
0 & 1 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 1
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 0 \\
& 1 & 0 \\
& & \ddots & \ddots \\
& & & 1
\end{bmatrix}
\]

BlockObj = **ijblock**(IJJordanBlocks) returns a new canonical block object BlockObj representing a structure of Jordan blocks associated with the infinite eigenvalue. The Jordan blocks are defined by the exponents \((s_n, ..., s_1)\) of the infinite elementary divisors. The vector IJordanBlocks = \([s_n, ..., s_1]\) is the structure integer partition representing the (unordered) sizes \((s_k)\)-by-\((s_k)\) of the blocks.

BlockObj = **ijblock**(IJJordanBlocks, Notation) also specifies the notation used for IJordanBlocks. Valid notations are:

- `segre` Sizes are ordered in a non-increasing order.
- `weyr` Weyr characteristics.
- `sizes` Sizes may be unordered. (default)

BlockObj = **ijblock** returns an empty Jordan block object.

The class **ijblock** provides the following methods for extracting information and modifying the canonical block object.

**ijblock** Methods:

- `size` - Total size of the represented canonical blocks.
- `numblk` - Number of canonical blocks.
- `copy` - Return a copy of the block object.
- `compare` - Compare two block objects.
- `isempty` - True for empty canonical block object.
- `issingular` - True for singular canonical block object.
- `isregular` - True for regular canonical block object.
- `set` - Set the canonical structure information.
- `get` - Get the canonical structure information.
- `sizes` - Canonical block sizes.
- `segre` - Segre characteristics.
- `weyr` - Weyr characteristics.
- `kcf` - Kronecker canonical form of the block object.
- `bcf` - Brunovsky canonical form of the block object.
- `char` - Convert a block object to a string.
- `block2cell` - Convert a block object to a cell array of strings.

**ijblock** Operators:
==, ~= (eq, ne) - Check if two block objects are equal.

See also fjblock, zjblock, mstruct, pstruct, ssstruct.

Reference page in Doc Center
doc ijblock

Class methods

cf Kronecker canonical form of the block object.

\[ [G,H] = \text{kcf}(\text{BlockObj}) \] returns the matrix pencil \( G-sH \) in Kronecker Canonical Form (\text{kcf}) specified by the Jordan block object \text{BlockObj} associated with the infinite eigenvalue. The matrix \( G \) is an identity matrix and \( H \) is in Jordan canonical form (\text{jcf}).

... = \text{kcf}(\text{BlockObj},'segre') sorts the blocks in non-increasing block size order. By default are the blocks presented in the order they were created.

See also fjblock/kcf, pstruct/kcf.

bcf Brunovsky canonical form of the block object.

\[ [A,B,C,D] = \text{bcf}(\text{BlockObj}) \] returns the matrix quadruple \( (A,B,C,D) \) of the system pencil \( S-sT = [A B; C D] - s[I 0; 0 0] \) in Brunovsky Canonical Form (\text{bcf}) specified by the Jordan block object \text{BlockObj} associated with the infinite eigenvalue. Consequently, the pencil \( S-sT \) consists only of Jordan blocks associated with a infinite eigenvalue (\( N \) blocks).

... = \text{bcf}(\text{BlockObj},'segre') sorts the blocks in non-increasing block size order. By default are the blocks presented in the order they were created.

See also kcf, jcf.

ijblock Create a Jordan block object associated with the infinite eigenvalue.

\text{ijblock} creates a canonical block object representing a Jordan block structure associated with the infinite eigenvalue.

Each \( n \)-by-\( n \) Jordan block associated with the infinite eigenvalue is defined as:

\[
N_n := \begin{vmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
\cdot & 0 & 1 \\
\end{vmatrix} - s \begin{vmatrix}
1 \\
0 \\
0 \\
\end{vmatrix}
\]

\text{BlockObj} = \text{ijblock}([\text{IJordanBlocks}]) returns a new canonical block object
BlockObj representing a structure of Jordan blocks associated with the infinite eigenvalue. The Jordan blocks are defined by the exponents \((s_n, \ldots, s_1)\) of the infinite elementary divisors. The vector \(\text{IJordanBlocks} = [s_n, \ldots, s_1]\) is the structure integer partition representing the (unordered) sizes \((s_k)\)-by-\((s_k)\) of the blocks.

BlockObj = \text{ijblock}(\text{IJordanBlocks}, \text{Notation}) also specifies the notation used for \(\text{IJordanBlocks}\). Valid notations are:

- 'segre' Sizes are ordered in a non-increasing order.
- 'weyr' Weyr characteristics.
- 'sizes' Sizes may be unordered. (default)

BlockObj = \text{ijblock} returns an empty Jordan block object.

The class \text{ijblock} provides the following methods for extracting information and modifying the canonical block object.

\text{ijblock} Methods:
- \text{size} - Total size of the represented canonical blocks.
- \text{numblk} - Number of canonical blocks.
- \text{copy} - Return a copy of the block object.
- \text{compare} - Compare two block objects.
- \text{isempty} - True for empty canonical block object.
- \text{issingular} - True for singular canonical block object.
- \text{isregular} - True for regular canonical block object.
- \text{set} - Set the canonical structure information.
- \text{get} - Get the canonical structure information.
- \text{sizes} - Canonical block sizes.
- \text{segre} - Segre characteristics.
- \text{weyr} - Weyr characteristics.
- \text{kcf} - Kronecker canonical form of the block object.
- \text{bcf} - Brunovsky canonical form of the block object.
- \text{char} - Convert a block object to a string.
- \text{block2cell} - Convert a block object to a cell array of strings.

\text{ijblock} Operators:
- ==, ~== (eq, ne) - Check if two block objects are equal.

See also \text{fjblock}, \text{zjblock}, \text{mstruct}, \text{pstruct}, \text{ssstruct}.

### 5.6 KBLOCK

Create a \(K\) block object associated with the infinite eigenvalue.

\text{kbblock} creates a canonical block object representing a symmetric \(K\) block structure associated with the infinite eigenvalue.

Each \(n\)-by-\(n\) symmetric \(K\) block is defined as:

\[
K_n := \begin{bmatrix}
0 & 1 \\
1 & -s & 0 & 1
\end{bmatrix}
\]
BlockObj = \texttt{kblock(KBlocks)} returns a new canonical block object representing a structure of symmetric K blocks ("anti-diagonal" Jordan blocks) associated with the infinite eigenvalue. The vector KBlocks = [s_n, ..., s_1] is the structure integer partition representing the (unordered) sizes (s_k)-by-(s_k) of the blocks.

BlockObj = \texttt{kblock(SKBlocks, Notation)} also specifies the notation used for KBlocks. Valid notations are:
- 'segre'  Sizes are ordered in a non-increasing order.
- 'weyr' Weyr characteristics.
- 'sizes' Sizes may be unordered. (default)

BlockObj = \texttt{kblock} returns an empty K block object.

The class \texttt{kblock} provides the following methods for extracting information and modifying the canonical block object.

\texttt{kblock} Methods:
- \texttt{size} - Total size of the represented canonical blocks.
- \texttt{numblk} - Number of canonical blocks.
- \texttt{copy} - Return a copy of the block object.
- \texttt{compare} - Compare two block objects.
- \texttt{isempty} - True for empty canonical block object.
- \texttt{issingular} - True for singular canonical block object.
- \texttt{isregular} - True for regular canonical block object.
- \texttt{set} - Set the canonical structure information.
- \texttt{get} - Get the canonical structure information.
- \texttt{sizes} - Canonical block sizes.
- \texttt{segre} - Segre characteristics.
- \texttt{weyr} - Weyr characteristics.
- \texttt{kcf} - Kronecker-like canonical form of the block object.
- \texttt{char} - Convert a block object to a string.
- \texttt{block2cell} - Convert a block object to a cell array of strings.

\texttt{kblock} Operators:
- \texttt{==, ~=} (\texttt{eq, ne}) - Check if two block objects are equal.

See also \texttt{skblock}, \texttt{spstruct}.

Reference page in Doc Center
doc kblock

\textbf{Class methods}

\texttt{kcf} Kronecker-like canonical form of the block object.

\[ [G,H] = \texttt{kcf}(\texttt{BlockObj}) \text{ returns the square symmetric matrix pencil } G-sH \]
in Kronecker-like canonical form specified by the K block object
BlockObj associated with the infinite eigenvalue.

... = kcf(BlockObj,'segre') sorts the blocks in non-increasing block
group size order. By default are the blocks presented in the order they were
created.

See also spstruct/kcf.

kblock Create a K block object associated with the infinite eigenvalue.

kblock creates a canonical block object representing a symmetric K
type block structure associated with the infinite eigenvalue.

Each n-by-n symmetric K block is defined as:

\[
K_n := \begin{pmatrix}
0 & 1 & 0 & 0 \\
1 & -s & 0 & 1 \\
1 & 0 & 0 & 1 \\
. & . & . & . \\
\end{pmatrix}
\]

BlockObj = kblock(KBlocks) returns a new canonical block object
BlockObj representing a structure of symmetric K blocks ("anti-
diagonal" Jordan blocks) associated with the infinite eigenvalue. The
vector KBlocks = [s_n, ..., s_1] is the structure integer partition
representing the (unordered) sizes (s_k)-by-(s_k) of the blocks.

BlockObj = kblock(SKBlocks, Notation) also specifies the notation used
for KBlocks. Valid notations are:

'segre'    Sizes are ordered in a non-increasing order.
'weyr'     Weyr characteristics.
'sizes'    Sizes may be unordered. (default)

BlockObj = kblock returns an empty K block object.

The class kblock provides the following methods for extracting
information and modifying the canonical block object.

kblock Methods:
sizes    - Canonical block sizes.
segre    - Segre characteristics.
weyr     - Weyr characteristics.
kcf - Kronecker-like canonical form of the block object.
char - Convert a block object to a string.
block2cell - Convert a block object to a cell array of strings.

kblock Operators:
==, ~= (eq, ne) - Check if two block objects are equal.

See also skblock, spstruct.

5.7 LSBLOCK

Create a left singular block object.

lsblock creates a canonical block object representing a left singular block structure.

Each (n+1)-by-n left singular block is defined as:
\[
\begin{bmatrix}
0 & 0 & 1 & 0 \\
1 & 0 & s & 0 \\
. & 0 & . & 1 \\
. & 1 & 0 & 0
\end{bmatrix}
\]

BlockObj = lsblock(LSBlocks) returns a new canonical block object representing a structure of left singular blocks (L^T blocks). The left singular blocks are defined by the row (left) minimal indices (eta_n, ..., eta_1), and the vector LSBlocks = [eta_n, ..., eta_1] is the structure integer partition representing the (unordered) sizes (eta_k+1)-by-(eta_k) of the blocks.

BlockObj = lsblock(LSBlocks, Notation) also specifies the notation used for LSBlocks. Valid notations are:
'segre' Sizes are ordered in a non-increasing order.
'weyr' Weyr characteristics.
'sizes' Sizes may be unordered. (default)

BlockObj = lsblock returns an empty left singular block object.

The class lsblock provides the following methods for extracting information and modifying the canonical block object.

lsblock Methods:
size - Total size of the represented canonical blocks.
numblk - Number of canonical blocks.
copy - Return a copy of the block object.
compare - Compare two block objects.
isempty - True for empty canonical block object.
issingular - True for singular canonical block object.
isregular - True for regular canonical block object.
set - Set the canonical structure information.
get - Get the canonical structure information.
sizes - Canonical block sizes.
segr - Segre characteristics.
weyr - Weyr characteristics.
kcf - Kronecker canonical form of the block object.
bcf - Brunovsky canonical form of the block object.
char - Convert a block object to a string.
block2cell - Convert a block object to a cell array of strings.

lsblock Operators:
==, ~= (eq, ne) - Check if two block objects are equal.

See also rsblock, pstruct, ssstruct.

Reference page in Doc Center
doc lsblock

Class methods

kcf Kronecker canonical form of the block object.

\[ [G,H] = \text{kcf}(\text{BlockObj}) \]
returns the matrix pencil \( G-sH \) in Kronecker Canonical Form (kcf) specified by the left singular block object \( \text{BlockObj} \). The pencil \( G-sH \) only consists of left singular blocks (L^T blocks).

\[ \ldots = \text{kcf}(\text{BlockObj},'segr') \]
sorts the blocks in non-increasing block size order. By default are the blocks presented in the order they were created.

See also pstruct/kcf.

bcf Brunovsky canonical form of the block object.

\[ [A,C] = \text{bcf}(\text{BlockObj}) \]
returns the pair of matrices \( (A,C) \) of the system pencil \( S-sT = [A; C] - s[I; 0] \) in Brunovsky Canonical Form (bcf) specified by the left singular block object \( \text{BlockObj} \). Consequently, the pencil \( S-sT \) consists only of left singular blocks (L^T blocks).

\[ [A,B,C,D] = \text{bcf}(\text{BlockObj}) \]
also returns the empty matrices \( B \) and \( D \) of the system pencil \( S-sT = [A B; C D] - s[I 0; 0 0] \).

\[ \ldots = \text{bcf}(\text{BlockObj},'segr') \]
sorts the blocks in non-increasing block size order. By default are the blocks presented in the order they were created.

See also kcf, ssstruct/bcf.

lsblock Create a left singular block object.

\text{lsblock} \] creates a canonical block object representing a left singular
block structure.

Each (n+1)-by-n left singular block is defined as:
\[
L_n^T := \begin{bmatrix}
0 & 0 \\
1 & 0 \\
. & 0 \\
. & 1 \\
0 & 1 \\
\end{bmatrix}
\]

BlockObj = \text{lsblock}(\text{LSBlocks}) returns a new canonical block object
BlockObj representing a structure of left singular blocks (L^T blocks).
The left singular blocks are defined by the row (left) minimal indices
(\text{eta}_n, \ldots, \text{eta}_1), and the vector LSBlocks = [\text{eta}_n, \ldots, \text{eta}_1] is
the structure integer partition representing the (unordered) sizes
(\text{eta}_{k+1})-by-(\text{eta}_k) of the blocks.

BlockObj = \text{lsblock}(\text{LSBlocks}, \text{Notation}) also specifies the notation used
for LSBlocks. Valid notations are:
- 'segre' Sizes are ordered in a non-increasing order.
- 'weyr' Weyr characteristics.
- 'sizes' Sizes may be unordered. (default)

BlockObj = \text{lsblock} returns an empty left singular block object.

The class \text{lsblock} provides the following methods for extracting
information and modifying the canonical block object.

\text{lsblock} Methods:
- \text{size} - Total size of the represented canonical blocks.
- \text{numblk} - Number of canonical blocks.
- \text{copy} - Return a copy of the block object.
- \text{compare} - Compare two block objects.
- \text{isempty} - True for empty canonical block object.
- \text{issingular} - True for singular canonical block object.
- \text{isregular} - True for regular canonical block object.
- \text{set} - Set the canonical structure information.
- \text{get} - Get the canonical structure information.
- \text{sizes} - Canonical block sizes.
- \text{segre} - Segre characteristics.
- \text{weyr} - Weyr characteristics.
- \text{kcf} - Kronecker canonical form of the block object.
- \text{bcf} - Brunovsky canonical form of the block object.
- \text{char} - Convert a block object to a string.
- \text{block2cell} - Convert a block object to a cell array of strings.

\text{lsblock} Operators:
- \&= - Check if two block objects are equal.

See also \text{rsblock}, \text{pstruct}, \text{ssstruct}. 
5.8 MBLOCK

Create a singular M block object.

\textit{mblock} creates a canonical block object representing a symmetric singular M block structure.

Each \((2n+1)\times(2n+1)\) symmetric M block is defined as:

\[
M_n := \begin{bmatrix} 0 & G_n^T & 0 \\ G_n & 0 & F_n \\ 0 & F_n^T & 0 \end{bmatrix},
\]

where

\[
G_n := \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \quad F_n := \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix},
\]

\text{note: G}_n - sF_n forms a right singular block \(L_n\), and \(G_n^T - sF_n^T\) a left singular block \(L_n^T\).

BlockObj = \textit{mblock}(MBlocks) returns a new canonical block object representing a structure of symmetric M blocks (pairs of singular blocks). The vector \(\text{MBlocks} = [\epsilon_n, ..., \epsilon_1]\) is the structure integer partition representing the (unordered) sizes \((2*\epsilon_k+1)\times(2*\epsilon_k+1)\) of the blocks.

BlockObj = \textit{mblock}(MBlocks, Notation) also specifies the notation used for \(\text{MBlocks}\). Valid notations are:

'\text{segre}' Indices are ordered in a non-increasing order.

'\text{weyr}' Weyr characteristics.

'\text{sizes}' Indices may be unordered. (default)

BlockObj = \textit{mblock} returns an empty M block object.

The class \textit{mblock} provides the following methods for extracting information and modifying the canonical block object.

\textit{mblock} Methods:

- \text{size} - Total size of the represented canonical blocks.
- \text{numblk} - Number of canonical blocks.
- \text{copy} - Return a copy of the block object.
- \text{compare} - Compare two block objects.
- \text{isempty} - True for empty canonical block object.
- \text{issingular} - True for singular canonical block object.
- \text{isregular} - True for regular canonical block object.
- \text{set} - Set the canonical structure information.
- \text{get} - Get the canonical structure information.
- \text{sizes} - Canonical block sizes.
- \text{segre} - Segre characteristics.
- \text{weyr} - Weyr characteristics.
- \text{kcf} - Kronecker-like canonical form of the block object.
- \text{char} - Convert a block object to a string.
block2cell - Convert a block object to a cell array of strings.

mblock Operators:
  ==, ~= (eq, ne) - Check if two block objects are equal.

See also smbloc, spstruct.

Reference page in Doc Center
doc mblock

Class methods

kcf Kronecker-like canonical form of the block object.

\[ [G,H] = \text{kcf}(\text{BlockObj}) \]
returns the symmetric matrix pencil \( G-sH \) in the
Kronecker-like canonical form specified by the singular M block object
\text{BlockObj}. The pencil \( G-sH \) only consists of singular blocks (SM blocks).

\[ \ldots = \text{kcf}(\text{BlockObj},'\text{segre}') \]
sorts the blocks in non-increasing block size order. By default are the blocks presented in the order they were created.

See also spstruct/kcf.

mblock Create a singular M block object.

\[ \text{mblock} \]
creates a canonical block object representing a symmetric singular M block structure.

Each \((2n+1)\times(2n+1)\) symmetric M block is defined as:

\[
M_n := \begin{bmatrix}
0 & G_n^T & 0 \\
G_n & 0 & F_n^T \\
-F_n & 0 & 0
\end{bmatrix}
\]

where

\[
G_n := \begin{bmatrix}
0 & 1 & 0 \\
0 & 0 & 1 \\
1 & 0 & 0
\end{bmatrix}, \quad \text{and} \quad F_n := \begin{bmatrix}
. & . & . \\
. & . & . \\
0 & 0 & 1 \\
0 & 1 & 0
\end{bmatrix}
\]

\[ \text{note: } G_n - sF_n \text{ forms a right singular block } L_n, \text{ and} \]
\[ G_n^T - sF_n^T \text{ a left singular block } L_n^T. \]

BlockObj = \text{mblock}(\text{MBlocks}) \] returns a new canonical block object
BlockObj representing a structure of symmetric M blocks (pairs of singular blocks). The vector \text{MBlocks} = [\text{epsilon}_n, \ldots, \text{epsilon}_1] is
the structure integer partition representing the (unordered) sizes
\((2*\text{epsilon}_k+1)\times(2*\text{epsilon}_k+1)\) of the blocks.

BlockObj = \text{mblock(MBlocks, Notation)} also specifies the notation used
for \text{MBlocks}. Valid notations are:

'segre' Indices are ordered in a non-increasing order.
'weyr'    Weyr characteristics.
'sizes'   Indices may be unordered. (default)

BlockObj = mblock returns an empty M block object.

The class mblock provides the following methods for extracting information and modifying the canonical block object.

mblock Methods:
size     - Total size of the represented canonical blocks.
numblk   - Number of canonical blocks.
copy     - Return a copy of the block object.
compare  - Compare two block objects.
isempty  - True for empty canonical block object.
issingular - True for singular canonical block object.
isregular - True for regular canonical block object.
set      - Set the canonical structure information.
get      - Get the canonical structure information.
sizes    - Canonical block sizes.
segre    - Segre characteristics.
weyr     - Weyr characteristics.
kcf      - Kronecker-like canonical form of the block object.
char     - Convert a block object to a string.
block2cell - Convert a block object to a cell array of strings.

mblock Operators:
==, ~= (eq, ne) - Check if two block objects are equal.

See also smblock, spstruct.

5.9 PBLOCK

Create an abstract parametric block object.

pblock is an abstract canonical block class representing a parametric block structure.

The class pblock also provides generic methods for extracting information and modifying the canonical block object.

See also mcsblock.

Reference page in Doc Center
doc pblock

Class methods

get Get the canonical structure information.
\( V = \text{get}(\text{BlockObj}, \text{PropertyName}) \) returns the value of the specified property for the canonical block object \( \text{BlockObj} \). \( \text{PropertyName} \) can be 'StructInt', 'Eigenvalue', or 'Parameter'. 'StructInt' returns the structure integer partition in Sizes notation, and 'Eigenvalue' or 'Parameter' returns the associated parameter (eigenvalue).

\( \text{Blockv} = \text{get}(\text{BlockObj}, \text{'StructInt'}, \text{Notation}) \) also specifies the notation of the returned vector \( \text{Blockv} \). Valid values of \( \text{Notation} \) are 'segre', 'weyr', and 'sizes' (default).

\[ [\text{Blockv}, P] = \text{get}(\text{BlockObj}, \text{Notation}) \] returns the structure integer partition with notation \( \text{Notation} \) in the vector \( \text{Blockv} \), and the associated parameter in \( P \). If \( \text{Notation} \) is omitted, 'sizes' is assumed.

See also \textit{set}.

**set** Set the canonical structure information.

\[ \text{set}(\text{BlockObj}, \text{PropertyName'}, \text{PropertyValue}) \] sets the value of the specified property for the canonical block object \( \text{BlockObj} \).

\[ \text{set}(\text{BlockObj}, \text{PropertyName1'}, \text{PropertyValue1'}, \text{PropertyName2'}, \text{PropertyValue2'}, ...) \] sets multiple property values with a single statement.

\( \text{PropertyName} \) can be 'StructInt', 'Eigenvalue', or 'Parameter'. 'StructInt' sets the structure integer partition, where \( \text{PropertyValue} \) is a vector of the block sizes. Either 'Eigenvalue' or 'Parameter' can be specified to set the associated parameter (eigenvalue).

\[ \text{set}(\text{BlockObj}, \text{StructInt'}, \text{PropertyValue'}, ..., \text{Notation}) \] also specifies in which notation the structure integer partition is represented in. Valid values of \( \text{Notation} \) are 'sizes' (default), 'segre', or 'weyr'. Specifying notation only make sense when 'StructInt' is also set.

\[ \text{set}(\text{BlockObj}, \text{Blockv'}, \text{P'}, \text{Notation}) \] is a compact form which sets the canonical block structure to the structure integer partition \( \text{Blockv} \) in notation \( \text{Notation} \) with the associated parameter \( \text{P} \). \( \text{P} \) and \( \text{Notation} \) can be omitted.

See also \textit{get}.

**compare** Compare two parameter block objects.

\[ C = \text{compare}(\text{B1}, \text{B2}) \] compares the two block objects \( \text{B1} \) and \( \text{B2} \) of the same class, and returns 0 if equal, 1 if \( \text{B1} > \text{B2} \), and -1 if \( \text{B1} < \text{B2} \). The comparison is done by comparing the sizes of the canonical blocks. First the largest block from each object are compared then the second largest until one is larger than the other or no more blocks exist. Any parameters are ignored.

\[ C = \text{compare}(\text{B1}, \text{B2'}, \text{'weyr'}) \] uses the Weyr characteristics resulting in
that the comparison is done by comparing the number of blocks. First
the number of all blocks are compared then the number of second
smallest and larger blocks until one quantity is larger than the other.

\[ [C_2, D] = \text{compare}(B_1, B_2, \text{Tol}) \] or
\[ [C_2, D] = \text{compare}(B_1, B_2, \text{Tol}, '\text{weyr}') \]
returns the row vector \( C_2 = [C, P] \),
where \( C \) is the result from above and \( P \) is the result from comparing the
parameters (eigenvalues). It uses the tolerance \( \text{Tol} \) for comparing the
parameters. Parameter \( p_1 \) from \( B_1 \) is assumed to be equal to \( p_2 \) from \( B_2 \)
if \( \text{abs}(p_1 - p_2) \leq \text{Tol} \), larger if \( p_1 > p_2 \), otherwise smaller. For blocks
with no parameter \( P \) is always 0. The optional \( D \) returns the difference
\( \text{abs}(p_1 - p_2) \) between the two parameters, or NaN if the blocks have no
parameter.

\[ C = \text{compare}(B_1, B_2, '\text{size}') \]
only compares the total sizes of the block
objects. Returns 0 if equal, 1 if \( S_1 > S_2 \), and -1 if \( S_1 < S_2 \).

See also \text{eq}.

\text{ne} (~=) Not equal relation between two block objects.

\( \text{Block1} \sim= \text{Block2} \) checks if the two canonical block objects are not
equal.

\text{eq} (==) Check if two block objects are equal.

\( \text{Block1} == \text{Block2} \) checks if the two canonical block objects are equal.
Note that for two blocks objects to be equal, those parameters must
also be exactly equal.

\text{pblock} Create an abstract parametric block object.

\text{pblock} is an abstract canonical block class representing a parametric
block structure.

The class \text{pblock} also provides generic methods for extracting
information and modifying the canonical block object.

See also \text{mcsblock}.

5.10 \text{RSBLOCK}

Create a right singular block object.

\text{rsblock} creates a canonical block object representing a right singular
block structure.

Each \( n \)-by-\((n+1) \) right singular block is defined as:

\[
L_n := \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} - s \begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}
\]
BlockObj = \texttt{rsblock}(\texttt{RSBlocks})\) returns a new canonical block object representing a structure of right singular blocks (L blocks). The right singular blocks are defined by the column (right) minimal indices (epsilon\(_n\), ..., epsilon\(_1\)), and the vector \texttt{RSBlocks} = [epsilon\(_n\), ..., epsilon\(_1\)] is the structure integer partition representing the (unordered) sizes (epsilon\(_k\))-by-(epsilon\(_k+1\)) of the blocks.

BlockObj = \texttt{rsblock}(\texttt{RSBlocks}, \texttt{Notation})\) also specifies the notation used for \texttt{RSBlocks}. Valid notations are:
- 'segre' Sizes are ordered in a non-increasing order.
- 'weyr' Weyr characteristics.
- 'sizes' Sizes may be unordered. (default)

BlockObj = \texttt{rsblock}\) returns an empty right singular block object.

The class \texttt{rsblock}\) provides the following methods for extracting information and modifying the canonical block object.

\texttt{rsblock Methods:}
- \texttt{size} - Total size of the represented canonical blocks.
- \texttt{numblk} - Number of canonical blocks.
- \texttt{copy} - Return a copy of the block object.
- \texttt{compare} - Compare two block objects.
- \texttt{isempty} - True for empty canonical block object.
- \texttt{issingular} - True for singular canonical block object.
- \texttt{isregular} - True for regular canonical block object.
- \texttt{set} - Set the canonical structure information.
- \texttt{get} - Get the canonical structure information.
- \texttt{sizes} - Canonical block sizes.
- \texttt{segre} - Segre characteristics.
- \texttt{weyr} - Weyr characteristics.
- \texttt{kcf} - Kronecker canonical form of the block object.
- \texttt{bcf} - Brunovsky canonical form of the block object.
- \texttt{char} - Convert a block object to a string.
- \texttt{block2cell} - Convert a block object to a cell array of strings.

\texttt{rsblock Operators:}
- ==, \sim (= \texttt{eq}, \texttt{ne}) - Check if two block objects are equal.

See also \texttt{lsblock}, \texttt{pstruct}, \texttt{ssstruct}.

Reference page in Doc Center:
\texttt{doc rsblock}
**kcf** Kronecker canonical form of the block object.

\[ [G,H] = \text{kcf}(\text{BlockObj}) \]

returns the matrix pencil \( G-sH \) in Kronecker Canonical Form (kcf) specified by the right singular block object BlockObj. The pencil \( G-sH \) only consists of right singular blocks (L blocks).

\[ ... = \text{kcf}(\text{BlockObj},'segre') \]

sorts the blocks in non-increasing block size order. By default are the blocks presented in the order they were created.

See also \text{pstruct}/kcf.

**bcf** Brunovsky canonical form of the block object.

\[ [A,B] = \text{bcf}(\text{BlockObj}) \]

returns the matrix pair \((A,B)\) of the system pencil \( S-sT = [A B] - s[I 0] \) in Brunovsky Canonical Form (bcf) specified by the right singular block object BlockObj. Consequently, the pencil \( S-sT \) consists only of right singular blocks (L blocks).

\[ [A,B,C,D] = \text{bcf}(\text{BlockObj}) \]

also returns the empty matrices \( C \) and \( D \) of the system pencil \( S-sT = [A B; C D] - s[I 0; 0 0] \).

\[ ... = \text{bcf}(\text{BlockObj},'segre') \]

sorts the blocks in non-increasing block size order. By default are the blocks presented in the order they were created.

See also kcf, ssstruct/bcf.

**rsblock** Create a right singular block object.

\text{rsblock} creates a canonical block object representing a right singular block structure.

Each \( n \)-by-\((n+1)\) right singular block is defined as:

\[
\begin{bmatrix}
0 & 1 & 0 \\
1 & 0 & 0 \\
& & -s \\
0 & 0 & 1
\end{bmatrix}
\]

\[
\text{L}_n := \begin{bmatrix}
0 & 1 & 0 \\
1 & 0 & 0 \\
\vdots & \vdots & \vdots \\
0 & 0 & 1
\end{bmatrix}
\]

\[ \text{BlockObj} = \text{rsblock}(\text{RSBlocks}) \]

returns a new canonical block object BlockObj representing a structure of right singular blocks (L blocks). The right singular blocks are defined by the column (right) minimal indices \((\epsilon_n, \ldots, \epsilon_1)\), and the vector \( \text{RSBlocks} = [\epsilon_n, \ldots, \epsilon_1] \) is the structure integer partition representing the (unordered) sizes \((\epsilon_k)\)-by-\((\epsilon_k+1)\) of the blocks.

\[ \text{BlockObj} = \text{rsblock}(\text{RSBlocks}, \text{Notation}) \]

also specifies the notation used for \( \text{RSBlocks} \). Valid notations are:

- 'segre' Sizes are ordered in a non-increasing order.
- 'weyr' Weyr characteristics.
'sizes'  Sizes may be unordered. (default)

BlockObj = rsblock returns an empty right singular block object.

The class rsblock provides the following methods for extracting
information and modifying the canonical block object.

rsblock Methods:
size       - Total size of the represented canonical blocks.
numblk     - Number of canonical blocks.
copy       - Return a copy of the block object.
compare    - Compare two block objects.
isempty    - True for empty canonical block object.
issingular - True for singular canonical block object.
isregular - True for regular canonical block object.
set        - Set the canonical structure information.
get        - Get the canonical structure information.
sizes      - Canonical block sizes.
segre      - Segre characteristics.
weyr       - Weyr characteristics.
kcf        - Kronecker canonical form of the block object.
bcf        - Brunovsky canonical form of the block object.
char       - Convert a block object to a string.
block2cell - Convert a block object to a cell array of strings.

rsblock Operators:
==, ~=(eq, ne)  - Check if two block objects are equal.

See also lsblock, pstruct, ssstruct.

5.11 SBLOCK

Abstract singular block class

sblock is an abstract canonical block class representing a singular
block structure.

See also rsblock, lsblock, mblock, smbblock, mcsblock.

Reference page in Doc Center
doc sblock

Class methods

sblock Abstract singular block class

sblock is an abstract canonical block class representing a singular
block structure.

See also rsblock, lsblock, mblock, smbblock, mcsblock.
5.12 SGBLOCK

Create a *Gamma block object.

\texttt{sgblock} creates a canonical block object representing a *Gamma block structure.

Each \( n \times n \) *Gamma block associated with a complex parameter \( \mu \) is defined as:

\[
\begin{bmatrix}
0 & \cdots & \\
& \ddots & \\
& & 1 & 1 \\
\end{bmatrix}
\]
\*Gamma_n(\mu) := \mu \begin{bmatrix}
0 & \cdots & \\
& \ddots & \\
& & -1 & -1 \\
\end{bmatrix}
\begin{bmatrix}
1 & 1 & 0 \\
\end{bmatrix}
\]

where \( |\mu| = 1 \).

BlockObj = \texttt{sgblock}(\texttt{SGBlocks}, P) returns a new canonical block object representing a structure of *Gamma blocks. The vector \( \texttt{SGBlocks} = [h_n, \ldots, h_1] \) is the structure integer partition representing the (unordered) sizes \((h_k)\times(h_k)\) of the blocks and \( P \) is the associated complex parameter. The parameter \( P \) must be on the complex unit circle, i.e., \( \text{abs}(P) = 1 \).

BlockObj = \texttt{sgblock}(\texttt{SGBlocks}, P, \texttt{Notation}) also specifies the notation used for \texttt{SGBlocks}. Valid notations are:

- 'segre' Sizes are ordered in a non-increasing order.
- 'weyr' Weyr characteristics.
- 'sizes' Sizes may be unordered. (default)

BlockObj = \texttt{sgblock} returns an empty *Gamma block object.

The class \texttt{sgblock} provides the following methods for extracting information and modifying the canonical block object.

\texttt{sgblock} Methods:

- \texttt{size} - Total size of the represented canonical blocks.
- \texttt{numblk} - Number of canonical blocks.
- \texttt{copy} - Return a copy of the block object.
- \texttt{compare} - Compare two block objects.
- \texttt{isempty} - True for empty canonical block object.
- \texttt{isregular} - True for regular canonical block object.
- \texttt{issingular} - True for singular canonical block object.
- \texttt{set} - Set the canonical structure information.
- \texttt{get} - Get the canonical structure information.
- \texttt{sizes} - Canonical block sizes.
- \texttt{segre} - Segre characteristics.
- \texttt{weyr} - Weyr characteristics.
- \texttt{ccf} - Congruence canonical form of the block object.
- \texttt{char} - Convert a block object to a string.
- \texttt{block2cell} - Convert a block object to a cell array of strings.
sgblock Operators:
==, ~=(eq, ne) - Check if two block objects are equal.

See also gblock, scmstruct.

Reference page in Doc Center
doc sgblock

Class methods

ccf Congruence canonical form of the block object.

H = ccf(BlockObj) returns the matrix H in Congruence Canonical Form (ccf) specified by the *Gamma block object BlockObj. The matrix H only consists of *Gamma blocks.

... = ccf(BlockObj,'segre') sorts the blocks in non-increasing block size order. By default are the blocks presented in the order they were created.

See also scmstruct/ccf.

sgblock Create a *Gamma block object.

sgblock creates a canonical block object representing a *Gamma block structure.

Each n-by-n *Gamma block associated with a complex parameter mu is defined as:

\[
\begin{bmatrix}
0 & \ldots \\
\vdots & \ddots \\
\end{bmatrix}
\]

\[\text{*Gamma}_n(\mu) := \begin{bmatrix}
\mu & 1 & 1 \\
-1 & -1 & \\
1 & 1 & 0 \\
\end{bmatrix}\]

where |\mu| = 1.

BlockObj = sgblock(SGBlocks, P) returns a new canonical block object BlockObj representing a structure of *Gamma blocks. The vector SGBlocks = [h_n, ..., h_1] is the structure integer partition representing the (unordered) sizes (h_k)-by-(h_k) of the blocks and P is the associated complex parameter. The parameter P must be on the complex unit circle, i.e., abs(P) = 1.

BlockObj = sgblock(SGBlocks, P, Notation) also specifies the notation used for SGBlocks. Valid notations are:

'segre' Sizes are ordered in a non-increasing order.
'weyr' Weyr characteristics.
'sizes' Sizes may be unordered. (default)

BlockObj = sgblock returns an empty *Gamma block object.
The class `sgblock` provides the following methods for extracting information and modifying the canonical block object.

**sgblock Methods:**
- `size` - Total size of the represented canonical blocks.
- `numblk` - Number of canonical blocks.
- `copy` - Return a copy of the block object.
- `compare` - Compare two block objects.
- `isempty` - True for empty canonical block object.
- `issingular` - True for singular canonical block object.
- `isregular` - True for regular canonical block object.
- `set` - Set the canonical structure information.
- `get` - Get the canonical structure information.
- `sizes` - Canonical block sizes.
- `segre` - Segre characteristics.
- `weyr` - Weyr characteristics.
- `ccf` - Congruence canonical form of the block object.
- `char` - Convert a block object to a string.
- `block2cell` - Convert a block object to a cell array of strings.

**sgblock Operators:**
- `==, ~=` (eq, ne) - Check if two block objects are equal.

See also `gblock`, `scmstruct`.

### 5.13 SHBLOCK

Create an SH block object.

`shblock` creates a canonical block object representing a skew-symmetric SH block structure associated with a finite or unspecified eigenvalue.

Each 2n-by-2n skew-symmetric SH block is defined as:

\[
\begin{pmatrix}
0 & J_n(\mu) \\
-J_n(\mu)^T & 0
\end{pmatrix}
\begin{pmatrix}
0 & I_n \\
-I_n & 0
\end{pmatrix}
\]

where \( J_n(\mu) \) is an \( n \times n \) Jordan block associated with the eigenvalue \( \mu \).

BlockObj = `shblock`(SHBlocks, Ev) returns a new canonical block object representing a structure of skew-symmetric SH blocks (pairs of Jordan blocks) associated with a finite or unspecified eigenvalue. The vector \( \text{SHBlocks} = [h_n, ..., h_1] \) is the structure integer partition representing the (unordered) sizes \( (2h_k) \)-by-\( (2h_k) \) of the blocks and \( \text{Ev} \) is the associated eigenvalue. \( \text{Ev} \) can be a complex number or NaN (unspecified eigenvalue). If \( \text{Ev} \) is omitted, the eigenvalue is assumed to be unspecified.

BlockObj = `shblock`(SHBlocks, Ev, Notation) also specifies the notation
used for SHBlocks. Valid notations are:

'vegre' Indices are ordered in a non-increasing order.
'veyr' Weyr characteristics.
'sizes' Indices may be unordered. (default)

BlockObj = shblock returns an empty SH block object.

The class shblock provides the following methods for extracting information and modifying the canonical block object.

**shblock Methods:**
- `size` - Total size of the represented canonical blocks.
- `numblk` - Number of canonical blocks.
- `copy` - Return a copy of the block object.
- `compare` - Compare two block objects.
- `isempty` - True for empty canonical block object.
- `issingular` - True for singular canonical block object.
- `isregular` - True for regular canonical block object.
- `set` - Set the canonical structure information.
- `get` - Get the canonical structure information.
- `sizes` - Canonical block sizes.
- `segre` - Segre characteristics.
- `weyr` - Weyr characteristics.
- `kcf` - Kronecker-like canonical form of the block object.
- `char` - Convert a block object to a string.
- `block2cell` - Convert a block object to a cell array of strings.

**shblock Operators:**
- `==, ~=` (eq, ne) - Check if two block objects are equal.

See also hblock, sspstruct.

Reference page in Doc Center
doc shblock

**Class methods**

**kcf** Kronecker-like canonical form of the block object.

\[ [G,H] = \text{kcf(BlockObj)} \] returns the square skew-symmetric matrix pencil \( G-sH \) in Kronecker-like canonical form specified by the SH block object BlockObj.

\( ... = \text{kcf(BlockObj,'segre') \} \) sorts the blocks in non-increasing block size order. By default are the blocks presented in the order they were created.

See also sspstruct/kcf.

**shblock** Create an SH block object.
**shblock** creates a canonical block object representing a skew-symmetric SH block structure associated with a finite or unspecified eigenvalue.

Each 2n-by-2n skew-symmetric SH block is defined as:

\[
\begin{pmatrix}
0 & J_n(\mu) \\
-J_n(\mu)^T & 0
\end{pmatrix}
\begin{pmatrix}
0 & I_n \\
-I_n & 0
\end{pmatrix}
\]

where \( J_n(\mu) \) is an n-by-n Jordan block associated with the eigenvalue \( \mu \).

BlockObj = **shblock**(SHBlocks, Ev) returns a new canonical block object BlockObj representing a structure of skew-symmetric SH blocks (pairs of Jordan blocks) associated with a finite or unspecified eigenvalue. The vector SHBlocks = \([h_n, \ldots, h_1]\) is the structure integer partition representing the (unordered) sizes \((2*h_k)\)-by-\((2*h_k)\) of the blocks and \(Ev\) is the associated eigenvalue. \(Ev\) can be a complex number or NaN (unspecified eigenvalue). If \(Ev\) is omitted, the eigenvalue is assumed to be unspecified.

BlockObj = **shblock**(SHBlocks, Ev, Notation) also specifies the notation used for SHBlocks. Valid notations are:

- 'segre'  \(\Rightarrow\) Indices are ordered in a non-increasing order.
- 'weyr'  \(\Rightarrow\) Weyr characteristics.
- 'sizes'  \(\Rightarrow\) Indices may be unordered. (default)

BlockObj = **shblock** returns an empty SH block object.

The class **shblock** provides the following methods for extracting information and modifying the canonical block object.

**shblock** Methods:

- **size** - Total size of the represented canonical blocks.
- **numblk** - Number of canonical blocks.
- **copy** - Return a copy of the block object.
- **compare** - Compare two block objects.
- **isempty** - True for empty canonical block object.
- **issingular** - True for singular canonical block object.
- **isregular** - True for regular canonical block object.
- **set** - Set the canonical structure information.
- **get** - Get the canonical structure information.
- **sizes** - Canonical block sizes.
- **segre** - Segre characteristics.
- **weyr** - Weyr characteristics.
- **kcf** - Kronecker-like canonical form of the block object.
- **char** - Convert a block object to a string.
- **block2cell** - Convert a block object to a cell array of strings.

**shblock** Operators:

- **==, ~== (eq, ne)** - Check if two block objects are equal.
See also hblock, sspstruct.

5.14 SKBLOCK

Create an SK block object associated with the infinite eigenvalue.

skblock creates a canonical block object representing a skew-symmetric SK block structure associated with the infinite eigenvalue.

Each 2n-by-2n skew-symmetric SK block is defined as:

\[
\begin{bmatrix}
0 & I_n & 0 & J_n(0) \\
-I_n & 0 & -J_n(0)^T & 0 \\
\end{bmatrix}
\]

where \( J_n(0) \) is an \( n \times n \) Jordan block associated with the zero eigenvalue.

BlockObj = skblock(SKBlocks) returns a new canonical block object
BlockObj representing a structure of skew-symmetric SK blocks (pairs of Jordan blocks) associated with the infinite eigenvalue. The vector

\( \text{SKBlocks} = [s_n, ..., s_1] \)

is the structure integer partition representing the (unordered) sizes \((2s_k)\times(2s_k)\) of the blocks.

BlockObj = skblock(SKBlocks, Notation) also specifies the
notation used for SKBlocks. Valid notations are:

- 'segre' Indices are ordered in a non-increasing order.
- 'weyr' Weyr characteristics.
- 'sizes' Indices may be unordered. (default)

BlockObj = skblock returns an empty SK block object.

The class skblock provides the following methods for extracting information and modifying the canonical block object.

skblock Methods:

- size - Total size of the represented canonical blocks.
- numblk - Number of canonical blocks.
- copy - Return a copy of the block object.
- compare - Compare two block objects.
- isempty - True for empty canonical block object.
- issingular - True for singular canonical block object.
- isregular - True for regular canonical block object.
- set - Set the canonical structure information.
- get - Get the canonical structure information.
- sizes - Canonical block sizes.
- segre - Segre characteristics.
- weyr - Weyr characteristics.
- kcf - Kronecker-like canonical form of the block object.
- char - Convert a block object to a string.
- block2cell - Convert a block object to a cell array of strings.
**skblock** Operators:

`==`, `~=` *(eq, ne)* - Check if two block objects are equal.

See also **kblock**, **sspstruct**.

Reference page in Doc Center

```
doc skblock
```

### Class methods

**kcf** Kronecker-like canonical form of the block object.

```
[G,H] = kcf(BlockObj) returns the matrix pencil G-sH in Kronecker-like
canonical form specified by the SK block object BlockObj associated
with the infinite eigenvalue.
```

```
... = kcf(BlockObj,'segre') sorts the blocks in non-increasing block
size order. By default are the blocks presented in the order they were
created.
```

See also **sspstruct/kcf**.

**skblock** Create an SK block object associated with the infinite eigenvalue.

```
skblock creates a canonical block object representing a skew-symmetric
SK block structure associated with the infinite eigenvalue.
```

Each 2n-by-2n skew-symmetric SK block is defined as:

```
\[
\begin{bmatrix}
0 & I_n \\
-I_n & 0
\end{bmatrix}
\begin{bmatrix}
0 & J_n(0) \\
-J_n(0)^T & 0
\end{bmatrix}
\]
```

where \( J_n(0) \) is an \( n \times n \) Jordan block associated with the zero
eigenvalue.

```
BlockObj = skblock(SKBlocks) returns a new canonical block object
BlockObj representing a structure of skew-symmetric SK blocks (pairs of
Jordan blocks) associated with the infinite eigenvalue. The vector
SKBlocks = \([s_n, ..., s_1]\) is the structure integer partition
representing the (unordered) sizes \((2s_k)\)-by-\((2s_k)\) of the blocks.
```

```
BlockObj = skblock(SKBlocks, Notation) also specifies the
notation used for SKBlocks. Valid notations are:
'segre' Indices are ordered in a non-increasing order.
'weyr' Weyr characteristics.
'sizes' Indices may be unordered. (default)
```

```
BlockObj = skblock returns an empty SK block object.
```
The class skblock provides the following methods for extracting information and modifying the canonical block object.

**skblock** Methods:
- `size` - Total size of the represented canonical blocks.
- `numblk` - Number of canonical blocks.
- `copy` - Return a copy of the block object.
- `compare` - Compare two block objects.
- `isempty` - True for empty canonical block object.
- `issingular` - True for singular canonical block object.
- `isregular` - True for regular canonical block object.
- `set` - Set the canonical structure information.
- `get` - Get the canonical structure information.
- `sizes` - Canonical block sizes.
- `segre` - Segre characteristics.
- `weyr` - Weyr characteristics.
- `kcf` - Kronecker-like canonical form of the block object.
- `char` - Convert a block object to a string.
- `block2cell` - Convert a block object to a cell array of strings.

**skblock** Operators:
- `==`, `~=` (eq, ne) - Check if two block objects are equal.

See also `kblock`, `sspstruct`.

### 5.15 SMBLOCK

Create a singular SM block object.

**smblock** creates a canonical block object representing a singular skew-symmetric SM block structure.

Each \((2n+1)\times(2n+1)\) skew-symmetric SM block is defined as:

\[
\begin{bmatrix}
0 & G_n & 0 \\
\end{bmatrix} \\
\begin{bmatrix}
0 & F_n \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
0 & G_n^T & 0 \\
\end{bmatrix} \\
\begin{bmatrix}
0 & F_n^T & 0 \\
\end{bmatrix}
\]

where

\[
G_n := \begin{bmatrix}
1 & 0 & 0 \\
\end{bmatrix} \\
\begin{bmatrix}
0 & 1 & 0 \\
\end{bmatrix}
\]

\[
F_n := \begin{bmatrix}
\end{bmatrix} \\
\begin{bmatrix}
\end{bmatrix}
\]

**note:** \(G_n - sF_n\) forms a right singular block \(L_n\), and \(G_n^T - sF_n^T\) a left singular block \(L_n^T\).

\[
\text{BlockObj} = \text{smblock(SMBlocks)} \text{ returns a new canonical block object}
\]

BlockObj representing a structure of skew-symmetric SM blocks (pairs of singular blocks). The vector `SMBlocks` = \([\epsilon_n, \ldots, \epsilon_1]\) is the structure integer partition representing the (unordered) sizes \((2*\epsilon_k+1)\times(2*\epsilon_k+1)\) of the blocks.

**BlockObj = smbblock(SMBlocks, Notation)** also specifies the notation used
for SMBlocks. Valid notations are:

'segre' Indices are ordered in a non-increasing order.
'weyr' Weyr characteristics.
'sizes' Indices may be unordered. (default)

BlockObj = smblock returns an empty SM block object.

The class smblock provides the following methods for extracting information and modifying the canonical block object.

smblock Methods:
size - Total size of the represented canonical blocks.
numblk - Number of canonical blocks.
copy - Return a copy of the block object.
compare - Compare two block objects.
isempty - True for empty canonical block object.
issingular - True for singular canonical block object.
isregular - True for regular canonical block object.
set - Set the canonical structure information.
get - Get the canonical structure information.
sizes - Canonical block sizes.
segre - Segre characteristics.
weyr - Weyr characteristics.
kcf - Kronecker-like canonical form of the block object.
char - Convert a block object to a string.
block2cell - Convert a block object to a cell array of strings.

smblock Operators:
==, ~= (eq, ne) - Check if two block objects are equal.

See also mblock, sspstruct.

Reference page in Doc Center
doc smblock

Class methods

kcf Kronecker-like canonical form of the block object.

[G,H] = kcf(BlockObj) returns the skew-symmetric matrix pencil G−sH in the Kronecker-like canonical form specified by the singular SM block object BlockObj. The pencil G−sH only consists of singular blocks (SM blocks).

... = kcf(BlockObj,'segre') sorts the blocks in non-increasing block size order. By default are the blocks presented in the order they were created.

See also sspstruct/kcf.
**smblock** Create a singular SM block object.

`smblock` creates a canonical block object representing a singular skew-symmetric SM block structure.

Each (2n+1)-by-(2n+1) skew-symmetric SM block is defined as:

\[
\begin{bmatrix}
0 & G_n & \cdots & 0 \\
G_n^T & 0 & \cdots & 0 \\
\vdots & \ddots & \ddots & \vdots \\
0 & \cdots & 0 & F_n
\end{bmatrix}
\]

\[
SM_n := \begin{bmatrix}
0 & -s \\
-G_n^T & 0 \\
-F_n^T & 0
\end{bmatrix}
\]

where

\[
G_n := \begin{bmatrix}
0 & 1 & 0 \\
1 & 0 & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

\[
F_n := \begin{bmatrix}
1 & \cdots & 0 \\
0 & \cdots & 1 \\
0 & \cdots & 0
\end{bmatrix}
\]

**note:** \(G_n - sF_n\) forms a right singular block \(L_n\), and
\(G_n^T - sF_n^T\) a left singular block \(L_n^T\).

BlockObj = `smblock(SMBlocks)` returns a new canonical block object representing a structure of skew-symmetric SM blocks (pairs of singular blocks). The vector `SMBlocks = [epsilon_n, ..., epsilon_1]` is the structure integer partition representing the (unordered) sizes (2*epsilon_k+1)-by-(2*epsilon_k+1) of the blocks.

BlockObj = `smblock(SMBlocks, Notation)` also specifies the notation used for `SMBlocks`. Valid notations are:

'`segre'` Indices are ordered in a non-increasing order.

'`weyr'` Weyr characteristics.

'`sizes'` Indices may be unordered. (default)

BlockObj = `smblock` returns an empty SM block object.

The class `smblock` provides the following methods for extracting information and modifying the canonical block object.

**smblock** Methods:

- **size** - Total size of the represented canonical blocks.
- **numblk** - Number of canonical blocks.
- **copy** - Return a copy of the block object.
- **compare** - Compare two block objects.
- **isempty** - True for empty canonical block object.
- **issingingular** - True for singular canonical block object.
- **isregular** - True for regular canonical block object.
- **set** - Set the canonical structure information.
- **get** - Get the canonical structure information.
- **sizes** - Canonical block sizes.
- **segre** - Segre characteristics.
- **weyr** - Weyr characteristics.
- **kcf** - Kronecker-like canonical form of the block object.
- **char** - Convert a block object to a string.
- **block2cell** - Convert a block object to a cell array of strings.
smblock Operators:
   ==, ~= (eq, ne) - Check if two block objects are equal.

See also mblock, ssstruct.

5.16 SWBLOCK

Create a *W block object.

swblock creates a canonical block object representing a *W block structure associated with a specified and admissible eigenvalue.

Each 2n-by-2n *W block associated with a finite eigenvalue mu is defined as:

\[
\begin{pmatrix}
0 & I_n \\
J_n(mu) & 0
\end{pmatrix}
\]

where |mu| > 1 and J_n(mu) is an n-by-n Jordan block with the associated eigenvalue mu.

BlockObj = swblock(SWBlocks, Ev) returns a new canonical block object BlockObj representing a structure of *W blocks associated with finite Jordan blocks. The vector SWBlocks = [h_n, ..., h_1] is the structure integer partition representing the (unordered) sizes (2*h_k)-by-(2*h_k) of the blocks and Ev is the associated eigenvalue. Ev must be a complex scalar with its complex modulus greater than one, i.e., abs(Ev) > 1.

BlockObj = swblock(SWBlocks, Ev, Notation) also specifies the notation used for SWBlocks. Valid notations are:

'segr'   Indices are ordered in a non-increasing order.
'weyr'   Weyr characteristics.
'sizes'  Indices may be unordered. (default)

BlockObj = swblock returns an empty *W block object.

The class swblock provides the following methods for extracting information and modifying the canonical block object.

swblock Methods:

size    - Total size of the represented canonical blocks.
numblk  - Number of canonical blocks.
copy    - Return a copy of the block object.
compare - Compare two block objects.
isempty - True for empty canonical block object.
isregular - True for regular canonical block object.
issingular - True for singular canonical block object.
set     - Set the canonical structure information.
get     - Get the canonical structure information.
sizes   - Canonical block sizes.
segr    - Segre characteristics.
weyr - Weyr characteristics.
ccf - Congruence canonical form of the block object.
char - Convert a block object to a string.
block2cell - Convert a block object to a cell array of strings.

swblock Operators:
==, ~= (eq, ne) - Check if two block objects are equal.

See also wblock, scmstruct.

Reference page in Doc Center
doc swblock

Class methods

swblock Create a *W block object.

swblock creates a canonical block object representing a *W block structure associated with a specified and admissible eigenvalue.

Each 2n-by-2n *W block associated with a finite eigenvalue mu is defined as:

\[
*W_n(mu) := \begin{bmatrix} 0 & I_n \\ J_n(mu) & 0 \end{bmatrix}
\]

where |mu| > 1 and J_n(mu) is an n-by-n Jordan block with the associated eigenvalue mu.

BlockObj = swblock(SWBlocks, Ev) returns a new canonical block object BlockObj representing a structure of *W blocks associated with finite Jordan blocks. The vector SWBlocks = [h_n, ..., h_1] is the structure integer partition representing the (unordered) sizes (2*h_k)-by-(2*h_k) of the blocks and Ev is the associated eigenvalue. Ev must be a complex scalar with its complex modulus greater than one, i.e., abs(Ev) > 1.

BlockObj = swblock(SWBlocks, Ev, Notation) also specifies the notation used for SWBlocks. Valid notations are:
- 'segre' Indices are ordered in a non-increasing order.
- 'weyr' Weyr characteristics.
- 'sizes' Indices may be unordered. (default)

BlockObj = swblock returns an empty *W block object.

The class swblock provides the following methods for extracting information and modifying the canonical block object.

swblock Methods:
size - Total size of the represented canonical blocks.
numbblk - Number of canonical blocks.
copy - Return a copy of the block object.
cmpare - Compare two block objects.
isempty - True for empty canonical block object.
issingular - True for singular canonical block object.
isregular - True for regular canonical block object.
set - Set the canonical structure information.
get - Get the canonical structure information.
sizes - Canonical block sizes.
segre - Segre characteristics.
weyr - Weyr characteristics.
ccf - Congruence canonical form of the block object.
char - Convert a block object to a string.
block2cell - Convert a block object to a cell array of strings.

swblock Operators:
==, ~= (eq, ne) - Check if two block objects are equal.

See also wbloc, scmstruct.

5.17 WBLOC

Create a W block object.

wbloc - creates a canonical block object representing a W block structure associated with a specified and admissible eigenvalue.

Each 2n-by-2n W block associated with a finite eigenvalue mu is defined as:

\[
W_n := \begin{bmatrix}
0 & I_n \\
J_n(mu) & 0
\end{bmatrix}
\]

where mu \sim \{0, (-1)^{(n+1)}\} and J_n(mu) is an n-by-n Jordan block with the associated eigenvalue mu.

BlockObj = wbloc(WBlocks, Ev) returns a new canonical block object
BlockObj representing a structure of W blocks associated with finite Jordan blocks. The vector WBlocks = [h_n, ..., h_1] is the structure integer partition representing the (unordered) sizes (2*\(h_k\))-by-(2*\(h_k\)) of the blocks and Ev is the associated eigenvalue. Ev must be a non-zero complex scalar and not equal to (-1)^\((\text{size of the block}/2 + 1)\) for all blocks.

BlockObj = wbloc(WBlocks, Ev, Notation) also specifies the notation used for WBlocks. Valid notations are:
'segre' Indices are ordered in a non-increasing order.
'weyr' Weyr characteristics.
'sizes' Indices may be unordered. (default)

BlockObj = wbloc returns an empty W block object.
The class `wblock` provides the following methods for extracting information and modifying the canonical block object.

**wblock Methods:**
- `size` - Total size of the represented canonical blocks.
- `numblk` - Number of canonical blocks.
- `copy` - Return a copy of the block object.
- `compare` - Compare two block objects.
- `isempty` - True for empty canonical block object.
- `issingular` - True for singular canonical block object.
- `isregular` - True for regular canonical block object.
- `set` - Set the canonical structure information.
- `get` - Get the canonical structure information.
- `sizes` - Canonical block sizes.
- `segre` - Segre characteristics.
- `weyr` - Weyr characteristics.
- `ccf` - Congruence canonical form of the block object.
- `char` - Convert a block object to a string.
- `block2cell` - Convert a block object to a cell array of strings.

**wblock Operators:**
- `==`, `~=` (eq, ne) - Check if two block objects are equal.

See also `swblock`, `cmstruct`.

Reference page in Doc Center
`doc wblock`

### Class methods

**ccf** Congruence canonical form of the block object.

\[ H = \text{ccf}(\text{BlockObj}) \] returns the square matrix \( H \) in Congruence Canonical Form (ccf) specified by the \( W \) block object \( \text{BlockObj} \).

... = \text{ccf}(\text{BlockObj},'\text{segre}') sorts the blocks in non-increasing block size order. By default are the blocks presented in the order they were created.

See also `zjblock/ccf`, `cmstruct/ccf`.

**wblock** Create a \( W \) block object.

\( \text{wblock} \) creates a canonical block object representing a \( W \) block structure associated with a specified and admissible eigenvalue.

Each 2n-by-2n \( W \) block associated with a finite eigenvalue \( \mu \) is defined as:

\[
\begin{bmatrix}
0 & I_n \\
I_n & 0
\end{bmatrix}
\]

\( W_n := \)

\[
\begin{bmatrix}
0 & I_n \\
I_n & 0
\end{bmatrix}
\]
\[ |J_n(\mu) \emptyset | \]

where \( \mu \sim \{0, (-1)^{(n+1)}\} \) and \( J_n(\mu) \) is an \( n \times n \) Jordan block with the associated eigenvalue \( \mu \).

BlockObj = \texttt{wblock}(\textit{WBlocks}, \textit{Ev}) returns a new canonical block object BlockObj representing a structure of \( W \) blocks associated with finite Jordan blocks. The vector \( \textit{WBlocks} = [h_n, ..., h_1] \) is the structure integer partition representing the (unordered) sizes \( (2h_k) \times (2h_k) \) of the blocks and \( \textit{Ev} \) is the associated eigenvalue. \( \textit{Ev} \) must be a non-zero complex scalar and not equal to \((-1)^{\text{size of the block}/2} + 1\) for all blocks.

BlockObj = \texttt{wblock}(\textit{WBlocks}, \textit{Ev}, \textit{Notation}) also specifies the notation used for \( \textit{WBlocks} \). Valid notations are:
- 'segre' Indices are ordered in a non-increasing order.
- 'weyr' Weyr characteristics.
- 'sizes' Indices may be unordered. (default)

BlockObj = \texttt{wblock} returns an empty \( W \) block object.

The class \texttt{wblock} provides the following methods for extracting information and modifying the canonical block object.

\texttt{wblock} Methods:
- \texttt{size} - Total size of the represented canonical blocks.
- \texttt{numblk} - Number of canonical blocks.
- \texttt{copy} - Return a copy of the block object.
- \texttt{compare} - Compare two block objects.
- \texttt{isempty} - True for empty canonical block object.
- \texttt{issingular} - True for singular canonical block object.
- \texttt{isregular} - True for regular canonical block object.
- \texttt{set} - Set the canonical structure information.
- \texttt{get} - Get the canonical structure information.
- \texttt{sizes} - Canonical block sizes.
- \texttt{segre} - Segre characteristics.
- \texttt{weyr} - Weyr characteristics.
- \texttt{ccf} - Congruence canonical form of the block object.
- \texttt{char} - Convert a block object to a string.
- \texttt{block2cell} - Convert a block object to a cell array of strings.

\texttt{wblock} Operators:
- \texttt{==}, \texttt{~=} (\texttt{eq}, \texttt{ne}) - Check if two block objects are equal.

See also \texttt{swblock}, \texttt{cmstruct}.

5.18 ZJBLOCK

Create a Jordan block object associated with the zero eigenvalue.

\texttt{zjblock} creates a canonical block object representing a Jordan block.
Each $n$-by-$n$ Jordan block associated with the zero eigenvalue is defined as:

$$J_n(0) := \begin{bmatrix} 0 & 1 & 0 \\ 0 & \ddots & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

BlockObj = \texttt{zjblock}(ZJordanBlocks) returns a new canonical block object BlockObj representing a structure of Jordan blocks associated with the zero eigenvalue. The vector $ZJordanBlocks = [s_n, \ldots, s_1]$ is the structure integer partition representing the (unordered) sizes $(s_k)$-by-$(s_k)$ of the blocks.

BlockObj = \texttt{zjblock}(ZJordanBlocks, Arg) also specifies the notation Arg used for $ZJordanBlocks$. Valid notation arguments are:

- 'segre' Sizes are ordered in a non-increasing order.
- 'weyr' Weyr characteristics.
- 'sizes' Sizes may be unordered. (default)

BlockObj = \texttt{zjblock} returns an empty Jordan block object.

The class \texttt{zjblock} provides the following methods for extracting information and modifying the canonical block object.

\texttt{zjblock} Methods:

- \texttt{size} - Total size of the represented canonical blocks.
- \texttt{numblk} - Number of canonical blocks.
- \texttt{copy} - Return a copy of the block object.
- \texttt{compare} - Compare two block objects.
- \texttt{isempty} - True for empty canonical block object.
- \texttt{issingular} - True for singular canonical block object.
- \texttt{isregular} - True for regular canonical block object.
- \texttt{set} - Set the canonical structure information.
- \texttt{get} - Get the canonical structure information.
- \texttt{sizes} - Canonical block sizes.
- \texttt{segre} - Segre characteristics.
- \texttt{weyr} - Weyr characteristics.
- \texttt{jcf} - Jordan canonical form of the block object.
- \texttt{ccf} - Congruence canonical form of the block object.
- \texttt{kcf} - Kronecker canonical form of the block object.
- \texttt{bcf} - Brunovsky canonical form of the block object.
- \texttt{char} - Convert a block object to a string.
- \texttt{block2cell} - Convert a block object to a cell array of strings.

\texttt{zjblock} Operators:

- \texttt{==, ~= (eq, ne)} - Check if two block objects are equal.

See also \texttt{fjblock, ijblock, cmstruct, scmstruct, mstruct}. 
Class methods

ccf Congruence canonical form of the block object.

H = ccf(BlockObj) returns the matrix H in the Congruence Canonical Form (ccf) specified by the Jordan block object BlockObj associated with the zero eigenvalue.

... = ccf(BlockObj,'segre') sorts the blocks in non-increasing block size order. By default are the blocks presented in the order they were created.

See also jcf, cmstruct/ccf, scmstruct/ccf.

zjblock Create a Jordan block object associated with the zero eigenvalue.

zjblock creates a canonical block object representing a Jordan block structure associated with the zero eigenvalue.

Each n-by-n Jordan block associated with the zero eigenvalue is defined as:

\[
J_n(0) := \begin{vmatrix} 0 & 1 & 0 \\
0 & . & 1 \\
0 & 0 & 0 \\
\end{vmatrix}
\]

BlockObj = zjblock(ZJordanBlocks) returns a new canonical block object BlockObj representing a structure of Jordan blocks associated with the zero eigenvalue. The vector ZJordanBlocks = [s_n, ..., s_1] is the structure integer partition representing the (unordered) sizes (s_k)-by-(s_k) of the blocks.

BlockObj = zjblock(ZJordanBlocks, Arg) also specifies the notation Arg used for ZJordanBlocks. Valid notation arguments are:

'segre' Sizes are ordered in a non-increasing order.

'weyr' Weyr characteristics.

'sizes' Sizes may be unordered. (default)

BlockObj = zjblock returns an empty Jordan block object.

The class zjblock provides the following methods for extracting information and modifying the canonical block object.

zjblock Methods:
size - Total size of the represented canonical blocks.
numblk - Number of canonical blocks.
copy - Return a copy of the block object.
compare - Compare two block objects.
isempty - True for empty canonical block object.
isingular - True for singular canonical block object.
isregular - True for regular canonical block object.
set - Set the canonical structure information.
get - Get the canonical structure information.
sizes - Canonical block sizes.
segre - Segre characteristics.
weyr - Weyr characteristics.
jcf - Jordan canonical form of the block object.
ccf - Congruence canonical form of the block object.
kcf - Kronecker canonical form of the block object.
bcf - Brunovsky canonical form of the block object.
char - Convert a block object to a string.
block2cell - Convert a block object to a cell array of strings.

zjblock Operators:
==, ~ = (eq, ne) - Check if two block objects are equal.

See also fjblock, ijbloc, cmstruct, scmstruct, mstruct.
6 Canonical Forms

Methods for computing the canonical form.

canonicalforms returns a matrix or matrix pencil in the canonical form of the specified canonical structure object.

Can only be accessed through a canonical structure object (a subclass of mcsstruct).

Reference page in Doc Center
doc canonicalforms

The class methods are listed below.

6.1 BCF

bcf System pencil in the Brunovsky canonical form.

\[[A,B,C,D] = bcf(StructObj)\] takes a state-space system structure object StructObj and returns the corresponding system pencil \( S-sT = [A B; C D] -s[I 0; 0 0] \), in a generalized Brunovsky Canonical Form (bcf).

\[[S,T] = bcf(StructObj)\] returns the system pencil \( S-sT \) above.

\[[A,B] = bcf(StructObj,'ab')\] returns the matrix pair \( (A,B) \) of the sub-system pencil \( [A B]-s[I 0] \) in controllability bcf.

\[[A,C] = bcf(StructObj,'ac')\] returns the matrix pair \( (A,C) \) of the sub-system pencil \( [A; C]-s[I; 0] \) in observability bcf.

\[[A2,B2,C2,D2,P,S,T,R,Q] = bcf(StructObj)\] returns a transformed system pencil and the transformation matrices such that
\[
\begin{bmatrix}
|P & S| & |A B| & -sI & 0 \\
|0 & T| & |C D| & 0 & 0
\end{bmatrix}
\begin{bmatrix}
P' & 0 \\
R & Q'
\end{bmatrix}
=\begin{bmatrix}
A2 & B2 \\
C2 & D2
\end{bmatrix} -sI 0
\]

where \( P, T, \) and \( Q \) are random orthonormal matrices and \( S \) and \( R \) are random matrices of conforming sizes.

\[[A2,B2,P,R,Q] = bcf(StructObj,'ab')\] or
\[[A2,C2,P,S,T] = bcf(StructObj,'ac')\] returns the transformed system pencil and the transformation matrices of the sub-system pencils \( [A B]-s[I 0] \) or \( [A; C]-s[I; 0] \), respectively.

\[[G,H,P,Q,S,T] = bcf(StructObj)\] returns the matrices \( G, H, P, \) and \( Q \) such that \( P(S-sT)Q' = G-sH \) where \( P \) and \( Q \) are random orthonormal
matrices.

\[ ... = \text{bcf}(\ldots, '\text{segre}') \] sorts the blocks in each canonical block object in non-increasing block size order. By default are the blocks presented in the order they were created.

Valid canonical structure objects are: ssstruct

See also ssstruct, kcf.

### 6.2 KCF

The \textit{kcf} Matrix pencil in the Kronecker canonical form.

\[ [G,H] = \text{kcf}(\text{StructObj}) \]

G and H are matrix pencils in Kronecker Canonical Form (\textit{kcf}) corresponding to the structure. (See below for the corresponding syntax for matrices.)

\[ [A,B,P,Q] = \text{kcf}(\text{StructObj}) \]

returns the matrices A, B, P, and Q such that \( P(G-sH)Q' = A-sB \) where \( P \) and \( Q \) are random orthonormal matrices.

\[ [A,B,P,Q,G,H] = \text{kcf}(\text{StructObj}) \]

also returns G and H in \textit{kcf}.

\[ ... = \text{kcf}(\text{StructObj}, '\text{segre}') \]

sorts the blocks in each canonical block object in non-increasing block size order. By default are the blocks presented in the order they were created.

Valid canonical structure objects are: pstruct, spstruct, sspstruct, ssstruct.

Examples:

\begin{verbatim}
>> pstr = pstruct([], [], {[2] [1]}, [1 3], [2];
>> [G,H] = pstr.kcf;
returns a matrix pencil of the following form:
\[
\begin{bmatrix}
    J2(1)-sI & 0 & 0 \\
    0 & J1(3)-sI & 0 \\
    0 & 0 & N2
\end{bmatrix}
\]
\end{verbatim}

See also pstruct, spstruct, sspstruct, ssstruct.

### 6.3 CCF

The \textit{ccf} Matrix in the congruent canonical form.

\[ C = \text{ccf}(\text{StructObj}) \]

\( C \) is a matrix in Congruent Canonical Form (\textit{ccf}) corresponding to the canonical structure information.
[X,Z] = \texttt{ccf(StructObj)} returns the two matrices Z and X such that Z*C*Z' = X, where Z is a random orthonormal matrix.

[X,Z,C] = \texttt{ccf(StructObj)} also returns C in \texttt{ccf}.

... = \texttt{ccf(StructObj,'segr')} sorts the blocks in each canonical block object in non-increasing block size order. By default are the blocks presented in the order they were created.

Valid canonical structure objects are: cmstruct, scmstruct.

Example:

\begin{verbatim}
>> C = ccf(cmstruct([2],[2],4));
\end{verbatim}
returns a 2-by-2 Gamma block and one W block of size 4-by-4 with the associated eigenvalue 4:

\begin{verbatim}
| 0 -1 0 0 0 |
| 1 1 0 0 0 |
| 0 0 0 0 1 |
C = | 0 0 0 0 1 |
| 0 0 4 1 0 |
| 0 0 0 4 0 |
\end{verbatim}

See also \texttt{cmstruct, scmstruct}.

6.4 JCF

\texttt{jcf} Matrix in the Jordan canonical form.

\begin{verbatim}
J = jcf(StructObj) takes a matrix structure object StructObj and returns a matrix J in Jordan Canonical Form (jcf) corresponding to the canonical structure information.
\end{verbatim}

[X,Z] = \texttt{jcf(StructObj)} returns the two matrices Z and X such that ZJZ' = X, where Z is a random orthonormal matrix.

[X,Z,J] = \texttt{jcf(StructObj)} also returns J in \texttt{jcf}.

... = \texttt{jcf(StructObj,'segr')} sorts the blocks in each canonical block object in non-increasing block size order. By default are the blocks presented in the order they were created.

Valid canonical structure objects are: mstruct.

Example:

\begin{verbatim}
>> J = jcf(mstruct([3 1],4)); returns a matrix J with one Jordan block of size 3-by-3 and one of size 1-by-1, both with eigenvalue 4.
\end{verbatim}

See also \texttt{mstruct, kcf}.
6.5 JNF

\texttt{jnf} Matrix in the Jordan normal form.

\[ J = \text{jnf}(\text{StructObj}) \text{ returns a matrix } J \text{ in Jordan Normal/Canonical Form of a matrix structure object } \text{StructObj. This methods calls } \text{jcf}. \]

See also \texttt{jcf}. 
7 Guptri functions

7.1 PCLUSTER

Compute and cluster generalized eigenvalues of a matrix pencil.

\[
eigs = \text{pcluster}(G,H)\]

computes and clusters the generalized eigenvalues of the square matrix pencil \(G-sH\). The average of the generalized eigenvalues classified to belong to the same cluster is considered as a (numerical) multiple eigenvalue. \(\text{Eigs}\) is a two-column matrix, where the first column is the generalized eigenvalues of \(G-sH\) and the second column is the corresponding cluster sizes. The number of rows in \(\text{Eigs}\) is consequently the same as the number of distinct (possible multiple) eigenvalues of \(G-sH\). By default the cluster method returned by \text{mcsevclmth} is used. Optional parameters to the default cluster method can be appended to the argument list, see below for available parameters.

\[\text{[Eig,S]} = \text{pcluster}(G,H)\] instead only computes the cluster size of the largest eigenvalue and returns the eigenvalue and cluster size in two separate variables.

\[
\text{Eigs} = \text{pcluster}(\text{Method},G,H, \ldots)\]

or

\[\text{[Eig,S]} = \text{pcluster}(\text{Method},G,H, \ldots)\] also specifies the method used to cluster the eigenvalues with optional extra parameters. For available methods see below.

\[
\ldots = \text{pcluster}('\text{norm}',G,H [,\text{Tol}])\]

uses an "over-simple" method where the eigenvalues that are within the norm tolerance \(\text{Tol}\) are classified to belong to the same cluster. By default \(\text{Tol} = \sqrt{\epsilon} \times \max(\text{norm}(G,'\text{fro'}),\text{norm}(H,'\text{fro'}))\).

For details see \text{private/cenorm}.

\[
\ldots = \text{pcluster}(\text{Method},G,H [,\text{DeltaMax} [,\text{Pmin}]])\]

uses a hierarchical clustering algorithm where the clustering is determined by a distance function \(D\) that satisfy \(D \leq \text{DeltaMax}\). \(\text{Pmin}\) is minimum number of clusters generated. Available distance functions \(D(X,Y)\) (specified with \text{Method}) are, where \(X\) and \(Y\) are two different clusters with \(r\) and \(s\) eigenvalues, respectively:

- '\text{min}' - \(D(X,Y) = \min(\text{abs}(X(i) - Y(j)))\), for all \(i,j\)
- '\text{max}' - \(D(X,Y) = \max(\text{abs}(X(i) - Y(j)))\), for all \(i,j\)
- '\text{avg}' - \(D(X,Y) = 1/(r*s) \times (\sum^r \text{abs}(X(i) - Y(j)))\)
- '\text{mean}' - \(D(X,Y) = \text{abs}(\sum(X)/r - \sum(Y)/s)\)
- '\text{var}' - \(D(X,Y) = \text{var}(X \cup Y), \text{(variance of } X \text{ union } Y)\)

Default values are \(\text{DeltaMax} = \sqrt{\epsilon} \times \max(\text{norm}(G,'\text{fro'}),\text{norm}(H,'\text{fro'}))\) and \(\text{Pmin} = 1\). If the matrix pencil is known to have \(k\) multiple eigenvalues, appropriate parameters are \(\text{DeltaMax} = \infty\) and \(\text{Pmin} = k\).

For details see \text{private/cehierarchy}.

\[
\ldots = \text{pcluster}(‘\text{gersh’},G,H [,\text{SizeE,SizeF} [,\text{Goal}]]))\]

or

\[
\ldots = \text{pcluster}(‘\text{gersh’},G,H [,\text{SizeEF} [,\text{Goal}]]))\]

uses a method based on Gershgorin circles to cluster the eigenvalues closest to \(\text{Goal}\) or the
largest eigenvalue (default). The function starts with isolating the cluster closest to the point Goal in the complex plane. If the cluster containing the largest eigenvalue is desired to be first then Goal should be set to Inf or omitted. SizeE and SizeF should be estimates of the $1$-norm of the errors known to affect G and H, respectively. The error estimates can also be specified as vector $\text{SizeEF} = [\text{SizeE} \ \text{SizeF}]$. If not specified or empty, $\text{SizeE} = \sqrt{\text{eps}} \times \|G\|_{\text{fro}}$ and $\text{SizeF} = \sqrt{\text{eps}} \times \|H\|_{\text{fro}}$.

For details see private/pcegershgorin.

$[\text{Eigs},E,\text{EvCluster}] = \text{pcluster}(\ldots)$ also returns all computed eigenvalues $E$ of $G-sH$ and the clustering of the eigenvalues in EvCluster. The integer value $k$ in EvCluster($i$) represents the cluster the eigenvalue $E(i)$ belongs to, where Ev($k$) is the new eigenvalue for the cluster. The eigenvalues in $E$ are sorted in non-increasing order.

See also private/cenorm, private/cehierarchy, private/pcegershgorin, mcluster, mcsevclmth.

### 7.2 PGGALLERY

Test matrix pencils for staircase computation and bounds.

$[G,H] = \text{pggallery}(\text{PencilName})$ returns a matrix pencil $G-sH$ for testing computation routines of the staircase form (canonical structure) and bounds. The string PencilName specifies the matrix pencil (see below).

$[S,T,Eg,Eh] = \text{pggallery}(\text{PencilName},\text{Pert})$ also adds a random perturbation of size Pert to the pencil, where $S = G + Eg$ and $T = H + Eh$. If Pert is zero or empty, no perturbation is added.

$[G,H,PStr] = \text{pggallery}(\ldots)$ or $[S,T,Eg,Eh,PStr] = \text{pggallery}(\ldots)$ returns the correct canonical form for the matrix pencil $G-sH$ as a pstruct object in PStr, where in the descriptions below the block notation is used:

- $R_n$ is an $n$-by-$n$ Jordan block,
- $L_n$ is a $(n+1)$-by-$n$ left singular block,
- $J_n(\mu)$ is an $n$-by-$n$ Jordan block with eigenvalue $\mu$, and
- $N_n$ is an $n$-by-$n$ Jordan block with an infinite eigenvalue.

A number before the block denotes the number of blocks of the same size and type.

Available test matrix pencils are (specified with PencilName and with optional parameters added as input arguments after Pert):

- 'boley1' - A 7x8 matrix pencil already in staircase form that corresponds to state-space system which is controllable but close to an uncontrollable system, for $\text{eps}=1$ at a distance $6e-4$ [Boley90, Example 2, p. 639] (see also [EdelmanMa00]). The default value for the optional parameter $\text{eps}$ is 1. The canonical form is $R7$ and for
eps=0, R6 + J1(a).

\[
\begin{bmatrix}
0 & 1 & 0 & \ldots \\
0 & 1 & 0 & \\
0 & 0 & 1 & 0
\end{bmatrix}
\quad
\begin{bmatrix}
1 & -1 & \ldots & -1 & 7 \\
0 & 1 & -1 & \ldots & -1 & 6 \\
0 & 1 & 0 & \\
1 & 0 & \\
\end{bmatrix}
\]

G = [ \ldots ] and H(eps) = [ \ldots : : ]

\[
\begin{bmatrix}
0 & 1 & 0 \\
0 & 1 & 0 \\
1 & 0 & \\
\end{bmatrix}
\quad
\begin{bmatrix}
1 & -1 & 2 \\
\end{bmatrix}
\]

'boley2' - A 3x4 matrix pencil from [Boley90, Example 3, p.639] with ill-conditioned eigenvalues. The canonical form is R3.

'degreg' - A degenerated regular 8x8 matrix pencil constructed from the Kronecker canonical form

\[ J2(6) + 2J1(6) + J3(2) + J1(2). \]

'dk_c1', 'dk_c2', 'dk_c3' - Examples from [DemmelKagstrom88, p. 142]. The examples have successively more ill-conditioned eigenvalues. The canonical form is R2 + J1(1) + J1(2) for all three cases.

'em' - Example from [EdelmanMa00, p. 1018]. The default value for the optional parameter d is 1.5e-8. The canonical form is R1 + J2(0). This pencil is sensitive even for for small perturbations (1e-14).

\[
\begin{bmatrix}
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\end{bmatrix}
\quad
\begin{bmatrix}
d & 0 & 0 & 0 \\
0 & d & 0 & 0 \\
0 & 0 & 1 & 0 \\
\end{bmatrix}
\]

'lmix' - An example of an 11x9 matrix pencil with the canonical form L0 + L8 + N1 originally coming from preprocessing of \textit{lmix:s} [Helmerson09].

'ones' - An example provided by [Mason04]:

G = ones(55,35) and H = pi*ones(55,35). The exact canonical form is 34R0 + 54L0 + J1(e), where e = (1/1925)/(pi/1925).

'rand' - A 5x7 random matrix pencil with the canonical form 5L1 + 15L2 (the generic pencil). With the two optional arguments m and n the size of the matrix pencil can be specified. If only one argument is provided the matrix pencil will be square and regular, see also 'square_rand'.

'rand_large' - A 650x503 random matrix pencil with the canonical form 85L3 + 62L4.

'rand_medium' - A 55x35 random matrix pencil with the canonical form 5L1 + 15L2.

'rand_square' - A 5x5 square regular random matrix pencil. With the optional n the size of the matrix pencil can be specified.

'reference' - A 14x12 matrix pencil with all types of canonical blocks.
constructed from the Kronecker canonical form
R1 + R0 + L2 + 3L0 + 2J1(6) + J1(2+5i) + J1(0) + N2 + N1

'test1' - A 3x4 matrix pencil which is constructed from the exact
Kronecker canonical form
R0 + R1 + L0 + J1(1e-4i)
by an equivalence transformation with two perturbed orthogonal
matrices [Karlsson15]. The constructed matrices G and H are
ill-conditioned, O(cond(G)) = O(cond(H)) = 1e16. pguptri compute the
correct canonical structure if the order of the regular blocks are
'ifz', 'izf', or 'fiz' (the two matrices G and H are swaped in the
computations), but fails for the other orders.

'test2' - A larger variant of 'test2' where the size of the matrix
pencil is 33x39 [Karlsson15]. The correct canonical form is
2R0 + 2R2 + 2R5 + 2J1(0) + 5J1(0.05) + 4J2(0) + J4(0).
The constructed matrix G is ill-conditioned, O(cond(G)) = 1e16.
pguptri compute the correct canonical structure if the order of the
regular blocks are 'ifz', 'izf', or 'fiz' (the two matrices G and H
are swaped in the computations), but fails for the other orders.

References

structure of pencils with simple eigenvalue estimates. siam J. Matrix


[DemmelKagstrom88] J. Demmel and B. Kagstrom. Accurate solutions of

[Helmerson09] A. Helmersson, Dept Electrical Engineering, Linkoping
University, Sweden. Private communication, 2009.

[Karlsson15] L. Karlsson, Dept Computing Science, Umea University,

communication, 2004.

See also mggallery, pguptri.

7.3 PGUPTRI

Compute the canonical structure information of a matrix pencil.
PStr = pguptri(G,H) returns a pstruct object PStr representing the
Kronecker canonical structure of G-sH, i.e., the sizes of existing
singular blocks and regular blocks with associated eigenvalues. The
Kronecker canonical structure information is computed using the Guptri
staircase algorithm.

PStr = pguptri(G,H,Tol,Gap) determines the structure with respect to
the deflation tolerance, Tol, and the required gap, Gap, in the
singular values of the orthogonal deflations. Tol and Gap are used to
make rank decisions by searching for adjacent singular values whose
ratio exceeds Gap and the smaller one is less than Tol. If Tol is a
vector [TolG TolH], different tolerances Tol are used for G and H. Tol
is the absolute uncertainty in the data and should be at least about
eps and nominally between 1e-8 and 1e-12. Gap should be at least 1 and
nominally 1000. If Tol and Gap are not provided, default tolerances are
used (see mcstolerance). By setting Tol or Gap to empty matrix the
default value is used for that tolerance.

pguptri(G,H,Tol,Gap,...) or
pguptri(G,H,...) can take the additional arguments in any order (if
specified, set Tol and/or Gap to empty matrix to use their default values):
'ZFI' | 'ZIF' | 'IFZ' | 'IZF' | 'FZI' | 'FIZ'
The string describes in which order the regular subpencils should
appear in the Guptri form. The letters F, Z, and I stand for
finite, zero, and infinite eigenvalues, respectively. The default
order is 'ZFI'.
'norm' | 'gersh' | 'min' | 'max' | 'avg' | 'mean' | 'var'
Specifies the method used to cluster the eigenvalues with default
tolerance. Optional tolerance parameter to the cluster method can
be specified in a subsequent argument CTol. For example, the following
call set CTol = eps for 'norm'
> pstr = pguptri(G,H,'norm',eps)
See pcluster for a full description of each method. By default, the
cluster method returned by mcsevclmth is used.
'absolute' | 'relative' | 'relative_strict'
If 'absolute', the tolerance Tol is used to determine the
nullspace of G and H.
If 'relative', relative tolerances are used for G and H:
TolG = norm(G,'fro')*Tol(1) and
TolH = norm(H,'fro')*Tol(2),
where Tol normally should be the machine precision but this will
in some cases be to optimistic.
If 'relative_strict', the value of Tol is ignored and (normally)
strict relative tolerances are used for G and H:
TolG = max(size(G))*eps(norm(G,'fro')) and
TolH = max(size(H))*eps(norm(H,'fro')).
Default is 'absolute'.
'zeros'|'nozeros'
If 'zeros', the singular values interpreted as zeros (with respect
to Tol and Gap) in the deflation process are set to zero.
Otherwise ('nozeros'), small singular values are kept and used in the computations. The default is 'zeros'.

'forceimpose'
If 'forceimpose', also singular values larger than Tol*Gap will be forced to zero in the separation steps. Default is that if a "large" singular value is going to be set to zero the current separation will stop and continue with the next. Instead a warning is issued and the returned S and T (see below) are not in exact Guptri staircase form.

'suppresswarnings'
Suppress all warnings. This option should be used with care!

`pguptri` can also return the following data:

```plaintext
[S,T] = pguptri(...)
[S,T,PStr] = pguptri(...)
[S,T,P,Q] = pguptri(...)
[S,T,P,Q,PStr] = pguptri(...)
[S,T,P,Q,dG,dH] = pguptri(...)
[S,T,P,Q,dG,dH,PStr] = pguptri(...)
```

where P and Q are unitary (orthonormal) and S and T are in generalized upper triangular form (Guptri staircase form), such that $P(S-sT)Q = (G+dG)-(H+dH)$. The Frobenius norm of $dG$ and $dH$ is an upper bound on the distance from $G-sH$ to a matrix pencil with the computed Kronecker structure as the exact one. Given the argument 'zeros' (default), \( \text{norm}([dG,dH]) \) is of size \( \text{norm}([G,H])*Tol \). Otherwise, given the argument 'nozeros', \( S-sT \) is an "exact" orthogonal equivalence transformation of \( G-sH \), and \( \text{norm}([dG,dH]) \) is of size \( \text{norm}([G,H])*\text{eps} \), where $dG$ and $dH$ are formed by $P*S*Q'-G$ and $P*T*Q'-H$, respectively.

Algorithm:
Based on the algorithm described in

See also `mguptri`, `pstruct`, `pcluster`, `mcstolerance`, `mcsevclmth`. 
8 Tangent spaces and codimensions

8.1 CMCODIM

Codimension of a matrix orbit under congruence.

\texttt{cmcodim}(H) computes the codimension of the tangent space of the congruence orbit of a matrix H. Default tolerance parameter of \texttt{rank} is used.

\texttt{cmcodim}(H,Tol) uses the specified tolerance Tol.

\texttt{cmcodim}(CMstr) determines the codimension of the orbit of the congruence matrix structure object CMstr. The codimension is determined with respect to the represented canonical structure not the tangent space. For further information, see \texttt{cmstruct/codim}.

See also \texttt{cmstruct/codim}, \texttt{mcodim}.

8.2 CMTANSPACE

Tangent space of the congruence orbit of a matrix.

\texttt{T = cmtanspace(A)} returns the matrix representation

\[ T = \texttt{kron(A.',In)} + \texttt{kron(In, A)*P} \]

of the tangent space to the congruence orbit(A) at A, where A is an n-by-n matrix, I_n is the n-by-n identity matrix, and P is the (n^2)-by-(n^2) permutation matrix that can "transpose" n-by-n matrices, i.e., vec(X')=P*vec(X) for any n-by-n matrix X. The tangent space is the range of the (n^2)-by-(n^2) matrix T.

The tangent space consists of the matrices of the form

\[ T_A = X'*A + A*X, \]

where X is an n-by-n matrix. Equivalently, the tangent vectors T_A can be represented as

\[ \text{vec}(T_A) = (\text{kron(A.',In)}+\text{kron(In, A)*P}) \text{vec}(X). \]

See also \texttt{cmcodim}, \texttt{mtanspace}, \texttt{scmtanspace}.

8.3 MCODIM

Codimension of a matrix space.

\texttt{mcodim}(A) computes the codimension of the tangent space of the similarity orbit of a square matrix A. Default tolerance parameter for \texttt{rank} is used.

\texttt{mcodim}(A,Tol) uses the specified tolerance Tol.
mcodim(Mstr) determines the codimension of the orbit of a matrix structure object Mstr. The codimension is determined with respect to the represented canonical structure not the tangent space.

See also mtanspace, mstruct/codim, pcodim.

8.4 MTANSPACE

Tangent space of the orbit of a matrix.

\[ T = \text{mtanspace}(A) \]

returns the matrix representation

\[ T = \text{kron}(A', I) - \text{kron}(I_n, A) \]

of the tangent space to the similarity orbit \( (A) \) at \( A \), where \( A \) is an \( n \times n \) matrix and \( I_n \) is the \( n \times n \) identity matrix. The tangent space is the range of the \( (n^2) \times (n^2) \) matrix \( T \).

The tangent space consists of the matrices of the form

\[ T_A = X \cdot A - A \cdot X, \]

where \( X \) is an \( n \times n \) matrix. Equivalently, the tangent vectors \( T_A \) can be represented as

\[ \text{vec}(T_A) = T \cdot \text{vec}(X). \]

See also mcodim, ptanspace.

8.5 PCODIM

Codimension of a matrix pencil orbit.

\[ \text{pcodim}(G,H) \]

computes the codimension of the tangent space of the strict equivalence orbit of a matrix pencil \( G-sH \). Default tolerance parameter of \( \text{rank} \) is used.

\[ \text{pcodim}(G,H,Tol) \] uses the specified tolerance \( Tol \).

\[ \text{pcodim}(Pstr) \]

determines the codimension of the orbit of the matrix pencil structure object \( Pstr \). The codimension is determined with respect to the represented canonical structure not the tangent space. For further information, see pstruct/codim.

See also ptanscpce, pstruct/codim, mcodim.

8.6 PTANSPACE

Tangent space of the orbit of a matrix pencil.

\[ T = \text{ptanspace}(G,H) \]

returns the matrix representation
of the tangent space to the strict equivalence orbit(G-sH) at G-sH,
where G-sH is a m-by-n matrix pencil and I_x is the x-by-x identity
matrix. The tangent space is the range of the (2mn)-by-(m^2+n^2) matrix
T.

The tangent space consists of the matrix pencils of the form
T_G - sT_H = X(G-sH) - (G-sH)Y,
where X is an m-by-m matrix and Y is an n-by-n matrix. Equivalently,
the tangent vectors T_G - sT_H can be represented as

<table>
<thead>
<tr>
<th>vec(T_G)</th>
<th>kron(G.',I_m)</th>
<th>kron(I_n,G)</th>
</tr>
</thead>
<tbody>
<tr>
<td>vec(T_H)</td>
<td>kron(H.',I_m)</td>
<td>kron(I_n,H)</td>
</tr>
</tbody>
</table>

See also pcodim, mtanspace.

8.7 S2CODIM

Codimension of a matrix pair.

s2codim([Type],A,B) or
s2codim([Type],A,C) computes the codimension of the tangent space of
the feedback equivalence orbit of a system pencil [A B]-s[I 0] (or
[A;C]-s[I;0]). Default tolerance parameter of rank is used.

The optional argument Type is a string that for a controllability pair
(A,B) should be 'ab' and for an observability pair (A,C) should be
'ac'. If the argument is absent, the sizes of A and B (or C) will
determine the type. If B is square, 'ab' is assumed.

s2codim([Type],A,C,Tol) or
s2codim([Type],A,B,Tol) uses the specified tolerance Tol.

s2codim(SSstr) determines the codimension of the orbit of state-space
structure object SSstr. The codimension is determined with respect to
the represented canonical structure not the tangent space. For further
information, see ssstruct/codim.

See also s2tanspace, ssstruct/codim, pcodim.

8.8 S2TANSPACE

Tangent space of the orbit of a matrix pair.

s2tanspace([Type],A,B) returns the matrix representation
of the tangent space to the feedback equivalence orbit([A B]-s[I 0]) at (A,B).

s2tanspace([Type],A,C) returns the matrix representation

\[
T(A,C) = \begin{bmatrix}
\kron(A.',I_n)-\kron(I_n,A) & \kron(C.',I_n) & 0 \\
-k\kron(I_n,C) & 0 & \kron(C.',I_p)
\end{bmatrix}
\]

of the tangent space to the feedback equivalence orbit([A;C]-s[I;0]) at (A,C).

The optional parameter, Type, is a string argument that for a controllability pair (A,B) should be 'ab' and for an observability pair (A,C) should be 'ac'. If the argument is absent, the sizes of A and B (or C) will determine the type. If B is square, 'ab' is assumed.

The tangent space for (A,B) and (A,C) consist of the matrices of the forms

\[
\begin{bmatrix} T_A & T_B \end{bmatrix} = X \begin{bmatrix} A & B \end{bmatrix} + \begin{bmatrix} A & B \end{bmatrix} -X 0; V W
\]

and

\[
\begin{bmatrix} T_A \\ T_B \\ T_C \\ T_C \end{bmatrix} = \begin{bmatrix} X Y \\ A \\ Z C \\ C \end{bmatrix},
\]

respectively, where X, Y, Z, V, and W are matrices of conforming sizes. Equivalently, the tangent vectors \( T_A \) and \( T_B \) at (A,B) can be represented as

\[
\begin{align*}
|\text{vec}(T_A)| &= |\kron(A.',I_n)-\kron(I_n,A)|
|\text{vec}(T_B)| &= |\kron(B.',I_n)|
|\text{vec}(T_C)| &= |\kron(I_n,B)|
|\text{vec}(T_C)| &= |\kron(I_n,B)|
|\text{vec}(T_B)| &= |\kron(I_n,B)|
|\text{vec}(T_B)| &= |\kron(I_n,B)|
\end{align*}
\]

and the tangent vectors \( T_A \) and \( T_C \) at (A,C) as

\[
\begin{align*}
|\text{vec}(T_A)| &= |\kron(A.',I_n)-\kron(I_n,A)|
|\text{vec}(T_C)| &= |\kron(C.',I_n)|
|\text{vec}(T_C)| &= |\kron(C.',I_n)|
|\text{vec}(T_C)| &= |\kron(C.',I_n)|
\end{align*}
\]

See also \texttt{s2codim}, \texttt{ssstruct/codim}.
8.9 SCMCODIM

Codimension of a matrix orbit under *congruence.

\texttt{scmcodim}(H) computes the codimension of the tangent space of the *congruence orbit of a matrix H over the field of real numbers. Default tolerance parameter of \texttt{rank} is used.

\texttt{scmcodim}(H,Tol) uses the specified tolerance Tol.

\texttt{scmcodim} (SCMstr) determines the codimension of the orbit of the *congruence matrix structure object SCMstr. The codimension is determined with respect to the represented canonical structure not the tangent space. For further information, see \texttt{scmstruct/codim}.

See also \texttt{scmstruct/codim}, \texttt{mcodim}.

8.10 SCMTANSPACE

Tangent space of the *congruence orbit of a matrix.

\( T = \texttt{scmtanspace}(A) \) returns the matrix representation

\[
\begin{bmatrix}
| & | \\
| \text{kron}(\text{re}(A)', \text{In})+\text{kron}(\text{In}, \text{re}(A))\text{*P} & -\text{kron}(\text{im}(A)', \text{In})+\text{kron}(\text{In}, \text{im}(A))\text{*P} \\
| & |
\end{bmatrix}
\]

of the tangent space to the *congruence orbit(A) at A over the field of real numbers, where A is an n-by-n matrix, re(A) and im(A) are the real and imaginary parts of A, \( I_n \) is the n-by-n identity matrix, and P is the \((n^2)\)-by-\((n^2)\) permutation matrix that can "transpose" n-by-n matrices, i.e., vec(X')=P*vec(X) for any n-by-n matrix X. The tangent space is the range of the \((2*n^2)\)-by-\((2*n^2)\) matrix T.

The tangent space consists of the matrix of the form

\( T_A = X.'*A + A*X, \)

where X is an n-by-n matrix. Equivalently, the tangent vectors \( T_A \) can be represented as

\[
\begin{bmatrix}
| & | \\
| \text{vec}(\text{re}(T_A)) & \text{vec}(\text{re}(X)) + \\
| & | \\
| \text{vec}(\text{im}(T_A)) & \text{vec}(\text{im}(X)) \\
| & |
\end{bmatrix}
\]

See also \texttt{scmcodim}, \texttt{cmtanspace}.
8.11 SPCODIM

Codimension of a symmetric matrix pencil orbit.

\[ \text{spcodim}(G,H) \] computes the codimension of the tangent space to the congruence orbit of a symmetric matrix pencil \( G-sH \). Default tolerance parameter of \texttt{rank} is used.

\[ \text{spcodim}(G,H,Th) \] uses the specified tolerance \( Th \).

\[ \text{spcodim}(SPstr) \] determines the codimension of the congruence orbit of the symmetric matrix pencil structure object \( SPstr \). The codimension is determined with respect to the represented canonical structure not the tangent space. For further information, see \texttt{spstruct/codim}.

See also \texttt{pstruct/codim}, \texttt{sspcodim}.

8.12 SPTANSPACE

Tangent space to the congruence orbit of a symmetric matrix pencil.

\[ T = \text{sptanspace}(G,H) \] returns the matrix representation

\[
T = \begin{vmatrix}
| \text{kron}(G.',\text{In}) + \text{kron}(\text{In},G)*P | \\
| \text{kron}(H.',\text{In}) + \text{kron}(\text{In},H)*P |
\end{vmatrix}
\]

of the tangent space to the congruence orbit \( (G-sH) \) at \( G-sH \), where \( G-sH \) is a \( n \)-by-\( n \) symmetric matrix pencil and \( I_n \) is the \( n \)-by-\( n \) identity matrix, and \( P \) is the \( (n^2) \)-by-\( (n^2) \) permutation matrix that can "transpose" \( n \)-by-\( n \) matrices, i.e., \( \text{vec}(X') = P*\text{vec}(X) \) for any \( n \)-by-\( n \) matrix \( X \). The tangent space is the range of the \( (2n^2) \)-by-\( (n^2) \) matrix \( T \).

The tangent space consists of the matrix pencils of the form

\[ T_G - sT_H = X'(G-sH) + (G-sH)X, \]

where \( X \) is an \( n \)-by-\( n \) matrix. Equivalently, the tangent vectors

\[ T_G - sT_H \]

can be represented as

\[
| \text{vec}(T_G) | \begin{vmatrix}
| \text{kron}(G.',I_n) | \\
| \text{kron}(I_n,G) |
\end{vmatrix}
\]

\[
= \begin{vmatrix}
| \text{vec}(X) - P*\text{vec}(X) |
\end{vmatrix}
\]

See also \texttt{spcodim}, \texttt{ptanspace}, \texttt{ssptanspace}.

8.13 SSSPCODIM

Codimension of a skew-symmetric matrix pencil orbit.

\[ \text{sspcodim}(G,H) \] computes the codimension of the tangent space of the congruence orbit of a skew-symmetric matrix pencil \( G-sH \). Default
tolerance parameter of \texttt{rank} is used.

\texttt{sspcodim}(G,H,Tol) uses the specified tolerance Tol.

\texttt{sspcodim}(SSPstr) determines the codimension of the congruence orbit of
the skew-symmetric matrix pencil structure object SSPstr. The
codimension is determined with respect to the represented canonical
structure not the tangent space.

See also \texttt{pstruct/codim}, \texttt{mcodim}.

\section*{8.14 \texttt{SSPTANSPACE}}

Tangent space to the congruence orbit of a skew-symmetric matrix pencil.

\texttt{T = ssptanspace}(G,H) returns the matrix representation

\[
T = \begin{bmatrix}
\kron(G.',\text{In}) + \kron(\text{In},G)*P \\
\kron(H.',\text{In}) + \kron(\text{In},H)*P
\end{bmatrix}
\]

of the tangent space to the congruence orbit\((G-sH)\) at \(G-sH\), where \(G-sH\)
is a \(n\)-by-\(n\) skew-symmetric matrix pencil and \(\text{I}_n\) is the \(n\)-by-\(n\) identity
matrix, and \(P\) is the \((n^2)\)-by-\((n^2)\) permutation matrix that can
"transpose" \(n\)-by-\(n\) matrices, i.e., \(\text{vec}(X')=P*\text{vec}(X)\) for any \(n\)-by-\(n\)
matrix \(X\). The tangent space is the range of the \((2n^2)\)-by-\((n^2)\) matrix
\(T\).

The tangent space consists of the matrix pencils of the form

\[
T_G - sT_H = X'(G-sH) + (G-sH)X,
\]

where \(X\) is an \(n\)-by-\(n\) matrix. Equivalently, the tangent vectors
\(T_G - sT_H\) can be represented as

\[
\begin{bmatrix}
\text{vec}(T_G) & \kron(G.',\text{I}_n) & \kron(\text{I}_n,G) \\
\kron(H.',\text{I}_n) & \text{vec}(X) & P*\text{vec}(X)
\end{bmatrix}
\]

See also \texttt{sspcodim}, \texttt{ptanspace}, \texttt{sptanspace}.
9 StartiGraph public interface functions

These functions are part of the StratiGraph Matlab plugin.

9.1 SGCLOSE

Close a StratiGraph window.

\[
\text{sgclose}(	ext{Label}) \quad \text{close the StratiGraph window with label Label.}
\]

The label can either be a positive integer or a string.

\[
\text{sgclose}('all') \quad \text{close all opened StratiGraph windows. Even those who are not opened from Matlab.}
\]

See also \text{sgopen}.

9.2 SGGET

Get the active structure in StratiGraph.

\[
\text{StructObj} = \text{sgget} \quad \text{returns a Matlab structure object of the active structure in StratiGraph. If no structure or an edge is active, StructObj is empty.}
\]

\[
\text{StructObj} = \text{sgget}(	ext{Label}) \quad \text{specifies the label of the StratiGraph window to get the data from.}
\]

See also \text{sgset}, \text{sgopen}.

9.3 SGGETCONSTRAINTS

Return a list of available setup constraints.

\[
\text{sggetconstraints} \quad \text{list all setups and available constraints in StratiGraph to the command window.}
\]

\[
\text{Constr} = \text{sggetconstraints}(	ext{Struct}) \quad \text{returns for a canonical structure object the list Constr with all available constraints for the corresponding setup in StratiGraph, where Struct can be a canonical structure object or the name of the canonical structure class as a string. The returned list is a cell-array of strings.}
\]

See also \text{sgset}.

9.4 SGLABELS

Return the labels of the opened StratiGraph windows.
Labels = sglabels returns a cell array with the labels of the opened
StratiGraph windows.

See also sgopen, sgclose.

9.5 SGOPEN

Open a StratiGraph window.

sgopen opens a new StratiGraph window.
Label = sgopen also returns the label of the StratiGraph window.

sgopen(Label) opens a new StratiGraph window with the label Label that
can be used in communication. The label must either be a positive
integer or a string. If the window Label already exists then it is
activated and send to front.

sgopen(Label,Args) start StratiGraph with the command line arguments
Args. Can only be used when no StratiGraph window is open.
For example:
sgopen('','-v') - returns the StratiGraph version.
sgopen('','-h') - display the help text.

See also sgset, sgget, sgclose.

9.6 SGSET

Send a canonical structure to StratiGraph.

sgset(StructObj,StrataType) set the start node in StratiGraph to
the canonical structure represented by the object StructObj. Any
existing graph in StratiGraph is deleted. The parameter
StrataType specifies the strata and must be either 'orbit' or
'bundle'. If StrataType is omitted, 'orbit' is assumed. By
default, if there exists a graph in StratiGraph the user will be
asked if it should be overwritten.

sgset(StructObj,StrataType,Constraint) also specify the setup
constraint. By default no constraint is used for the setup. To
list all available constraints for a canonical structure object
call the method getsgconstraints on the object or the function
sggetconstraints. By setting Constraint to 'default' or empty
string, no constraint is used.

sgset('matrix', A, StrataType) initializes StratiGraph with the
structure of the matrix A.

sgset('abpair', A, B, StrataType) initializes StratiGraph with
the structure of the controllability pair (A,B).
sgset('acpair', A, C, StrataType) initializes StratiGraph with the structure of the observability pair (A,C).

sgset('pencil', G, H, StrataType) initializes StratiGraph with the structure of the matrix pencil G-sH.

sgset(...,ForceOverwrite) let the user suppress the prompt if any existing graph in StratiGraph should be overwritten by setting the boolean ForceOverwrite to true.

sgset(...,ForceOverwrite,Label) specifies the label of the StratiGraph window to be initialize. If no label is provided the initialization is send to the first window.

See also sgget, sgopen, sggetconstraints, sgsettolerance.


10 Software License

Matrix Canonical Structure (MCS) Toolbox for Matlab
Copyright (c) 2013-2019, The MCS Team*
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer listed in this license in the documentation and/or other materials provided with the distribution.

- Neither the name of the copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

The copyright holders provide no reassurances that the source code provided does not infringe any patent, copyright, or any other intellectual property rights of third parties. The copyright holders disclaim any liability to any recipient for claims brought against recipient by any third party for infringement of that parties intellectual property rights.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

*) The MCS Team@Umeå University consists of

Programming: Stefan Johansson, Pedher Johansson, and Andrii Dmytryshyn.

Scientific contribution: Bo Kågström, Erik Elmroth, Pedher Johansson, Stefan Johansson, and Andrii Dmytryshyn.