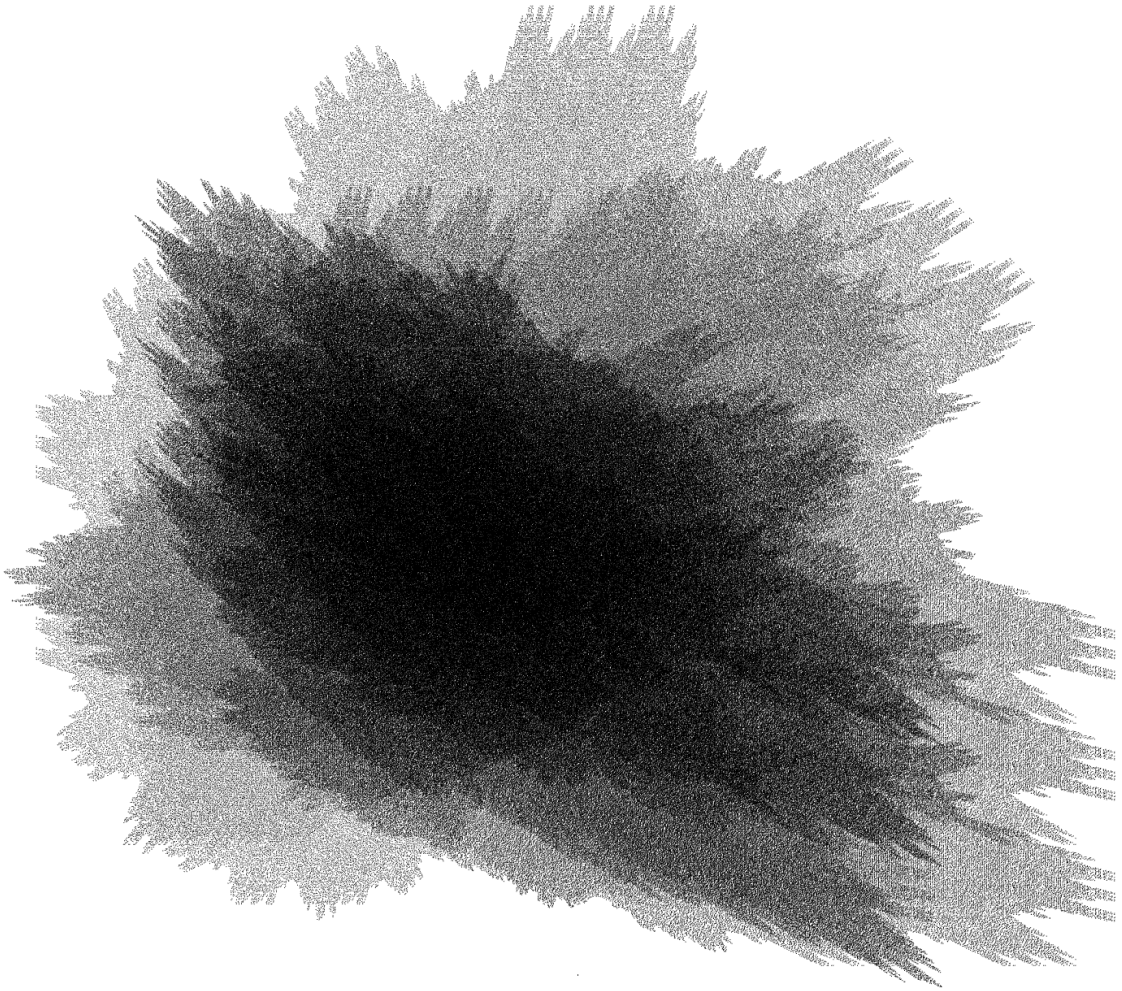
User Manual

`m`-, `sequence`-, `subdivision`-, `tjsr`- and `tmisc`-package for Matlab

(The `t`-packages)



Thomas Mejstrik

November 26, 2018

Contents

1 Installation **3**

2 Overview over the included packages **4**
2.1 Naming/calling conventions and data-format . . . 4
2.2 m-package . . . 5
2.3 sequence-package . . . 5
2.4 subdivision-package . . . 5
2.5 tjsr-package . . . 6
2.6 tmisc-package . . . 7
2.7 test-package . . . 8

3 subdivision-package **9**
3.1 constructordering . . . 9
3.2 getS . . . 10
3.3 blf . . . 12
3.4 tile . . . 13
3.5 constructOmega . . . 13
3.6 constructV . . . 14
3.7 restrictmatrix . . . 15
3.8 transitionmatrix . . . 16
3.9 Example showing how to use the subdivision-package . . . 16

4 tjsr-package **18**
4.1 findsmp . . . 18
4.2 tgallery . . . 20
4.3 invariantsubspace . . . 23
4.4 tjsr . . . 24
4.5 tjsr_getpolytope . . . 36
4.6 preprocessmatrix . . . 36

5 Test-drivers **38**

6 Copyright **39**
6.1 Cover-image . . . 39
6.2 m-, sequence-, subdivision-, tjsr- and tmisc-package . . . 39
6.3 The JSR toolbox [6] . . . 39
6.4 SeDuMi . . . 39

Bibliography **39**

1 Installation

In order to install the `m-`, `sequence-`, `subdivision-`, `tjsr-` and `tmisc`-package do the following steps:

1. Copy the folder containing this file in the directory of your choice. If you change the name of the folder do not put a name with space in it, for instance `'toolbox_t'` and not `'toolbox t'`.

2. Open Matlab, select *Home > Set Path* and add that folder (the folder containing this file) and the sub-folders `m`, `sequence`, `@sequence`,`subdivision`.

   You may also want to copy the folders `test` and `@cell`. These are not necessary but helpful. `test` contains nice examples how to use the packages, `@cell` implements some operations for cell-arrays.

   Manually you can use, in the command line, `addpath <pathFolder>` for this folder and its subfolders. In order to save this for further Matlab sessions type `savepath`.

3. If you have not installed the packages `SeDuMi`, `The JSR toolbox` and the `Gurobi` solver[1] yet, you must install them too.

   The archive containing this file may also contain the folders `SeDuMi` and `JSR_louvain`. Add these folders and all of its sub-folders to the Matlab-path, as described in step 2. If this archive does not contain the `SeDuMi` and `The JSR toolbox`, or they are not working since they may be out-dated, they can be downloaded from sedumi.ie.lehigh.edu and de.mathworks.com/matlabcentral/fileexchange/33202-the-jsr-toolbox

   To install the *Gurobi solver*, you must install it from the Gurobi page http://www.gurobi.com/ and follow the instructions there. Academic users get free full-featured, no-size-limit versions of Gurobi to use in class or for research.

4. The `subdivision-` and the `tjsr`-package need at least Matlab R2016b.

   The `sequence-` and the `subdivision`-package depend on the Matlab `Symbolic Math Toolbox` and the `Signal Processing Toolbox` but also run without the latter.

   The `sequence-` and the `tjsr`-package depend on the Matlab `Parallel Computing Toolbox` but should run without it.

   If these toolboxes are not installed, they should be installed as described in the Matlab documentation.

5. The packages should now be available from any directory. Type `setupt(1)` to run a self-test of all included functions. If the test fails, you may run `setupm(1)`, `setupsequence(1)`, `setupsubdivison(1)`, `setuptjsr(1)` or `setuptmisc(1)` to test the single packages.

The Toolbox has been tested on several OS's (Windows, Linux, Mac) and Matlab versions (R2016b, R2017a, R2017b, R2018b). However, if you encounter any problem please contact us at

tommsch@gmx.at.

[1] The packages also run without the Gurobi solver, but magnitudes slower.

2 Overview over the included packages

This is the documentation for the most important functions from the Matlab-packages `m`, `sequence`, `subdivision`, `tjsr` and `tmisc` (summarized as the `t`-packages). The documentation for all functions (including those in this manual) can be read with the Matlab-command `help name`, where `name` is the name of a function, e.g. `help tjsr`. Some functions have an extended help-file which can be read with `help name>fullhelp`, e.g. `help tjsr>fullhelp`.

The packages can be downloaded from http://tommsch.com and probably from Matlab's file exchange.

2.1 Naming/calling conventions and data-format

- Most functions expect vectors in column format, contrary to Matlab's default. I.e. if you want the column vectors $\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}^T$, $\begin{bmatrix} 7 & 8 & 9 \end{bmatrix}^T$ written in one matrix, you must do it like this[1]

$$\begin{bmatrix} 1 & 7 \\ 2 & 8 \\ 3 & 9 \end{bmatrix}.$$

- All of the functions use name-value pairs to pass options. Some options do not need a value – those can be used with or without a value. If used without a value or with the value 1 the option is enabled, if used with the value 0, the option is disabled. Other arguments behind options which do not expect a value lead to undefined behaviour.
- Optional arguments are written inside of square brackets, except when they are *Options*, i.e. all options are optional.
- All names are singular and written in lower-case, with the only exception when a corresponding mathematical symbol uses an upper-case letter.
- The variables `idx` and `val` in the source-code are used only locally. They are only valid for some lines of code. They are re-used, since Matlab has no scope for variables.
- Since Matlab (nearly) has no types for variables, we denote parameters which shall be whole numbers with the type `integer`, even if they are `double`s in reality.
- The letters `XX` in the source-code, indicates things which should be changed.

[1] Internally most of the functions also use this data-format, in particular `tjsr`. Since sparse arrays in Matlab do not work well with this data-format, in a future releas the function `tjsr` may change its data-format to Matlab's default.

2.2 m-package

This package implements functions which generalize Matlab functions to n-arrays for arbitrary $n \in \mathbb{N}_0$, and give them a consistent behaviour and interface. Mathematical functions which are defined for an arbitrary number of arguments, but Matlab only accepts a small number (usually two), are also included in this package. All functions should have the same input/output format as the original Matlab functions – at least for basic inputs. Most of the functions do not rely on other packages. The function `setupm` performs a self-test of the package.

Some names of functions of this package collide with functions from the Matlab `Mapping Toolbox`, in particular `plotm`, `surfm`.

So far the following functions have been implemented: `allm`, `anym`, `cart2sphm`, `convm`, `dec2base`, `factorialm`, `ind2subm`, `isvectorm`, `kronm`, `lcmm`, `maxm`, `minm`, `nchoosekm` `ndimsm`, `onesm`, `padarraym`[2] `parsem`, `plotm`, `repmatm`, `sizem`, `sph2cartm`, `squeezem`, `summ`, `upsamplem`, `zerosm`,

The package also includes a small set of functions, originally not included in Matlab. These are:

`sph2cartm2` Transforms hypershperical to cartesian coordinates, assumes that the radius is one.

`cart2sphm2` Transforms cartesian to hyperspherical coordinates, but does not return the radius.

`parsem` Parses `varargin` and is easier to use then Matlab's `parse`[3].

`repcellm` Works like `repmat` but returns cell-arrays.

`setupm` Performs a self-test of the `m`-package.

2.3 sequence-package

This package implements the vector space $\ell_0(\mathbb{Z}^s)$, $s \in \mathbb{N}$. The design-goal is, that `sequence`s behaves exactly as arrays (whenever it makes sense) and code written for arrays can be used without modifications for `sequence`s. Unfortunately, direct referencing is not implemented yet. The package depends on the `tmisc` and `m`-package.

The package furthermore depends on the Matlab `Symbolic Math Toolbox`. It also uses functions from the Matlab `Signal Processing Toolbox` but also runs without the latter.

The function `setupsequence` performs a self-test of the package.

So far the following functions are implemented: `characteristic` (χ), `diffsequence` ($\tilde{\nabla}_\mu$, ∇^k), `norm` ($\|\cdot\|_p$), `supp` (supp), `symbol` (the corresponding symbol), `upsample` ($\uparrow_M$), `conv` ($*$), `nnz`, `ndims`, `size`, `ref`, and all point-wise operations.

2.4 subdivision-package

This package implements functions for the work with subdivision schemes. Most of them can be used as a black-box. The package depends on the `m`-, `tmisc`- and `sequence`-package.

The package furthermore depends on the Matlab `Parallel Toolbox` and `Symbolic Math Toolbox`. It also uses functions from the Matlab `Signal Processing Toolbox` but also runs without the latter.

The function `setupsubdivision` performs a self-test of the package.

Important functions

`constructordering` Constructs the data-type ordering.

`getS` Returns subdivision operators in this packages format.

`blf` Plots the basic limit function of a multiple subdivision scheme.

`tile` Plots the attractor corresponding to a multiple subdivision scheme.

`constructOmega` Constructs the set Ω for a multiple subdivision scheme.

`constructV` Constructs a basis for the space $V_k(\Omega)$ [3].

`restrictmatrix` Restricts matrices to a subspace.

`transitionmatrix` Constructs transition matrices.

[2] Written by Notlikethat, stackoverflow.com/users/3156750/notlikethat, (2014) published under Common Creative Licence.
[3] Most functions in the other packages rely on that function, thus it is included here.

num2ordering Computes number expansions for multiple multivariate number systems.

ordering2num Computes multivariate numbers corresponding to multiple number systems.

setupsubdivision Examples how to use the subdivision-package and self-test for the package.

Remaining functions

characteristic Returns the characteristic function of an index set.

compresscoordinates Returns an array representing the graph of a function.

constructdigit Constructs the usual digit set $M[0,1)^s \cap \mathbb{Z}^s$.

constructOmega Constructs the set Ω [3].

constructU Constructs a basis for the space U [2].

constructVbar Constructs a basis for the space $\bar{V}_k(\Omega)$ [3].

dimVVbar Computes the dimension of spaces the $V_k(\Omega)$ and $\bar{V}_k(\Omega)$ [3].

daubechiesmask Returns the mask coefficients for Daubechies' wavelet.

findperiod Searches for periodics in sequences.

isodering Tests whether input is an ordering.

isS Tests whether input is are subdivision operators.

isT Tests whether input are transition matrices.

multiplyS Concatenates subdivision operators.

mask2symbol Computes the symbol of a mask.

normalizeS Normalizes the values of the masks of subdivision operators.

ordering2vector Takes an ordering and returns it as a vector of certain length.

peter Removes randomly columns of arrays.

supp Computes the support of a mask.

symbol2mask Computes the masks from given symbols.

checktile Tests if an attractor is a tile.

tilearea Good heuristic test if an attractor has area one (for dimension 1 and 2).

vector2ordering Wrapper function for `findperiod`.

2.5 `tjsr`-package

This package implements functions to compute the JSR using the modified invariant-polytope algorithm. The package depends on the `m`- and the `tmisc`-package and partly on the `subdivision`-package, but runs also without the latter. The package furthermore depends on `The JSR toolbox` v1.2 (beta) [6], the `SeDuMi`-package v3.2 [10], the `Gurobi`-Solver v8.0 and the Matlab `Parallel Toolbox`.

If the Gurobi-solver is not installed, Matlab-functions will be used instead and the algorithm will run some magnitudes slower. The function `setuptjsr` performs a self-test of the package.

Important functions

findsmp[4] Searches for s.m.p.-candidates. Various algorithms available.

invariantsubspace[5] Searches for invariant subspaces of matrices.

setuptjsr Performs a self-test.

tgallery Returns sets of matrices, mostly used for `tjsr`.

[4] Copyright for algorithm `'genetic'` by [1]. Copyright for algorithm `'gripenberg'` by [6]

[5] Copyright for algorithm `'perm'` and `'basis` by [6]

tjsr[6] Computes the joint spectral radius of the set of matrices.

tjsr_getpolytope Returns the constructed invariant polytope.

preprocessmatrix Simplifies sets of matrices while preserving its JSR.

Remaining functions

binaryMat Returns the sets of binary matrices.

blockjsr Returns the JSR of block diagonal matrices.

codecapacity Given forbidden differences, returns matrices whose JSR is related to their capacity.

computepolytopenorm Computes the Minkowski-norm.

daubechiesmatrix[7] Constructs matrices whose JSR is related to the Daubechies' wavelets regularity.

estimatepolytopenorm Estimates the Minkowski-norm.

estimatejsr Rough estimate of the joint spectral radius.

chooseval Selects highest values of a vector.

intersectinterval Intersects intervals.

leadingeigenvector Returns all leading eigenvectors of matrices.

makeorderinggraph Constructs the graph given a partially ordered set.

makepositive Multiplies arrays such that its first non-zero entry is positive and its norm is preserved.

paritionatepolytope Partitions points in $\mathbb{R}^s$ into clusters of nearby points.

reducelength For patterns, removes repetitions and cycles it such that they have has smallest lexicographic value.

removecombination Constructs a minimal set of cycles.

tbuildproduct_fast[8] Constructs the product of matrices corresponding to a ordering.

tliftproduct[8] Computes all products of length less or equal k.

tliftsemidefinite[8] Computes semi-definite liftings of the matrices M

trho[8] Computes the spectral radius of matrices.

All functions with prefix **tjsr_** are subroutines of **tjsr** and are not documented here, since they are subject to big changes, whenever the main function **tjsr** is changed.

2.6 tmisc-package

This package contains useful functions. Most other packages rely on this package. The package depends on the m-package. The function **setuptmisc** performs a self-test of the package. The function **setupt** performs a self-test of all packages.

Important functions

findperiod Searches for periodics of digit sequences.

grCenter Finds the centre of a tree.

grVerCover Computes all local minimal vertex-covers of a graph.

intersectspace[9] Finds a basis of the intersection of N subspaces.

mixvector Constructs all possible combinations of values[†].

normalize Normalizes matrices in various ways.

[6] Uses code from [6]

[7] Copyright for algorithm **'jung'** by [6]. Copyright for algorithm **'gugl'** by Nicola Guglielmi.

[8] Copyright by [6]

[9] Copyright by Ondrej Sluciak, ondrej.sluciak@nt.tuwien.ac.at

removezeros Deletes zeros in arrays in various ways.

repcell Returns the replicated argument as a cell array.

searchincellarray Searches in cell arrays.

setplus Element-wise addition of vectors.

setuptmisc Performs a self-test.

setupt Performs a self-test of all described packages

tbuildProduct[8] Constructs the product of matrices corresponding to a ordering.

uniquecell Same behaviour as `unique`, but for cell-arrays.

vdisp[10] Compact (but sometimes ugly) display of (nested) objects.

vprintf[10] Powerful version of `sprintf` with the additional specifier `%v`.

Remaining functions

cprintf[11] Displays styled formatted text in the command window.

flatten Changes a nested cell array to a flat cell array.

identifymatrix Returns standard properties of matrices.

issquare Tests if an array is a (hyper)-square.

issym Tests if an object is symbolic.

iswholenumber Tests if an array contains only whole numbers.

limsup Computes the (cumulative) lim sup of vectors.

liminf Computes the (cumulative) lim inf of vectors.

lexicographic Orders vectors in a norm-lexicographic ordering.

nestedcellfun Wrapper function calling `cellfun` for each cell in a (nested) (cell-)array.

nondiag Extracts non-diagonal parts of 2-arrays and 2-cells.

num2color Assigns colours to integers.

savetocellarray Stores values in a cell array corresponding to a linear index-vector.

subsco Indexing of matrices by coordinate-vectors.

tif Ternary if operator.

unflatten Changes a flat cell array to a nested cell array.

2.7 `test`-package

This package consists of examples showing how to make use of the other packages. Most of the functions are not written well and are poorly documented. Still it should be clear from the code how to use them. The package includes

testchecktile Checks if the area of attractors is one.

testfindsmp Compares various algorithms finding s.m.p.-candidates.

testjsr Compares various algorithms computing the JSR.

testnorm Demonstrates how `estimatepolytopenorm` works.

testnorm2 Tests a strange difference-metric which could estimate the Minkowski-norm.

testnormdecay Constructs sequence of matrices with maximal spectral radius below some threshold.

testnumextremalpoints Generates random points and their convex hull.

testtjsr Tests the algorithm `tjsr` using example-matrices.

[10] Uses code by Yair Altman (2015) and code by Stefan, University of Copenhagen
[11] Copyright by Yair Altman (2015)

3 subdivision-package

3.1 constructordering

The ordering in which the subdivision operators are applied for multiple subdivision schemes, is defined by the "data-type" *ordering*. An ordering is a representation of an infinite periodic sequence.[1] It is stored as an 1x2 - cell array, where the first cell is the non-periodic part, and the second cell is the periodic part. Each cell can have an arbitrary number of rows.

The function `constructordering` takes vectors representing the non-periodic parts and the periodic parts of an ordering and returns it as a cell array.

3.1.1 Syntax

```
[ oo ] = constructordering( oo1, [pp1, oo2, pp2, ... ])
```

3.1.2 Input

`oo1` vector of numbers, mandatory
The non-periodic part of the first row.

`pp1` vector of numbers, optional
The periodic part of the first row.

`ooi/ppi` vector of numbers, optional
The periodic part of the `i`th row.

3.1.3 Output

`oo` ordering
The ordering defined by the input arguments.

3.1.4 Note

- If `oo1` is an ordering, `oo1` is returned unchanged.
- All periodic and non-periodic parts must have the same length.
- The number of arguments is either 1 or an even number.

3.1.5 Example Usage

- `constructordering([1 2],[3 3])` returns `{[1 2],[3 3]}` which corresponds to the infinite sequence $1, 2, 3, 3, 3, 3 \ldots$.
- The number $\frac{1}{3}$ can be represented by `constructordering([],[3])`. Since *ordering*s do not encode a decimal point, this sequence could also stand for the number $3.333\cdots$, etc..

[1] Thus the multiple subdivision schemes defined by this package can be seen as stationary schemes.

3.2 getS

This function returns *subdivision operators*. These are cell arrays, each row describing on subdivision operator, with entries

`{ mask (a), dilation (M), digit-set (D), ..., name (n)}`.

The variables are:

`mask (a)` n-array
Mask a of the subdivision operator.

`dilation (M)` $n \times n$ matrix
Dilation matrix M of the subdivision operator.

`digit-set (D)` $n \times |\det M|$ matrix
Set of representatives $D \simeq \mathbb{Z}^s / M\mathbb{Z}^s$.

`name (n)` string
Name of the subdivision operator. Is only used for displaying purposes.

In future releases of the package, data-fields may be added to subdivision-operators. The `name` will always be the last entry in each row.

The function `getS` returns examples of *subdivision operator*s in the described format.

3.2.1 Syntax

```
[ S ] = getS(dim || cellarray || 'name' || list, [options])
```

3.2.2 Input

`dim` integer
Returns all unnamed subdivision operators saved in the file.

`cellarray` cell-array `{[a],M,[D],[n]}` where `a`, `M`, `D`, `n` is as described above.
Takes a subdivision operator (or parts from it) and computes the missing variables. Only `M` is mandatory and thus the cell array must be at least of size 1×2.

If `D` is not given, $D := M\mathbb{Z}^s \cap \mathbb{Z}^s$, where s is the dimension of M. If `a` is not given $a := \chi D$, where χ is the characteristic function.
If `n` is not given $n :=$ `unnamed`.

`name` string
Returns the named subdivision operator saved in the file with name `name`.

`list` name-value pairs of strings
The possible names are `'a'` or `'mask'`, `'M'` or `'dilation`, `'D'` or `'digit'`, `'n'` or `'name'`. The possible values are as described above.

Only of this inputs can be used.

3.2.3 Options

`'supp'` Instead of the digit sets, the support of the masks is returned as the digit sets.

`'Omega'` Instead of the digit sets, the support of the masks minus the digit sets is returned as the digit sets.

`'characteristic'` Instead of the masks, the characteristic function of the digit sets is returned as the masks.

`'nocheck'` Disables basic checks of the data-integrity.

`'bigcheck'` Enables checks about necessary conditions of convergent subdivision operators.

`'verbose',val` integer, default: 1
Verbose level.

3.2.4 Output

`S` cell array of subdivision operator(s)

3.2.5 Note

All subdivision operators in a cell array should be for the same dimension.

3.2.6 Possible values for `name`

The possible strings (among some other strings) are returned with the command `gets('help')`. The following list is only a subset of all possible strings.

Univariate schemes

`'1_all'` Returns all named subdivision schemes of dimension 1.

`'1_rand'` Returns a random subdivision scheme of dimension 1.

`'1_4point'` The 4-point scheme $a = \frac{1}{16}\left[-1 \quad 0 \quad 9 \quad 16 \quad 9 \quad 0 \quad -1\right]$, $M = 2$.

`'1_DD'` The Dubuc-Deslauriers sch. $a = \frac{1}{256}\left[3 \quad 0 \quad -25 \quad 0 \quad 150 \quad 256 \quad 150 \quad 0 \quad -25 \quad 0 \quad 3\right]$, $M = 2$.

`'1_strange_interpolatory'` An interpolatory which does not fulfil $a(0) = 1$.

`'1_Hassan_Dodgson_3point'` The Hassan-Dodgson 3-point scheme $a = \frac{1}{16}\left[1 \quad 5 \quad 10 \quad 10 \quad 5 \quad 1\right]$, $M = 3$.

`'1_balanced_ternary'` The balanced ternary number system $M = 3$, $D = \{-1,\ 0,\ 1\}$.

`'1_devil_stairs'` A subdivision scheme whose basic limit function is the devil-stairs function.

`'1_spline_binary-2'` The second order B-Spline scheme.

`'1_three_disjoint_Om'` A scheme with has three disjoint invariant sets Ω [3].

`'1_daubechies',n` The n^{th} Daubechies wavelet scaling function.

`'1_cantor'` A scheme whose attractor is the Cantor-set.

Bivariate schemes

`'2_all'` Returns all named subdivision schemes of dimension 2.

`'2_rand'` Returns a random subdivision scheme of dimension 2.

`'2_sierp'` A scheme whose attractor is the Sierpinski triangle.

`'2_rqj43'` Example from [9, Example 4.3].

`'2_butterfly'` The butterfly scheme.

`'2_twindragon'` Scheme whose attractor is the twindragon.

`'2_twindragon'` Scheme whose attractor is the flash.

`'2_V0neqV0bar_1'` Scheme where $V_0 \neq \bar{V}_0$ [3].

`'2_superomega'` Scheme whose super-tile is the cover-image.

`'2_McLure'` Non-integer scheme with self affine 4-tile.

Quadrovariate schemes

`'4_cex_Pot97'` Dilation matrix which does not posses a digit set such that the corresponding attractor is a tile [8].

3.2.7 Example Usage

`getS(2)` Returns all unnamed subdivision operators of dimension 2. The returned set may be empty, if there are no unnamed subdivision operators.

`getS('2_butterfly')` Returns the butterfly scheme.

`getS('1_all')` Returns all named subdivision operator of dimension 1.

3.3 blf

This function plots the basic limit function of multiple subdivision schemes.

3.3.1 Syntax

```
[ c, PM, xyzv, oo ] = blf( [oo], S, [options] )
```

3.3.2 Input

`[oo]` ordering, optional
Defines the ordering in which the subdivision operators are applied. If `oo` is not given, a random ordering is computed.

`S` subdivision operators (or something else)
The parameter `S` is passed to `getS` and the returned value is used.

3.3.3 Options

`'diff',val` $1 \times dim$-vector or integer, default: 0
Computes (partial) derivatives (finite differences). If `diff` is a vector the given partial derivative is computed. If `diff` is a scalar, then all partial derivatives of that order are computed.

`'plot',cellarray` cell array or scalar,default: `{}`
Arguments passed to `plotm`, e.g. `{'Color','red'}`. If `plot` $= 0$, nothing is plotted.

`'verbose',val` integer, default: 1
Verbose level.

`'start'` dim-array,default: δ_0
Starting sequence.

`'iteration',val` integer, default: depends on `S`
Computation stops after `iteration` many iterations.

`'maxiteration',val` integer, default: 50
Maximum number of iterations.

`'numpoint',val` integer, default: 30000
Number of points in the output-sequence to be computed.

`'maxnumpoint',val` integer, default: 30000
Maximum number of points in the output-sequence to be computed.

3.3.4 Output

`c` dim-array or cell array of dim-arrays.
Iterated sequence, i.e. `c` $= S_{\texttt{oo}_n} \cdots S_{\texttt{oo}_2} S_{\texttt{oo}_1}(\texttt{start})$. If `diff` is given and there is more then one partial derivative of the demanded ordering, `c` is a cell array.

`PM` $dim \times dim$ matrix
Dilation matrix corresponding to the returned mesh.
There is a bug, and the returned matrix be the transposed.

`xyzv` $dim + 1 \times N$ matrix
Column vectors v containing the function values at the position $xyz \in \mathbb{R}^{dim}$.

`oo` vector
Ordering used. Equals `oo` (Input) if given.

3.3.5 Example Usage

```
[c,PM,xyzv,So]=blf('2_butterfly','iteration',3,'diff',1)
```

3.4 tile

Plots the attractor corresponding to a multiple subdivision scheme.

3.4.1 Syntax

```
[ Q, oo ] = tile( [oo], S, [options])
```

3.4.2 Input

`[oo]` ordering, optional
The ordering in which the subdivision operators are applied

`S` subdivision operators
The subdivision operators

3.4.3 Options

`'supertile',n` integer
Computes the supertile instead of the attractor.

`'verbose',val` integer, default: 1
Verbose level.

`'round',val` integer or 1×2-vector,default: `1e-2, 1e-12`
If `round` is an integer, the same round value is used in each iteration. If `round` is a vector, the round values are linearly interpolated.

`'start',array` *dim*-array, default: δ_0
The starting set.

`'digit'` default: false
Computes iterated digit sets instead of the attractor.

`'plot',cellarray` cell array or scalar,default: `{}`
Arguments passed to `plotm`, e.g. `{'Color','red'}`. If `plot` $= 0$, nothing is plotted.

`'iteration',val` integer, default: depends on `S`
Computation stops after `iteration` many iterations.

`'maxiteration',val` integer, default: 50
Maximum number of iterations.

`'numpoint',val` integer, default: 30000
Number of points in the output-sequence to be computed.

3.4.4 Output

`Q` $dim \times N$ matrix
The computed attractor.

`oo` vector
Ordering used. Equals `oo` (Input) if given.

3.4.5 Example Usage

```
tile('2_frayed_squares','round',[.1 1e-2])
tile('1_cantor','digit')
tile([getS('2_rand'); getS('2_rand')],'supertile','iteration',10)
```

3.5 constructOmega

Constructs the set Ω as described in [3].

3.5.1 Syntax

```
[ Om ] = constructOmega( S, [options] )
```

3.5.2 Input

`S` cell array of subdivision operators

3.5.3 Options

`'Omega',val` matrix, optional, default: automatically computed
Starting set. If not given, a point is computed where to start from. In some cases, the algorithm may not find a good starting point, and the returned set is not minimal.

`'verbose',val` integer, default: 1
Verbose level.

`'lexicographic'` default: false
Orders the output set lexicographically.

`'stable'` default: false
Does not order the set the output set.

3.5.4 Output

`Om` matrix
The set Ω.

3.5.5 Example Usage

```
constructOmega('2_butterfly','Omega',[2;2],'stable')
```

3.6 constructV

Constructs a basis for the space $V_k(\Omega)$, as described in [3].

3.6.1 Syntax

```
[ V ] = constructV( Om, [k] , [options])
```

3.6.2 Input

`Om` $dim \times N$ array of column vectors.
The set for which V_k shall be constructed

`[k]` integer or a vector of integers greater equal zero, optional
Index or indices k.

3.6.3 Options

`'verbose',val` integer, default: 1
Verbose level.

`'01'` Allows to give input `Om` as a logical array, e.g. `constructV( [1 0; 1 1; 1 0], 0, '01','verbose',2)` is equivalent to `constructV( [0 0; 1 0; 1 1; 2 0]' , 0,'verbose',2)`.

3.6.4 Output

`V` matrix (or cell array of matrices) of column vectors.
The basis (bases) for the space V_k. If k is empty, then all spaces V_k are computed for which $\dim V_k > 0$.

3.6.5 Note

The functions `constructOmegabar` and `constructU` construct the spaces $\bar{V}_k$ and U_k as described in [3]. They have nearly the same interface as `constructV`, so they are not described in this manual. See the `help` of these functions for more information.

3.6.6 Example Usage

```
Om=constructOmega('2_butterfly'); constructV(Om,1)
```

3.7 restrictmatrix

Restricts matrices to a subspace and checks whether the subspace is invariant or not, i.e. with the notation from below,

$$\mathtt{TT} = \begin{bmatrix} \mathtt{TA} & * \\ \mathtt{NULL} & \mathtt{TR} \end{bmatrix} = \mathtt{BASIS}^{-1} \cdot \mathtt{T} \cdot \mathtt{BASIS}. \tag{3.1}$$

3.7.1 Syntax

```
[ TA, TT, TR, NULL, BASIS ] = restrictmatrix( T, A, [options] )
```

3.7.2 Input

`T` square matrix or cell array of square matrices
The matrices to be restricted

`A` rectangular matrix
Matrix with the columns of the subspace as basis

3.7.3 Options

`'verbose',val` integer, default: 1
Verbose level

`'epsilon',val` double,default: 10^{-12}
The matrices are considered to be invariant if a norm of the residua `NULL{i}` is greater then $dim \cdot$ `epsilon`.

`'smallsize'` Removes all columns of `A` (starting from the last column) without changing the rank of `A`.

3.7.4 Output

If the input `T` is a cell array, then the outputs `TA`, `TT`, `TR` and `NULL` are also cell arrays. `BASIS` is always a matrix.

`TA` $dim_A \cdot dim_A$ matrix (or cell array of)
Restriction of `T` to `A`

`TT` $dim_T \cdot dim_T$ matrix (or cell array of)
`T` in the basis of `A` complemented to a basis of $\mathbb{R}^{dim_T}$.

`TR` $dim_{T-A} \cdot dim_{T-A}$-matrix (or cell array of)
Lower right corner of matrix `TT`.

`NULL` $dim_{T-A} \cdot dim_A$-matrix (or cell array of)
Lower left corner of `TT`. If `T` is `A` invariant, then `NULL` consists of zeros only.

`BASIS` $dim_T \cdot dim_T$ matrix (or cell array of)
Complemented basis of `A` to a basis of $\mathbb{R}^{dim_T}$.

3.7.5 Example Usage

```
restrictmatrix([1 1 0; 0 1 1; 1 0 1],[1 -1 0; 0 1 -1]')
```

3.8 transitionmatrix

Constructs transition matrices $T_{d,\Omega} = (a(\alpha - M\beta + d))_{\alpha,\beta\in\Omega}$ where a are subdivision masks, M are dilation matrices, $d \in D \simeq \mathbb{Z}^s/M\mathbb{Z}^s$, all three taken from a set of subdivision operators, and $\Omega \subseteq \mathbb{Z}^s$ is an $T_{\mathbb{Z}^s}$ invariant set.

3.8.1 Syntax

```
[ T, Om ] = transitionmatrix( S, [options] )
```

3.8.2 Input

`S` cell array of subdivision operators

3.8.3 Options

`'Omega',val` $dim \cdot N$ integer matrix, default: is automatically constructed using `constructOmega`
Index set used for construction of the transition matrices.

`'onlyindices'` Function instead returns $dim + \#\Omega \times \#Omega$ matrix with the indices used for construction of the transition matrices (instead of the values of the mask a at the indices position).

`'noflat'` Function instead returns cell array of cell arrays, each corresponding to one subdivision operator.

`'infindices'` Entries of indices not element of the support of the masks, are replaced by ∞ instead of 0.

`'colsum',val` double, default: column-sum of the first column of the first transition matrix in `T`
Tests if the columns sum up to `colsum`.

`'verbose',val` integer,default: 1
Verbose level.

3.8.4 Output

`T` cell array of matrices
The transition matrices.

`Omega` The set Ω used to compute the transition matrices.

3.8.5 Example Usage

```
vdisp(transitionmatrix(1/4*[1 4 3]',2))
```

3.9 Example showing how to use the subdivision-package

This example computes the Hölder regularity of the stationary subdivision scheme given by the subdivision operator $S = (a, M)$, with mask $a = \frac{1}{4}\begin{bmatrix} 1 & 1 & 3 & 3 \end{bmatrix}$ and dilation $M = 2$. We first generate the cell array subdivision operators

```
S=getS('a',1/4*[1;1;3;3],'M',2);
   %There is a bug in Matlab which throws an error for the command disp(S).
   %Thus the semi-colon is important.
```

To plot the basic limit function we call

```
blf(S);
```

We next construct the transition matrices and restrict them to the invariant subspace V_0. We also construct the space $\bar{V}_0$. To apply the joint spectral radius approach to compute the regularity of subdivision-schemes, the dimension of $V_0(\Omega)$ and $\bar{V}_0(\Omega)$ must coincide.

```
[T, Om]=transitionmatrix(S); %compute transition matrices and the set Omega
plotm(Om,'x') %plot the set Omega
V0=constructV(Om,0)                    %Construct space orthogonal to polynomial sequences
V0bar=constructVbar(Om,0);             %Construct space of differences of delta
if(size(V0,2)~=size(V0bar,2));
    fprintf('Set Omega is wrong.');    %Test ifV0==V0bar
end
TV0=restrictmatrix(T,V0); vdisp(TV0); %restrict transition matrices and display them
```

The last step is to compute the Hölder regularity using the modified invariant polytope algorithm.

```
[JSR, type]=tjsr(TV0);
al=-log(JSR)/log(2) %Hoelder regularity. The 2 comes from the dilation M=2.
```

4 tjsr-package

4.1 findsmp

This function searches for s.m.p.-candidates using various algorithms, these are: the *modified Gripenberg algorithm* as described the acompanying paper and used by default, the *Gripenberg algorithm* as implemented in [6], a standard *brute force algorithm* and the *genetic algorithm* as implement in [1]. All algorithms share the same input and output format, thus this function can be used to easily compare them

It must be noted, that the modified Gripenberg algorithm is parallelised, whereas the others are not. The genetic algorithm has a bug and sometimes reports wrong spectral radii and candidates. Gripenbergs algorithm and the genetic algorithm only return *one* s.m.p.-candidate. Thus they are not suitable for finding smp-candidates as needed for the (modified) invariant polytope algorithm.

4.1.1 Syntax

```
[ cand, nearlycand, info ] = findsmp( T, [algorithm], [options] )
```

4.1.2 Input

`T` cell array of square matrices of the same size
The input matrices.

`[algorithm]` (optional), string
Algorithm to use:

`'modifiedgripenberg'` (default) Modified Gripenberg algorithm.

`'gripenberg'` Gripenbergs algorithm [4].

`'bruteforce'` Brute force algorithm.

`'genetic'` Genetic algorithm [1].

4.1.3 Output

`cand` cell array of column vectors
Ordering of the products with highest normalized spectral radius.

`nearlycand` cell array of column vectors
Orderings of products with nearly highest normalized spectral radius.

`info` struct
Additional info, depending on the used algorithm and options. May contain the following fields:

`info.time` double
Time in seconds needed for the computation.

`info.jsrbound` 1x2 vector
Bounds for the JSR.

`info.graph` Matlab graph object
All computed norms and spectral radii as directed graph.

`info.count` integer
Approximate number of computed matrices.

4.1.4 Options

Options available for all algorithms

`'verbose',val` integer, default: 1
Defines the verbose level.

`'maxtime',val` double, default: ∞
Maximal runtime in seconds (checked at the beginning of each iteration).

`'norm',handle` function-handle, default: `@norm`
Used norm in the computation

Options for algorithm `'modifiedgripenberg'`

`'delta',val` double, default: 0.99, (experimental, may be removed)
Delta used in the Gripenberg Algorithm. All products of matrices with normalized norm less than $\rho_c/$`delta`, where ρ_c is the current upper bound for the JSR, are thrown away.

`'N',val` integer, default: depends on the size and number of matrices
Number of kept products in each step. If `N` $= \infty$, the algorithm behaves like Gripenberg's algorithm.

`'maxsmpdepth',val` integer, default: $\simeq 100$
Maximal length of products searched for.

`'JSR',val` double, default: 0
Candidates to be found must have at least `JSR` spectral radius.

`'searchonlyonecandidate'` default: false
Searches until one candidate is found, `maxsmpdepth` is ignored. .

`'sufficientbound',val` double, default: ∞
Algorithm terminates if a candidate with bound higher than `sufficientbound` has been found.

`'plot',string` string, default: 0
Some plot output. If `string` = `'tree'` a graph showing all computed orderings of products is plotted and also saved in `info.graph`.

`'nearlycandidate',val` double, default: 0.99
Candidates with normalized spectral radius greater than `nearlycandidate*info.bounds(1)` are considered as nearly-s.m.p.s.

`'shortnearlycand',val` double, default: 1
Removes all nearly-s.m.p.s whose ordering is longer than `shortnearlycand`*maximal-length-of-s.m.p.-candidates.

`'sparse',flag` boolean, default: auto
Flag if the matrices are sparse.

`'select',flag` integer, default: 1
Strategy how to select new candidates.

- `0` (bad) Choose the N products with biggest norm.
- `1` (default) Choose the $N/2$ products with biggest and $N/2$ products with smallest norm.
- `2` (not so good) Choose N random products.

Options for algorithm `'genetic'`

`'popsize',int` integer, default: 300, minimum:10
Population size.

`'maxgen',int` integer,default: 1000
Maximum number of generations.

`'maxstall',int` integer, default: 15
Maximum number of stalling iterations before increasing the maximum product length.

`'maxtotstall',int` integer, default: 100
Maximum number of stalling iterations before terminating the algorithm.

`'mutantprop',val` double, default: 0.3
Mutation probability of a given product.

`'muteprop',val` double, default: 0.2
Mutation proportion of a given product.

Options for algorithm `'gripenberg'`

`'delta',val` double, default: 0.99
Delta used in the Gripenberg Algorithm. All products of matrices with normalized norm less than $\rho_c/$`delta`, where ρ_c is the current upper bound for the JSR, are thrown away.

`'maxeval',val` double, default: ∞
Maximum number of evaluations.

`'maxtime',val` double, default: ∞
Maximal time used for computation.

`'maxsmpdepth',val` double, default: 10
Maximal length of products searched for.

Options for algorithm `'bruteforce'`

`'minsmpdepth',val` integer, default: 1
Minimal length of products.

`'maxsmpdepth',val` integer, default: 10
Maximal length of products.

`'JSR',val` , double, default: 0
Candidates must have at least `JSR` spectral radius.

`'searchonlyonecandidate'` default: false
Searches until one candidate is found, `maxsmpdepth` is ignored.

4.1.5 Example Usage

```
[ c, nc, info ] = findsmp( [1 -1; 3 -2], [1 3; -1 -1], 'maxsmpdepth', 15 )
[ c, nc, info ] = findsmp( [1 -1; 3 -2], [1 3; -1 -1], 'plot', 'tree' )
[ c, nc, info ] = findsmp( [1 -1; 3 -2], [1 3; -1 -1], 'gripenberg' )
```

4.2 `tgallery`

This function provides example-sets of matrices. It is useful for testing algorithms and other purposes. It also makes use of Matlab´s `gallery`.

4.2.1 Syntax

```
[ val ] = tgallery( what, dim, N, k, [options] )
```

4.2.2 Input

`what` string, mandatory
Controls the return value.

`dim` integer, mandatory in most cases
Dimension of matrices. In some cases

`N` integer, mandatory in most cases
Number of matrices.

k anything, mandatory in some cases
Number of matrices.

The variables may have another meaning in some cases, as described below.

4.2.3 Options

'verbose',val integer, default: 1
Verbose level.

'seed',val integer or struct returned by **rng**, default: empty
Seed for random number generator. If **seed** is set, Matlab's random number generator has the same state before and after execution of the function.

'sparse',val double, default: 0
Returns sparse matrices

'sparse',val double, default: 0
Returns sparse matrices

'int' flag
Returns integer matrices.

'pos' flag
Returns matrices with positive entries.

'rho' flag
Returns matrices with spectral radius 1.

'norm' flag
Returns matrices with 2-norm 1.

4.2.4 Output

val cell array of square matrices
The returned matrices.

4.2.5 Notes

Most of the options preserve no special properties of the matrices. E.g.: The entries in the matrix returned by `tgallery('rand_gauss',10,1,'pos')` do not have a Gaussian distribution anymore.

4.2.6 Possible values for `what` and their mandatory arguments

We list all possible values for **what**, followed by the mandatory parameters. For example,

```
tgallery('rand_pm1', 2, 1, 'seed', 10)
```

returns a cell array with one element, containing the 2×2 matrix

$$\begin{bmatrix} 0 & 0 \\ 1 & -1 \end{bmatrix}.$$

Random matrices

'rand_stochastic',dim,N Column-stochastic matrices.

'rand_doublestochastic',dim,N Double-stochastic matrices.

'rand_stochastic_neg',dim,N Column-stochastic matrices with positive and negative values.

'rand_doublestochastic_neg',dim,N Double-stochastic matrices with pos. and neg. values.

'rand_TU',dim,len Transition matrices of a subdivision scheme in $\mathbb{R}^{\texttt{dim}}$ with arbitrary dilation matrix and mask with $\texttt{len}^{\texttt{dim}}$ non-zero entries, restricted to the subspace U as defined in [2].

'rand_TV0', dim, len Transition matrices of a subdivision scheme in $\mathbb{R}^{\texttt{dim}}$ with arbitrary dilation matrix and mask with $\texttt{len}^{\texttt{dim}}$ non-zero entries, restricted to the subspace V_0 as defined in [3].

'rand_pm1',dim,N Random matrices with values -1, 0, 1.[1]

'rand_bool',dim,N Random matrices with values 0, 1.[1]

'rand_gauss',dim,N Matrices with normally distributed values …

'rand_equal',dim,N Matrices with equally distributed values in $[-1,\ 1]$ …

'rand_unitary',dim,N Unitary matrices.

'rand_zero',dim,N Matrices with spectral radius 0.

'rand_colu_0',dim,N Matrices with pos. entries, column-norm=1 and i.i.d. singular values.

'rand_colu_1',dim,N Matrices with non-neg. entries, column-norm=1 and i.i.d. singular values.

'rand_corr_0',dim,N Correlation matrices with pos. entries and i.i.d. singular values.

'rand_corr_1',dim,N Correlation matrices with non-neg. entries and i.i.d. singular values.

'rand_hess',dim,N Orthogonal upper Hessenberg matrices.

'rand_orthog_1' Orthogonal matrices with complex leading eigenvalue.

'rand_orthog_2' Orthogonal matrices with complex dual leading eigenvalue.

Matrices from applications

'binary',dim,N,k Matrices with linear entries which equal the number k in base 2. If there exists $\tilde{\texttt{k}} < \texttt{k}$ such that the returned set for $\tilde{\texttt{k}}$ would be the same, the function returns the empty set.

'binary2',dim,N,k The same as **'binary'**, but if there exists $\tilde{\texttt{k}} < \texttt{k}$ such that the returned set for $\tilde{\texttt{k}}$ would have the same JSR, the function returns the empty set.

'cex' Pair of 2x2 matrices which has no s.m.p. [11], returned approximately with 61 digits.

'code',C (C is a cell array of row-vectors) Matrices whose JSR is related to the capacity of a code with forbidden differences C. See **codecapaciy** for more information.

'daub',dim Matrices whose JSR is closely related to the Hölder-exponent of Daubechies' wavelets.

'euler',dim Matrices in connection with the Euler partition function [5].

'nondominant' Set $\{\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix},\ \frac{4}{5}\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}\}$ with non-dominant s.m.p..

Matrices from papers

'grip_p52' Matrices $\{\frac{1}{5}\begin{bmatrix} 3 & 0 \\ 1 & 3 \end{bmatrix},\ \frac{1}{5}\begin{bmatrix} 3 & -3 \\ 0 & -1 \end{bmatrix}\}$ from [4, p. 52].

'grip_p45' Matrices $\{\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix},\ \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}\}$ from [4, p. 45].

'morris_p3' Matrices $\{\begin{bmatrix} 2 & 2 \\ 0 & 0 \end{bmatrix},\ \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}\}$ from [7, p. 3].

'prot2012_p35' Example for a Pascal rhombus [5, p. 35].

'prot2012_p40' Example from [5, p. 40].

'prot2012_p43' Example for a Euler binary problem [5, p. 43].

'prot2012_p44' Example for a Euler ternary problem [5, p. 44].

'prot2016' Matrices from a subdivision scheme [5, p. 33, p. 35, p. 50].

'mejstrik_longsmp',x Matrices $\tilde{\mathcal{C}}_x = \{\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix},\ \begin{bmatrix} 0 & 0 \\ x & 0 \end{bmatrix}\}$ with s.m.p.-length $\approx e \cdot x$.

[1] **rand_pm1** and **rand_bool** are good example to test the balancing of multiple trees in the invariant polytope algorithm.

'mejstrik_119' Matrices $\mathcal{X} = \{ \begin{bmatrix} \frac{15}{92} & \frac{-73}{79} \\ \frac{56}{59} & \frac{89}{118} \end{bmatrix}, \begin{bmatrix} \frac{-231}{241} & \frac{-143}{219} \\ \frac{103}{153} & \frac{-38}{65} \end{bmatrix} \}$ with s.m.p.-length 119.

'mejstrik_119_2' Matrices $\mathcal{X} = \{ \begin{bmatrix} \frac{2936824268245481}{18014398509481984} & \frac{-1040403609775579}{1125899906842624} \\ \frac{8548815905856585}{9007199254740992} & \frac{6793656059295549}{9007199254740992} \end{bmatrix}, \begin{bmatrix} \frac{-8633497502531381}{9007199254740992} & \frac{-2940665869269935}{4503599627370496} \\ \frac{6063507393887449}{9007199254740992} & \frac{-5266479768915639}{9007199254740992} \end{bmatrix} \}$ with s.m.p.-length 119.

4.2.7 Example Usage

```
tgallery('rand_sparse_pos',100,2,[],0.01)
tgallery('rand',4,2,100)
```

4.3 invariantsubspace

Searches for invariant subspaces of matrices $M \in \mathcal{M}$, i.e. a change of basis B such that all matrices $M \in \mathcal{M}$ have block-triangular form in basis B. Uses three different algorithm, `permTriangul` and `jointTriangual` from [6] as well as an implementation of [2]. The returned matrices still may have invariant subspaces which can be found found using this algorithm.

4.3.1 Syntax

```
[ Mret, B ] = invariantsubspace( M, ['type'], [options] )
```

4.3.2 Input

`M` cell array of matrices
The matrices.

`['type']` (optional), string, default: `'auto'`

`'none'` Nothing happens, `Mret={M}`, `B=eye(dim)`.

`'perm'` Tries to find a permutation such that all `M{i}` are in block-diagonal form.

`'basis'` Tries to find a basis such that all `M{i}` are in block-diagonal form.

`'trans'` If `M` are transition matrices coming from a subdivision-scheme, the algorithm tries to find subspaces of differences U, as described in [2]. First computes numerically, then symbolically, then using `vpa`.

`'auto'` (default) The algorithm tries `'perm'`, `'basis'` then `'trans'` (numerically).

4.3.3 Options

`'verbose',int` integer, default: 1
Verbose level.

4.3.4 Output

`Mret` cell array of matrices
The blocks in the block diagonal of the matrices in basis `B`.

`B` matrix
Basis, i.e. `B^(-1)*M{i}*B` has block-diagonal form.

4.3.5 Example Usage

```
[ M, B ] = invariantsubspace( {[1 0 ; 1 2], [3 -1; -1 3]}, 'basis', 'verbose', 2 )
```

4.4 tjsr

Computes the JSR of a set of square-matrices and can be used as a black-box (whenever the algorithm reports no errors).

4.4.1 Syntax

```
[ JSR, info, allinfo ] = tjsr( M, [options] )
```

4.4.2 Input

`M` Cell array of matrices
The input matrices whose JSR shall be computed.

4.4.3 Important options

We first list the options which are the most important. Those will be sufficient for the standard-user. At the end we list all available parameters.

`'proof'` default: false
Proofs the invariance of the polytope after termination of the algorithm and returns bounds for the JSR w.r.t. that polytope. The proof uses Matlab functions and is *very* slow and may even fail in some cases.

`'maxsmpdepth',int` integer, default: depends on the size and number of matrices
Maximal length of s.m.p.-candidates to search for.

`'plot',string` string, default: `'none'`

- `'norm'` Plots intermediate norms
- `'polytope'` Plots the constructed polytope
- `'L'` Plots the number of vertices left to compute (at the moment)

`'nearlycandidate',val` double, default: $\simeq 0.9999$
Relative difference between s.m.p.-candidates and nearly-s.m.p.s. If you are sure that a certain s.m.p.-candidate is an s.m.p., but the algorithms returned intermediate bounds stuck on some level, try to play with that value.

`'ordering',cell` cell array of matrices of column vectors, default: empty
Orderings of s.m.p.-candidates.

`'nobalancing'` default: false
Disables balancing.

`'balancingvector',val` vector, default: empty
If given, these numbers are used to balance the multiple cyclic trees. The vector must have as many entries as there are cyclic-roots (including extra-vertices). Otherwise the algorithm hase undefined behaviour.

`'invariantsubspace',string` string, default: `'auto'`
Whether to search for invariant subspaces or not, which can take a long time.

- `'none'` Nothing happens, `Mret={M}`, `B=eye(dim)`.
- `'perm'` Tries to find a permutation such that all `M{i}` are in block-diagonal form.
- `'basis'` Tries to find a basis such that all `M{i}` are in block-diagonal form.
- `'trans'` If `M` are transition matrices coming from a subdivision-scheme, the algorithm tries to find subspaces of differences U, as described in [2]. First computes numerically, then symbolically, then using `vpa`.
- `'auto'` (default) The algorithm tries `'perm'`, `'basis'` then `'trans'` (numerically).

`'delta',val` double, default: 1
Accuracy. For `delta` < 1 algorithm is faster, but returns only bounds for the JSR.

`'verbose',val` integer, default: 1
Verbose level. If `verbose` < 0 the algorithm suppresses error-messages (not recommended!).

4.4.4 Notes

- The algorithm is parallelised and starts the default Matlab-pool if there is no pool available. If a special pool is needed (e.g. a non-local pool), it has to be started by the user prior starting the `tjsr`-algorithm.
- The algorithm is split up into three main-functions, responsible for the following tasks:
  - `tjsr`: Preprocessing the input; Starting the Matlab pool; Restarting the algorithm with different parameters (if needed); Postprocessing the output.
  - `tjsr_preworker`: Finding invariant subspaces, starting `tjsr_worker` for each subspace; Finding candidates; Balancing trees; Setting up the cyclic-root.
  - `tjsr_worker`: Computing the invariant polytope.
- Most of the sub-routines of the algorithm are in separate files with the prefix `tjsr_`. We do not described these functions here, since they are subject to big changes whenever the main function `tjsr` is changed. Nevertheless, most of them have a documentation inside of their source-code.

4.4.5 Output

Screen Output

Output written in red must be read. For some options, the algorithm delivers wrong results and these messages warn in these cases. These messages are printed again after termination of the algorithm.

Output in front of the progress bar

- `Time: x/y` Time needed for last iteration/total time needed for building the tree.
- `JSR = [ x, y ]` Interval in which the JSR lies.
- `norm = x` Current minimal computed norm of the polytope.
- `In: x, Out: y` Number of points which lie inside or outside the polytope, checked by estimating the Minkowski-norm.
- `#test: x/y` Number of points to test in this iteration/number of points to check in total
- `#V: x/y` Number of points in simplified polytope/number of points in polytope in total
- `'Test old vertex'` Old vertices of polytope get estimated again.

Output in the progress-bar

- `i` Vertex is proofed to be inside of the polytope, but norm is unknown
- `x` Vertex is proofed to be outside of the polytope, but norm is unknown
- `_` Vertex is inside of polytope
- `.` Vertex is machine-epsilon-near to old vertex and is usually not considered to compute the norms of new vertices
- `,` Vertex is 1000*machine-epsilon-near to old vertex
- `o` Vertex is slightly outside
- `O` Vertex is far outside
- `m` negative value occurred during computation of norm, vertex is added
- `8` Inf occurred during the computation of the norm, vertex is added
- `?` NaN or Inf occurred during the computation of the norm, vertex is added
- `E` Some error occurred during the computation of the norm, vertex is added

Verbose levels higher than 2 generate much more output, which is not described here.

Data Output

`JSR` double or 1x2-vector.
Interval containing the exact value for the JSR or an interval containing the JSR.

`info` struct
Contains nearly all data which was generated during the computation. Most important fields are described here, all other fields are described below.

`info.cyclictree.ordering` cell array of matrices of column vectors
All orderings used for the roots of the cyclic tree.

`info.cyclictree.smpflag` vector
Defines what the orderings are: 0 : s.m.p.-candidate, 1 : nearly-s.m.p, 2 : extra-vertex.

`info.cyclictree.V` cell array of matrices
All generated vertices, each column is one vertex. To obtain the (invariant) polytope call `tjsr_getpolytope(info)`

`info.info.errortext` string
All important error-messages.

`info.JSR` double or 1x2-vector
The same as `JSR`.

`info.counter` struct
Some self-explaining numbers.

`info.block` cell array of structs, only returned if there are invariant subspaces
If returned, each cell in `info.block` contains the `info`-struct for that block. The sub-structs `info.info` and `info.counter` contains aggregated informations from `info.block{:}`.

`allinfo` cell array of structs
If the allgorithm restarts, only the `info` struct of the very last run is returned (to save memory). All other `info`-structs are returned as the cell array `allinfo`.

4.4.6 Example usage

The algorithm (in the optimal case) does not need to be called with any parameters, i.e. a call of the form `tjsr(A)`, where `A` is the cell array of matrices whose JSR shall be computed, is sufficient. Nevertheless, in some examples the algorithm does not work as expected and manual interaction is necessary.

For our examples in this section we make use of the function `tgallery` (see Section 4.2) which returns examples matrices. Some options used for these examples are described in detail only in the later Section 4.4.7.

- This example shows how to specify the cyclic-roots. We use for the example the set of matrices `A=tgallery('rand_pm1',3,2,'seed',100)`, i.e. $\mathtt{A} = \{\mathtt{A}_1, \mathtt{A}_2\}$,

$$\mathtt{A}_1 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & -1 \\ 1 & -1 & -1 \end{bmatrix}, \quad \mathtt{A}_2 = \begin{bmatrix} 0 & 0 & 1 \\ -1 & -1 & 0 \\ 1 & 1 & -1 \end{bmatrix}.$$

  The set `A` has an s.m.p. $A_1 A_2^8$, which can be computed with `tjsr(A)`.

  The command `tjsr(A,'ordering',{[2]'})` starts the algorithm with a wrong s.m.p.. The algorithm finds better s.m.p.-candidates and restarts several times. Note that automatic extra-vertices are disabled, if we specify an ordering.

  If we want to add extra-vertices we can do it in two ways.

  ```
  tjsr(A,'extravertex',{[.1 0 0]'})
  ```

  – This command specifies only the extra-vertex. The s.m.p.-candidates and nearly-s.m.p.s are computed automatically.

  ```
  tjsr(A,'ordering',{[1 2 2 2 2 2 2 2 2]', []'},'smpflag',[0 2],'v0',...
    {[0.058585928823279 -0.687551547968790 0.723768303968635]', [.1 0 0]'})
  ```

- This command specifies vectors of the cyclic-roots. The downside is, that one need to give the exact eigenvalues of all s.m.p.-candidates and nearly-s.m.p.s.
- The option `'smpflag',[0 2]` specifies that we want two cyclic-roots. The first is the root corresponding to an s.m.p.-candidate (number `0`), the second root corresponds to an extra-vertex (number `2`). If we also want to specify a root for a nearly-s.m.p., we have to use the number `1`.
- The option `'v0',{[0.05859 -0.68755 0.72377]' [.1 0 0]'}` specifies the eigenvectors/vectors used to start the cyclic-root. They must be given as a cell array of column vectors.
- If one also wants to specify the dual-leading eigenvectors, this has to be done using the option `'v0s'`.
- The option `'ordering',{[1 2 2 2 2 2 2 2 2]', []'}` specifies the orderings of the cyclic-roots. The first is the ordering of the s.m.p.. *Orderings of extra-vertices MUST be empty.*

  Note that *orderings MUST be given as column vectors*, and the they are written in reversed polish notation, i.e. the ordering `1 2 3` is the product $\mathtt{A}_3\mathtt{A}_2\mathtt{A}_1$.

  If there is more than one ordering corresponding to a vector `v0`, it must be given as a matrix. E.g.: `'ordering',{[1 2 2 2; 1 2 2 0]'}` means that $\mathtt{A}_2^3\mathtt{A}_1\mathtt{v0}_1 = \mathtt{A}_2^2\mathtt{A}_1\mathtt{v0}_1 = \mathtt{v0}_1$.

- This example presents balancing, automatic extra-vertices and approximate computation options. We use for the example the set of Daubechies-matrices `D7=tgallery('daub',7)`.

  `tjsr(D7)` computes that this set has two s.m.p.s, $\mathtt{D7}_1$ and $\mathtt{D7}_2$.

  `tjsr(D7,'balancingvector',[1 1.02 .01 .01 .01 .01 .01])` Uses the given balancing vector. This option is useful when one wants to proof the invariance of the invariant polytope by hand.

  `tjsr(D7,'nobalancing')` This command disables the balancing (and the algorithm will not terminate). By setting the verbose level to 4, `tjsr(D7,'nobalancing','verbose',4)`, we see that the third line of numbers does not stop to grow. This line corresponds to the number of added vertices of the third cyclic-tree.

  `tjsr(D7,'autoextravertex',0)` This command disables the auto extra vertices. Since the polytope for these matrices is very flat, the LP-program fails to compute the Minkowski-norm and reports all vertices to be outside (This can be seen by the fact, the the computed norms are ∞ always).

  `tjsr(D7,'nobalancing','delta',.99999)` This command multiplies the matrices prior computing the cyclic-tree by 0.99999. Thus the algorithm will terminate, although we did not balance the trees. Clearly, the returned value is not exact but an interval.

  `tjsr(D7,'epspolytope',-.1)` This command influences when a vertex is considered to be inside of the polytope. A negative value means, that even points which are outside are considered to be inside. The algorithm automatically increases the value of `epspolytope` whenever there are no vertices left which can get children until the value is bigger than `epslinprog`.

  This option speeds up the computation of the invariant polytope at the beginning, but in total leads to much bigger polytopes and a slowdown of the algorithm. For approximate computation of the JSR, the option `'delta'` is preferable.

- This example presents invariant subspace options. We use for the example the random boolean matrices, `B=tgallery('rand_bool',4,2,43)`.

  `tjsr(B)` finds two invariant subspaces.

  `tjsr(B,'invariantsubspace','none')` disables the search for invariant subspaces. In some cases, the search for invariant subspaces may take a long time, especially when the number of matrices is big or the dimension is high.

- This example presents some plotting options. We use for the example a random set of 10 non-negative matrices with spectral radius 1 and dimension 20,
  `T=tgallery('rand_rho_pos',20,10,116)`.

  `tjsr(T,'plot','norm')` Plots the computed norms of vertices and colors them according whether they have children or not.
  It is also possible to plot the norms using the string `'info_norm'`, but then the plot does not look as interesting.

`tjsr(T,'plot','L')` Plots the number of added vertices and the number of remaining vertices to compute. The graph usually has the shape of a Gaussian.

`tjsr(tgallery('rand_rho',6,2,100),'plot','tree')`[2] Plots the graphs of the cyclic trees.
To change the labels of the vertices, change the code in `tjsr_plotoutput`.

`tjsr(T,'plot','polytope')` Plots the polytope/cone. If the dimension is higher than 3, a random subset of directions is chosen to be plotted in each iteration.

`tjsr(T,'plot','info_normest')` Plots the estimated Minkowski-norms.
The prefix `'info_'` allows to plot any data contained in the `info`-struct. Fields in the sub-struct `cyclictree` can be addressed directly. All others need to be called with their full name. Thus the option `'plot','info_normest'` is equivalent to
`'plot','info_cyclictree.normest'`

`tjsr(T,'plot','info_normest_norm','fastnorm',0)` Plots the real norms against the estimated Minkowski-norms. We have to add the option `'fastnorm',0` since otherwise the Minkowski norms of some points would not be computed.
If one wants to plot more data from the `info`-struct, the variables to be plotted have to be seperated with an underscore `_`.

`tjsr(T,'plot','info_normest_norm_rho','fastnorm',0)` Plots the estimated Minkowski-norms against the real norms against the spectral radii.
Also more than two variables are possible.

`tjsr(T,'plot','info_normest_L')` Plots the estimated Minkowski-norms and the number of processed vertices. If the variables are not compatible in size or format, the algorithm tries to plot them anyhow.

4.4.7 All options

Pre- and postprocessing options

controlling pre-processing and post-processing steps.

`'clc'` default: false
Clears the console before starting the algorithm.

`'maxnumrestart',int` integer, default: 10
Maximum number of restarts.

`'pauseonreset',flag` boolean, default: false
Waits for a key-press after every restart.

`'proof',flag` boolean, default: false
Proofs the invariance of the polytope after termination of the algorithm and returns bounds for the JSR w.r.t. that polytope.

- `0` Do not test the invariance.
- `1` Use the original matrices to test.
- `2` Use the normalized matrices to test, i.e. the set `M`/ JSR.

This option works only if there are no invariant subspace (and under some other conditions). The test is done using Matlab's linprog and without any tricks speeding-up the computation, implying it is *very slow*.

`'nopreprocess'` default: false
If this option is not set, the input matrices are preprocessed using the following steps:

- Equal matrices are removed from the input set (all but one).
- All matrices in `M` are multiplied with the number modulus 1, s.t. all first non-zero entries are positive.

[2] Since the tree for the matrices `T` is too big, we use a different set of matrices for that example.

Preworker options

controlling the search for candidates, nearly-candidates and extra-vertices, etc..

`'JSR',val` 1x2-vector, default: empty
An initial true! estimate for the JSR. The value is used (amongst other things) to search for s.m.p-candidates. The option may be removed in a future release.

`'ordering,val` cell array of matrices of column vectors, default: empty
The orderings of the s.m.p.-candidates, nearly-candidates and extra-vertices. Extra-vertices have empty ordering. Each cell contains all the orderings belonging to the same leading eigenvalue. If the orderings have different length, zeros must be appended.
E.g.: `'ordering',{[1 2; 2 0]',[1 1 2]'}`.

`'smpflag',val` row-vector, default: empty
Defines whether the candidates are an s.m.p. or not. 0=candidate, 1=nearly-candidate, 2=extravertex. If `smpflag` is given, `ordering` must be given too.
E.g.: `'smpflag',[0 1]`

`'v0',val` cell array of column vectors, default: empty
The corresponding eigenvectors to the candidates/the starting vectors. If `v0` is given, `ordering` must be given , `v0s` should be given.

`'v0s',val` cell array of column vectors, default: empty
The corresponding dual eigenvectors to the candidates. If `v0s` is given, `v0` must be given.

`'balancingdepth',val` integer, default: 4
Depth used for balancing multiple trees. If the balancing takes too long, try to decrease that value.

`'balancingvector',val` vector, default: empty
Balancing-vector. Must have as many entries as there are cyclic trees.
E.g.: `'balancingvector',[1 0.8]`.

`'nobalancing'` default: false
Disables balancing. This is equivalent to `'balancingvector',[1 1 1 ... 1]`.

`'extravertex',val` cell array of column vectors, default: empty
Extra-vertices can be given in two ways: Either as a cell array of vectors as argument of `'extravertex'`, or in the cell array of `'v0'` with corresponding `smpflag` set to 2.
E.g.: `'extravertex',{[0.1 0 0]', [0 0.1 0]'}`

`'multiplicity',val` vector, default: empty
The multiplicity of the corresponding leading eigenvalues in `v0`. This option is nearly useless and only sometimes used to choose between algorithms (P) and (R).

`'autoextravertex',val` double, default: 0.1
Adds a vertex for all directions whose corresponding singular value is less than `autoextravertex`.

`'maxnumcandidate',val` integer, default: `numel(M)*10` (subject to be changed)
If there are more candidates than `maxnumcandidate`, the algorithm restarts and `maxsmpdepth` is reduced.

`'nearlycandidate',val` double, default: $\simeq 0.9999$
Maximal relative difference between normalized spectral radii of s.m.p.-candidates and nearly-s.m.p.s.

`'noclassify'` default: false
By default, all s.m.p. candidates (and their cyclic permutations) are examined whether they have same eigenvectors, in which case they are grouped together and only one tree is built up for them. This may take a long time in some cases. This options disables that behaviour. Default=false.

`'invariantsubspace',string` string, default: `'auto'`
Whether to search for invariant subspaces or not, which can take a long time.
See `invariantsubspace` for more information.

- `'none'` Nothing happens, `Mret={M}`, `B=eye(dim)`.
- `'perm'` Tries to find a permutation such that all `M{i}` are in block-diagonal form.
- `'basis'` Tries to find a basis such that all `M{i}` are in block-diagonal form.

'trans' If M are transition matrices coming from a subdivision-scheme, the algorithm tries to find subspaces of differences U, as described in [2]. First computes numerically, then symbolically, then using vpa.

'auto' (default) The algorithm tries 'perm', 'basis' then 'trans' (numerically).

'nomultipleeigenvector' default: false
Only takes one leading eigenvector per candidate, even if there are more.

'complexeigenvector',flag integer, default: 2
Defines how complex eigenvectors shall be treated.

0 Complex eigenvectors are kept.

1 Complex eigenvectors are removed, if there is also a real eigenvector corresponding to the same product.

2 (default) Complex eigenvectors are removed, if there is at least one real eigenvector among all products.

3 Real vectors are computed which span the real subspace of the complex leading eigenvectors (not implemented yet).

4 Complex eigenvectors are removed.

'minJSR', val double, default: 0
Minimal value of normalized spectral radius of s.m.p.-candidates to be found. This option should not be used, since it is ignored most times.

'maxsmpdepth',int integer, default: depends on the size and number of matrices
Maximal length of s.m.p.-candidates to search for.

'findsmp_N',int integer, default: depends on the size and number of matrices
Number of products kept in each step of the findsmp algorithm. See findsmp for more information.

'delta',val double, default: 1
Matrices are multiplied by delta after the construction of the cyclic tree. A smaller value leads to faster convergence, but the algorithm cannot return the exact value of the JSR anymore.

Worker options

controlling how the invariant polytope is built up.

'algorithm',val integer, default: empty
The norm to be used:

0 or 'P' $\|\cdot\|_{\mathrm{co}_- V}$, i.e. cone-norm.

1 or 'R' $\|\cdot\|_{\mathrm{co}_s V}$, i.e. symmetrized polytope-norm.

2 or 'C' $\|\cdot\|_{\mathrm{absco}\, V}$, i.e. complex polytope-norm.

[] (default) The algorithm is determined automatically.

'fastnorm',val integer, default: 1
Whether the norms shall be estimated prior their exact computation.

0 Do not estimate.

1 (Default) Check only whether points are inside.

2 Check whether points are inside or outside.

'naturalselection',int integer, default: depends on the number of available threads
Minimum number of vertices whose norms are computed in each level.

'naturalselectiontype',val integer, default: +inf
How to select new vertices.

inf or -inf (Default=+inf) Use three times norm-estimate, one time parent-Minkowski-norm.

1 or -1 Use norm-estimate. Fastest type, but the intermediate bounds converge slowly.

2 or -2 Use parent-Minkowski-norm.

`3 or -3` Use spectral radii of matrix-products.

`100 or -100` (for debugging) Use negative spectral radii of matrix-products.

If the value is positive, then all children of a vertex are selected, if at least one is selected. If the value is negative, the algorithm can't produce intermediate bounds for the JSR, but may be faster.

`'testoldvertex',val` integer, default: 1
Whether old vertices shall be estimated if they lie inside the polytope or not. In some cases, this option tremendously increases the performance of the algorithm.

`0` Never

`1` (Default) Sometimes

`2` Always

`'simplepolytope',val` integer, default: 10^{-8}
Vertices with distance roughly less than `simplepolytope` to the polytope are not used in the norm computation. `simplepolytope` should be greater than `epslinprog`.

`'epspolytope',val` double, default: $\simeq 2 \cdot 10^{-9}$
Vertices with norm bigger than $1 -$ `epspolytope` are considered to be outside the polytope. Value is automatically increased during the computation if it less then `epslinprog`, in particular `epspolytope` can be less then zero. See the example section for more information.

Termination options

controlling the termination of the algorithm. Note that most of criteria are only tested at the beginning of each iteration.

`'maxstepnumber',val` double, default: ∞
Computation stops if more than `maxstepnumber` vertices are tested.

`'maxtreetime',val` double, default: ∞
Computation stops after the construction of the tree takes more than `maxtreetime` seconds.

`'maxiteration',val` double, default: ∞
Computation stops after iterating the algorithm `maxiteration` times.

`'maxtime',val` double, default: ∞
Computation stops after `maxtime` seconds.

`'maxstepnumber',val` double, default: ∞
Computation stops if the polytope has more than `maxstepnumber` vertices.

`'testeigenplane',val` double, default: $-\infty$ (i.e. this option is disabled)
Tests whether a vertex lies more than `testeigenplane` outside of the supporting eigen-planes defined by the candidates. If a vertex lies outside, the candidates are not s.m.ps. Positive values report a lot of false-positives. If `testeigenplane` $= 1$, the algorithm changes the value to -10^{-10}. This behaviour may be changed in a future release.

`'testspectralradius',val` double, default: -10^{-10}
Tests whether the intermediately occurring spectral radii are greater than $1 +$ `testspectralradius`. Positive values report usually false-positives. If `testspectralradius` $= 1$, the algorithm changes the value to -10^{-10}. This behaviour may be changed in a future release.

`'validateupperbound',val` double, default: 0
Algorithm terminates if upper estimate of JSR is less than `validateupperbound`. Also changes `epspolytope`. If `validateupperbound` < 0, this option is ignored.

`'validatelowerbound',val` double, default: ∞
Algorithm terminates if the lower estimate of the JSR is greater than `validatelowerbound`.

Output options

controlling the output during the computation, and partly also the return-values.

`'plot',string` string, default: empty

- `'norm'` Plots intermediate norms
- `'polytope'` Plots the constructed polytope
- `'L'` Plots the number of vertices left to compute (at the moment)

See `tjsr_plotoutput` for further options.

`'verbose',val` integer, default: 1
Verbose level. If `verbose` < 0 the algorithm suppresses error-messages (not recommended!). Default=1.

`'diary'` integer
Starts the Matlab diary and sets the default value for `save` to 2.

`'profile'` integer
Starts the Matlab profiler.

`'save',val` integer, default: 0
How much of the output (diary, plots, variables) shall be saved to disk.

- `0` (Default) Do not save output.
- `1` Save output at end.
- `2` Save output after each iteration.
- `3` Save output after each iteration in different files.

Debug options

merely for testing the algorithm (should not be changed by the standard-user).

`'balancing'` If set, this indicates that we are balancing.

`'memory'` If set, algorithm tries to save memory (Not available at the moment).

`'alwaysout'` If set, then all points are assumed to be outside of the invariant polytope.

`'epsequal',val` double, default: 10^{-12}
Epsilon used to compare floating numbers for equality.

`'epslinprog'` double, default: depends on the LP-solver
Epsilon used for the linear programming part. If the Gurobi-solver is used, this value is fixed to 10^{-9}. If Matlab's `linprog` is used, this value must be $\geq 10^{-10}$.

`'waitafterbalancing'` If set, algorithm waits for key-pressing after balancing.

`'rholeqval',val` If set, matrix products whose spectral radius is greater than `rholeqval` $+ 10 *$ `eps` are discarded and it is assumed their respective vertices are inside of the polytope.

`'showquantity,val` double, default: 25
Controls up to which size sets of vectors, matrices, etc.. are displayed.

4.4.8 All elements in the `info`-struct

This struct contains nearly all the computed data by the algorithm. It consists of several sub-structs which are explained here. Only a subset of those entries are returned always. These are `info.JSR`, `info.info.infotext` and `info.info.errorcode`.

`info.JSR` double or 1x2-vector
Value or bound for the JSR.

`info.M_normalized` cell array of matrices
Preprocessed and scaled input matrices `M`.

info.M_original cell array of matrices
Original input matrices M.

info.arguments cell array
Processed calling arguments

info.arguments_raw cell array
original calling arguments.

info.balancing cell array of structs
Information obtained during balancing. Contains a subset of the entries in cyclictree.

info.counter struct
Information about how often things happend.

info.cyclictree struct
The invariant polytope (cyclic tree).

info.info struct
Various data

info.lambda double
Normalized spectral radius of the s.m.p.-candidate. Usually equals the first entry in JSR.

info.opt struct
Used options.

info.counter

This sub-struct contains mostly self-explaining data.

info.counter.iteration integer
How often the algorithm iterated.

info.counter.numberofvertex integer
Number of vertices in the polytope.

info.counter.numblock integer
Number of invariant blocks.

info.counter.nummatrix integer
Number of input matrices.

info.counter.numcandidate integer
Number of s.m.p.-candidates.

info.counter.numnearlycandidate integer
Number of nearly-candidates.

info.counter.numextravertex integer
Number of extra-vertices.

info.counter.numordering integer
Equals numcandidate + numnearlycandidate + numextravertex.

info.counter.numstepbig integer
Number of steps done by the linear-programming port.

info.counter.numstepsmall integer
Number of processed vertices by the algorithm.

info.counter.starttime vector
Time when the algorithm started.

info.counter.starttreetime vector
Time when the construction of the tree started.

info.counter.totaltime double
Time needed to terminate.

info.counter.treetime double
Time needed to build up the tree.

`info.cyclictree`

This sub-struct contains the data of the cyclic tree. There are three types of main data structures present.

- Vectors (with as many elements as there are cyclic trees). Each number corresponds to one cyclic tree.
- Cell arrays (with as many elements as there are cyclic trees). Each entry corresponds to one cyclic tree.
- Cell arrays (with as many elements as there are cyclic trees) of matrices. Each column of a matrix corresponds to a vertex of the cyclic tree.

In the following we use the name *ordering* for candidates, nearly-candidates and extra-vertices.

`info.cyclictree.ordering` cell array of matrices of column vectors
The orderings of the candidates, nearly-candidates and extra-vertices. The latter have empty ordering.

`info.cyclictree.smpflag` row-vector
Defines what the orderings are: 0 : s.m.p.-candidate, 1 : nearly-s.m.p, 2 : extra-vertex.

`info.cyclictree.v0` cell array of column vectors
Starting vector for each ordering.

`info.cyclictree.v0s` cell array of column vectors
Dual vector for each ordering.

`info.cyclictree.balancingvector` row-vector
Balancing factors.

`info.cyclictree.multiplicity` row-vector
Multiplicity of the orderings vectors.

`info.cyclictree.orho` vector
Normalized spectral radii of each ordering. Extra-vertices have the value `NaN`.

`info.cyclictree.oclass` cell array of matrices of column vectors
Orderings of products which are already contained in the cyclic tree. This field may be removed in a future release.

`info.cyclictree.maxlengthordering` integer
Maximal length of the orderings for each tree.

`info.cyclictree.L` cell array of row-vectors
Number of vertices in each tree.

`info.cyclictree.livingvertex` row-vector
Number of vertices without children which are not inside the polytope.

`info.cyclictree.timelvl` row-vector
Time spent for each iteration.

`info.cyclictree.normlvl` row-vector
Computed norm in each iteration. This sequence is not necessarily monotonic.

`info.cyclictree.level` cell array of row-vectors
Number of the iteration in which the corresponding vertex was added.

`info.cyclictree.norm` cell array of row-vectors
Computed norm of the vertices.

`info.cyclictree.normest` cell array of row-vectors
Estimated norm of the vertices.

`info.cyclictree.normparent` cell array of row-vectors
Computed norm of the parent vertices.

`info.cyclictree.o` cell array of matrices of column-vectors
Ordering of the product to obtain the respective vertex in `info.cyclictree.V`.

`info.cyclictree.parent` cell array of row-vectors
Index of parent-vertex.

`info.cyclictree.rho` cell array of row-vectors
Spectral radii of the product to the corresponding vertices.

info.cyclictree.status cell array of row-vectors
Indicates whether a vertex has children (1) or not (0).

info.cyclictree.V cell array of matrices of column-vectors
All generated vertices. To obtain the (invariant) polytope call `tjsr_getpolytope(info)`.

info.cyclictree.Vs cell array of matrices of column-vectors
All generated dual vertices. Usually only those for the cyclic root are computed.

info.cyclictree.V_intermediate cell array of matrices of column-vectors
Vertices of the polytope, only used internally.

info.cyclictree.ub_intermediate double
Upper bound for the JSR, only used internally.

info.info

Contains mostly info and error data.

info.info.errorcode integer
Termination-code. Negative values are successful termination, all other values indicate bad termination.

- `-80` Worker was not started due to user-input (not used)
- `-60` JSR is less than `validateupperbound`.
- `-50` JSR is greater than `validatelowerbound`.
- `-40` Exact value was found during balancing (not used).
- `-20` Algorithm terminated successfully and there were invariant subspaces
- `-10` Algorithm terminated successfully.
- `-5` Algorithm terminated successfully and `delta` > 0.
- `0` Input error.
- `inf` Unkown error.
- `nan` Strange error.
- `10` No candidate was found.
- `20` Candidate is no s.m.p..
- `30` No balancing vector found.
- `60` Candidate with higher normalized spectral radius found.
- `70` `maxtime` reached.
- `75` `maxtreetime` reached.
- `80` `maxstepnumber` reached.
- `90` `maxvertexnumber` reached.
- `100` `maxtreedepth` reached.
- `110` Some vertex lies outside the supporting eigenplanes, thus the candidate is no s.m.p.
- `120` `maxiteration` reached.
- `130` Too much candidates found.
- `170` An error in an invariant subspace occured (The algorithm usually does not recover from that error and aborts).
- `180` JSR is likely to be zero (This algorithm cannot handle that case).
- `1000` Complex leading eigenvectors (The algorithm cannot handle that case at the moment).

info.info.errorinformation usually cell array but may have different format
Used to pass information of errors from `tjsr_worker` back to `tjsr`.

info.info.infotext string
Most of (and even more) output text.

info.info.errortext string
All error-messages.

info.info.dim integer
Dimension of the input-matrices.

info.info.algorithm integer
Used algorithm. 0=cone (case (P)), 1=polytope (case (R)), 2 complex-polytope (case (C)).

info.info.matrixtype struct
Self explaining properties of the input matrices.

info.info.findsmp struct
Data returned from **findsmp**.

info.opt

Struct where all described options are saved.

info.block

Only set if there are invariant subspaces. If so, each cell in **info.block** contains the **info**-struct for that block and **info.info** and **info.counter** contains aggregated informations from **info.block{:}**.

4.5 tjsr_getpolytope

Returns the vertices of the invariant polytope.

4.5.1 Syntax

```
[ VV ] = tjsr_getpolytope( info )
```

4.5.2 Input

info info-struct as returned by **tjsr**.

4.5.3 Output

VV matrix of column vectors
The invariant polytope.

4.5.4 Example Usage

```
[JSR,info]=tjsr(tgallery('rand_gauss',3,2,'seed',10))
VV=tjsr_getpolytope(info)
plotm([VV -VV],'resolution',0,'MarkerSize',100)
info.opt.plot='polytope'
tjsr_plotoutput(info)
```

4.6 preprocessmatrix

4.6.1 Syntax

Simplifies sets of matrices while preserving its joint spectral radius (for default-values).

```
[ M ] = preprocessmatrix( M, [options] )
```

4.6.2 Input

M cell-array of matrices
The input matrices

4.6.3 Options

`'inverse',bool` boolean, default: false
Takes the Moore-Penrose pseudo-inverse of all matrices.

`'addinverse',bool` boolean, default: false
Adds the Moore-Penrose pseudo-inverses to the returned set.

`'transpose',bool` boolean, default: false
Returns the transposed matrices.

`'addtranspose',bool` boolean, default: false
Adds the transposed matrices to the returned set.

`'makepositive',bool` boolean, default: true
Multiplies each matrix with the unique number modulo 1, such that the first non-zero entry of each matrix is positive.

`'timestep',val` double, default: false
Returns the matrices $I \cdot (1 - \texttt{timestep}) + \texttt{M\{i\}} \cdot \text{valtimestep}$ where I is the identity matrix. This transformation is used in connection with the computation of the stability of linear systems.

`'perturbate',val` double, default: 0
Perturbates the matrices randomly by $\texttt{randn} \cdot \texttt{perturbate}$.

`'removezero',bool` boolean, default: true
Removes all (but one) zero matrices.

`'removeduplicate,bool` boolean, default: true
Removes duplicates. Uses no tolerance for comparing floating point numbers.

`'basechange,val` various data-type, default: 0
Performs a base change.

`0` No base change is made.

`'random'` A random base change is made.

`1` The eigenvectors of `M{1}` is used, implying that `M{1}` is in Jordan-normal-form. If `M{1}` is badly scaled, `M{2}` is used, etc.

`A` (where `A` is an invertible matrix). The matrices $\texttt{A}^{-1}\texttt{M\{i\}A}$ are returned.

`'exponential',val` boolean, default: false
The matrix exponential of each matrix is taken.

`'nodouble',val` boolean, default: false
Matrices are not converted to double.

`'verbose',int` integer, default: 1
Verbose level.

If no options are given, the matrices are processed in the order as written above. The second to last step is `'makepositive'` again. If at least one option (except 'verbose') is given, only that option is used.

4.6.4 Output

`M` cell array of matrices
The processed matrices. If no matrices are removed or added, the returned array has the same size and topology as the input array. Otherwise it is a vector.

4.6.5 Example Usage

```
preprocessmatrix({[-1 2; 2 3],[-1 2; 2 3]})
```

5 Test-drivers

Apart from the described example-usages in this documentation, and the examples in the help of each function, the functions `setupm`, `setupsequence`, `setupsubdivision`, `setuptjsr` and `setuptmisc` also contain examples.

The function `setupt` calls each of these functions and succeeded on the architectures

- Intel Core i5-4670S@3.8GHz, 8GB RAM,
  Linux 4.15.0-38, Ubuntu 16.04.5 LTS,
  Matlab R2017a,
  The JSR toolbox v1.2b, SeDuMi Toolbox v1.32, with and without Gurobi v8.0.1,
- Intel Xeon IvyBridge-Ep E5-2650v2@2.6GHz, 64GB RAM,
  Linux 3.10.0, CentOS Linux 7,
  Matlab R2016b/R2017b/R2018b
  The JSR toolbox v1.2b, SeDuMi Toolbox v1.32, with and without Gurobi v7.5.1/Gurobi v8.0.1,
- Intel Core i5-760@2.8GHz, 8GB RAM,
  Windows 7 SP1,
  Matlab R2017a,
  The JSR toolbox v1.2b, SeDuMi Toolbox v1.32, with and without Gurobi v8.0.1,
- [1]Intel Core i7@2.5GHz, 16GB RAM,
  OSX 10.10.5 (Yosemite), macOS 10.14.1 (Mojave),
  Matlab R2017b,
  The JSR toolbox v1.2b, SeDuMi Toolbox v1.32, Gurobi v8.0.1.

The packages do not run on Matlab versions prior R2015b.

[1]Parts of the `SeDuMi`-package did not work on this configuration. But these parts are not used by the `m-`, `sequence-`, `subdivision-`, `tjsr-` and `tmisc-package`.

6 Copyright

6.1 Cover-image

6.2 `m-`, `sequence-`, `subdivision-`, `tjsr-` and `tmisc`-package

6.3 The JSR toolbox [6]

6.4 SeDuMi

Bibliography

[1] V. Blondel, C.T. Chang, *A genetic algorithm approach for the approximation of the joint spectral radius*, 30^{th} Benelux Meeting on Systems and Control, (2011), perso.uclouvain.be/chia-tche.chang.

[2] M. Charina, V.Yu. Protasov, *Regularity of anisotropic refinable functions*, Appl. Comput. Harm. A., (2017).

[3] M. Charina, T. Mejstrik, *Multiple multivariate subdivision schemes: matrix and operator approaches*, J. Comput. Appl. Math., in press.

[4] G. Gripenberg, *Computing the joint spectral radius*, Linear Alg. Appl., 234, 43–60, (1996).

[5] N. Guglielmi, V.Yu. Protasov, *Exact Computation of Joint Spectral Characteristics of Linear Operators*, Found. Comut. Math., 13, 37–39, (2013).

[6] J.M. Hendrickx, R.M. Jungers, G. Vankeerberghen, *JSR: A Toolbox to Compute the Joint Spectral Radius*, Conf. on Hybrid Systems: Computation and Control Proc., (2014). de.mathworks.com/matlabcentral/fileexchange/33202-the-jsr-toolbox

[7] I.D. Morris, *A rapidly-converging lower bound for the joint spectral radius via multiplicative ergodic theory*, Adv. Math. 225 (6), 3425–3445, (2010).

[8] , A. Potiopa, *A problem of Lagarias and Wang*, Master thesis, Siedlce University, Poland, (1997).

[9] Chen D.R., Jia R.Q., S.D. Riemenschneider, *Convergence of Vector Subdivision Schemes in Sobolev Spaces*, Appl. Comput. Harmon. Anal. 12 (1), 128–149, (2002).

[10] J.F. Sturm, *Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones*, Optim. Methods Softw. 11–12, 625–653, (1999), sedumi.ie.lehigh.edu

[11] V.D. Blondel, J.N. Tsitsiklis, *The boundedness of all products of a pair of matrices is undecidable*, Syst. Control Lett., 41(2), 135–140, (2000).