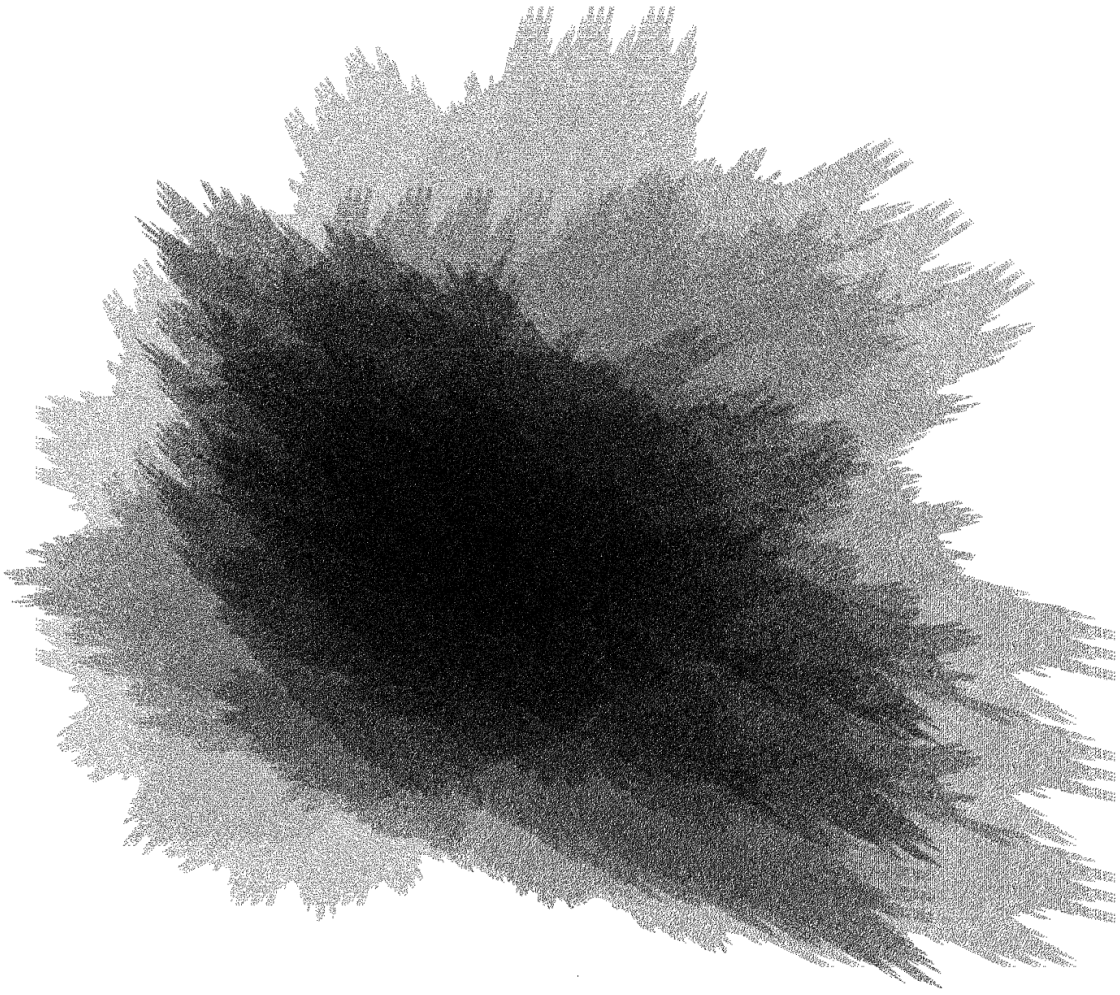
User Manual

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

(The t-packages)
v1.0 – 2 March 2019



Thomas Mejstrik

March 2, 2019

Contents

1 Installation and Requirements

1.1 System requirements

In order to use the packages, you 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 can be installed as described in the Matlab documentation.

1.2 Installation

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

1. Copy the folders `m`, `sequence`, `@sequence`, `subdivision`, `tjsr` and `tmisc` in the directory of your choice.

   You may also want to copy the folder `@cell`, which is not necessary, but helpful and implements some operations for cell-arrays.

2. Open Matlab, select *Home > Set Path* and add the folder of your choice and the sub-folders `m`, `sequence`, `subdivision`, `tjsr`, `tmisc` and `tJSR_louvain`.

   In order to save this for further Matlab sessions type `savepath`[1].

3. If you have not installed the `Gurobi` solver[2] yet, you must install it too. To install the `Gurobi` solver, you must download it from the Gurobi page http://www.gurobi.com/ and follow the instructions there. Academic users get free versions of Gurobi.

4. 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 architectures (Windows, Linux, Mac) and Matlab versions (R2016b, R2017a, R2017b, R2018a, R2018b). However, if you encounter any problem please contact the author at

tommsch@gmx.at.

[1] This command may not have any effect, if you start Matlab from a wrong working directory and/or you do not have privileges to change the Matlab path.

[2] 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 of the 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 (maybe) 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 two column vectors $\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}^T$, $\begin{bmatrix} 7 & 8 & 9 \end{bmatrix}^T$ written in one matrix, all of the packages function expect them as[1]

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

- All of the functions use *name-value* pairs to pass options, e.g. `tjsr(...,'plot','norm')`, where `'plot'` is a *name* and `'norm'` is a *value*. Some names expect boolean values, in which case the value can be omitted. 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 in this documentation, with the exception when they are *Options*, i.e. all options are optional.

- All function-names/names/values/etc. are singular and written in lower-case, with the only exception when a corresponding mathematical symbol uses an upper-case letter.

- 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 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.

- The letters `XX` in the source-code, indicates things which should be changed.

2.2 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` contain examples too. They also test every function included in the packages.

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, with and without Gurobi v8.0.1,

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

- 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, with and without Gurobi v7.5.1/Gurobi v8.0.1,
- Intel Core i5-760@2.8GHz, 8GB RAM,
  Windows 7 SP1,
  Matlab R2017a/R2018a
  The JSR toolbox v1.2b, with and without Gurobi v8.0.1,
- Intel Core i7@2.5GHz, 16GB RAM,
  OSX 10.10.5 (Yosemite), macOS 10.14.1 (Mojave),
  Matlab R2017b,
  The JSR toolbox v1.2b, Gurobi v8.0.1.

The packages do not run on Matlab versions before and including R2015b.

2.3 `m`-package

This package implements functions which generalize Matlab functions to n-arrays for arbitrary $n \in \mathbb{N}_0$, and equips them with 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.

Dependencies/Collisions

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

Functions

So far the following functions have been implemented: `allm`, `anym`, `cart2sphm`, `convm`, `dec2basem`, `factorialm`, `ind2subm`, `isvectorm`, `kronm`, `lcmm`, `maxm`, `minm`, `nchoosekm`, `ndimsm`, `onesm`, `parsem`, `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 hyperspherical to Cartesian coordinates, assuming 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`[2].

`plotm` Unified interface for the visualization of various data types.

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

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

`padarraym` Works like `padarray` from the `Image processing toolbox`, but works also for cell arrays, etc..[3],

2.4 `sequence`-package

This package implements the vector space $\ell_0(\mathbb{Z}^s)$, $s \in \mathbb{N}$. The design-goal is, that `sequence`s behave 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.

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

Dependencies

The package depends on the `tmisc` and `m`-package.

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

Functions

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

2.5 `subdivision`-package

This package implements functions for the work with subdivision schemes. Most of them can be used as a black-box.

Dependencies

The package depends on the `m`-, `tmisc`- and `sequence`-package. The package furthermore depends on the Matlab `Parallel Toolbox` and `Symbolic Math Toolbox`. Furthmore it uses functions from the Matlab `Signal Processing Toolbox` but also runs without the latter.

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 of a multiple subdivision scheme.

`constructOmega` Constructs the set Ω_C of a multiple subdivision scheme [3].

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

`restrictmatrix` Restricts matrices to a subspace.

`transitionmatrix` Constructs transition matrices.

`num2ordering` Computes number expansions for multiple multivariate number systems.

`ordering2num` Computes numbers corresponding for multivariate 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 $M[0,1)^s \cap \mathbb{Z}^s$.

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

`constructVt` Constructs a basis for the space $\tilde{V}_k(\Omega)$ [3].

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

`daubechiesmask` Returns the mask coefficients for Daubechies' wavelet.

`findperiod` Searches for periodics in sequences.

`isodering` Determines if input is ordering.

`isS` Determines if input are subdivision operators.

`isT` Determines if input are transition matrices.

multiplyS Concatenates subdivision operators.

mask2symbol Computes the symbol of a mask.

normalizeS Normalizes the values of a mask.

ordering2vector Converts an ordering to 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 Tests heuristically if an attractor is a tile (for dimension 1 and 2).

vector2ordering Wrapper function for `findperiod`.

2.6 `tjsr`-package

This package implements functions to compute the JSR using the modified invariant-polytope algorithm.

Dependencies

The package depends on the `m`- and the `tmisc`-package and partly on the `subdivision`-package but also runs without the latter. The package furthermore depends on the `Gurobi`-Solver and the Matlab `Parallel Toolbox`.

If the `Gurobi`-solver is not installed, Matlab-functions will be used as a fall-back and the algorithm runs magnitudes slower. If the `Parallel Toolbox` is not installed, the algorithm will run single-threaded.

Important functions

findsmp[4] Searches for s.m.p.-candidates using various algorithms.

invariantsubspace[5] Searches for invariant subspaces of matrices using various algorithms.

setuptjsr Performs a self-test of the `tjsr`-package.

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

tjsr Computes the joint spectral radius.

tjsr_getpolytope Returns the constructed invariant polytope returned by `tjsr`.

preprocessmatrix Simplifies sets of matrices while preserving its JSR.

Remaining functions

binarymatrix Returns sets of binary matrices.

blockjsr Returns the JSR of block diagonal matrices, given the JSR of the blocks.

codecapacity Returns matrices whose JSR is related to their capacity, given forbidden differences.

computepolytopenorm Computes the Minkowski-norm.

daubechiesmatrix[6] Constructs matrices whose JSR is related to the Daubechies' wavelets regularity.

estimatepolytopenorm Estimates the Minkowski-norm.

estimatejsr Rough estimate of the JSR.

chooseval Selects highest values of a vector.

intersectinterval Intersects intervals.

leadingeigenvector Returns all leading eigenvectors of a matrix.

makeorderinggraph Constructs the graph corresponding to a partially ordered set.

[4] Copyright for algorithm `'genetic'` by [1]. Copyright for algorithm `'gripenberg'` by [6] under the 3-clause BSD License.
[5] Copyright for algorithm `'perm'` and `'basis'` by [6] under the 3-clause BSD License.
[6] Copyright for algorithm `'jung'` by [6] under the 3-clause BSD License. Copyright for algorithm `'gugl'` by Nicola Guglielmi.

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 Removes periodics and cycles vectors such that they have smallest lexicographic value.

removecombination Constructs a minimal set of cycles.

Functions taken from others

tavailable_memory[7] Returns the available memory.

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

tcellDivide[7] Divides matrices in a cell.

tgenNecklaces[7] Generation of all necklaces.

tgraphSCC[7] Finds the strongly connected components of graph.

tjointTriangul[7] Searches for invariant subspaces of matrices.

tjsr_zeroJsr[7] Decides if the JSR of a set of matrices is equal to zero.

tliftproduct[7] Computes all products of matrices of a given length.

tliftsemidefinite[7] Computes semi-definite liftings of matrices.

tpermtriangual[7] Searches for invariant subspaces 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.

2.7 `tmisc`-package

This package contains useful functions. Most other packages rely on this package.

Dependencies

The package depends on the `m`-package.

Important functions

findperiod Searches for periodics of digit sequences.

grCenter Finds the centre of a tree.

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

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

mixvector[10] Constructs all possible combinations of a set i.e. the Cartesian product.

normalize Normalizes matrices in various ways.

removezeros Deletes zeros in arrays in various ways.

repcell Repeats copies of an array.

rho[7] Computes the spectral radius of matrices.

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.

[7] Taken from `The JSR-toolbox` [6]. Published under the 3-clause BSD License.
[8] Uses code from [6]. Copyright: 3-clause BSD License.
[9] Copyright by Ondrej Sluciak, ondrej.sluciak@nt.tuwien.ac.at under the 2-clause BSD License.
[10] Uses code from Jos van der Geest, samelinoa@gmail.com, under the 2-clause BSD License.

tbuildProduct[11] Constructs the product of matrices corresponding to an ordering.

trho Computes the spectral radius of matrices.

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

vdisp[12] Compact (but sometimes ugly) display of (nested) objects.

vprintf Powerful version of **sprintf** with the additional specifier **%v**.

Remaining functions

cprintf[13] Displays styled formatted text in the command window.

flatten Converts nested cell arrays to flat cell arrays.

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.

mat Converts a vector to a matrix

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 Converts a flat cell array to a nested cell array.

vec Converts a matrix to a vector

[11] Uses code from [6]. Copyright: 3-clause BSD License.
[12] Uses code by Stefan, University of Copenhagen, under the 2-clause BSD License.
[13] Copyright by Yair Altman (2015) under the 2-clause BSD License.

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.

Syntax

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

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 non-periodic/periodic part of the $\texttt{i}^{th}$ row.

Output

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

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.

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 $\frac{1}{3}$ could 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 a finite set of *subdivision operators*. This is a cell array, each row describing one subdivision operator. Each row has the entries

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

The variables are:

mask a *dim*-array
Mask a of the subdivision operator.

Note that univariate subdivision schemes have column vectors as masks.

dilation M $dim \times dim$ matrix
Dilation matrix M of the subdivision operator.

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

name n string
Name of the subdivision operator.

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.

Syntax

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

Input

dim integer
Returns all subdivision operators encoded in the file `getS` in the section `%UNNAMED SUBDIVISION OPERATORS`.
E.g.: `getS(2);`

cellarray cell-array `{[a],M,[D],[n]}` where `a`, `M`, `D`, `n` are 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, $D := M\mathbb{Z}^{dim} \cap \mathbb{Z}^{dim}$.
If `a` is not given $a := \chi_D$, where χ is the characteristic function.
If `n` is not given $n :=$ `'unnamed'`.

E.g.: `getS({[0.5 1 0.5]',2});`

name string
Returns the subdivision operator with name `name` encoded in the file `getS` in the sub-routine `getS_named`.

E.g.: `getS('1_Hassan_Dodgson_3point');`

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.

E.g.: `getS('a',[.25 .5 .25;.5 1 .5;.25 .5 .25],'M',[2 0; 0 2],'n','tensorlinear');`

Options

'bigcheck' Enables some data-integrity checks.

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

'help' The strings of the named subdivision operators (i.e. the allowed strings for `name`) are printed (among some other strings)[2].

[2] This is actually an option for the function `parsem`.

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

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

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

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

Output

`S` cell array of subdivision operator(s)

Note

- All subdivision operators in a cell array should be for the same dimension.
- Matlab versions prior to R2018a cannot display subdivision operators for dimension 1, and throw an error if attempted to do so.

Subset of the possible values for `name`

Univariate schemes

`'1_all'` All named subdivision schemes of dimension 1.

`'1_rand'` 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 scheme $a = \frac{1}{256}\left[3 \quad 0 \quad -25 \quad 0 \quad 150 \quad 256 \quad 150 \quad 0 \quad -25 \quad 0 \quad 3\right]$, $M = 2$.

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

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

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

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

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

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

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

Bivariate schemes

`'2_all'` All named subdivision schemes of dimension 2.

`'2_rand'` A random subdivision scheme of dimension 2.

`'2_butterfly'` The butterfly scheme.

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

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

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

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

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

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

Trivariate schemes

`'3_rand'` Returns a random subdivision scheme of dimension 3.

Quadrovariate schemes

`'4_rand'` Returns a random subdivision scheme of dimension 4.

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

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.

`getS('a',[.25 .5 .25;.5 1 .5;.25 .5 .25],'M',[2 0; 0 2],'n','tensorlinear');` The first bivariate tensor-product B-spline.

3.3 `blf`

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

Syntax

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

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), mandatory
The parameter `S` is passed to `getS` and the returned value is used.

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.

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

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

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

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

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

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

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

Output

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

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. I.e. for given `S` and `oo`, the set defined by

$$\texttt{M}_{\texttt{oo}_1}^{-1}\texttt{D}_{\texttt{oo}_1} + \texttt{M}_{\texttt{oo}_1}^{-1}\texttt{M}_{\texttt{oo}_2}^{-1}\texttt{D}_{\texttt{oo}_2} + \texttt{M}_{\texttt{oo}_1}^{-1}\texttt{M}_{\texttt{oo}_2}^{-1}\texttt{M}_{\texttt{oo}_3}^{-1}\texttt{D}_{\texttt{oo}_3} + \cdots .$$

Syntax

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

Input

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

`S` subdivision operators, mandatory
The subdivision operators

Options

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

`'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.

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

`'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.

`'supertile',n` integer
Computes the supertile $K = \cup_j (M_j^{-1}K + D_j)$ instead of the attractor.

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

Output

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

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

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_C as described in [3].

Syntax

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

Input

`S` cell array of subdivision operators, mandatory

Options

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

`'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.

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

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

Output

`Om` matrix
The set Ω_C.

Example Usage

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

Basic implementation

```
function [ Om ] = constructOmega( S, Om )
% S: cell array of subdivision schemes. Each row consists of a, M and D.
% Om: (Optional) the starting set
% Ex: a=1/3*[1 2 3 2 1]; M1=[2 -1;1 -2]; M2=[1 1;1 -2]; D=[0 1 2;0 0 0];
%     constructOmega({a, M1, D; a, M2, D})
    a=S(:,1); M=S(:,2); D=S(:,3);                  %extract the sets a,M and D
    J=numel(a);                                    %number of subdivision operators
    dim=size(M{1},1);                              %the dimension
    if(nargin==1); Om=zeros(dim,1); end            %if Omega is not given, set it to zero
    while(true)
        sizebefore=size(Om,2);                     %used to check if elements were added
        for j=1:J                                  %iterate through all subdiv. operators
            OmN=M{j}\setplus(supp(a{j},dim),Om,-D{j}); %compute new possible entries
            OmN=round(OmN(:,sum(abs(OmN-round(OmN)),1)<.5/abs(det(M{j})))); %round to integers
            Om=unique([Om OmN]','rows')';          %remove duplicates
        end
        if(size(Om,2)==sizebefore); break; end     %if no elements were added, terminate
    end

function [ X ] = setplus( varargin )
% setplus(A,B) = { x=a+b : a in A, b in B}, operates column wise
% Ex: setplus([1 2; 1 0],[0 -1;-1 -1]); %Output: [0 1 1 2;0 -1 0 -1]
    sze=size(varargin,2);                  %number of sets
    X=varargin{sze};                       %the output set
    for i=sze-1:-1:1                       %iterate through all sets
        A=varargin{i};                     %the set to be added
        X=repmat(A,1,size(X,2))+reshape(repmat(X,size(A,2),1),size(A,1),[]); %add the set
        X=unique(X','rows')';              %remove duplicates
    end

function [ L ] = supp( a, dim )
% returns the support of an array. First entry is supposed to have index (0,0,...,0)
% Ex: supp([1 1;0 1],2) %Output: [0 0 1;0 1 1];
    L=zeros(dim,nnz(a));                   %output variable
    CO=cell(1,dim);                        %dummy-variable to do calculation with indices
    j=1;                                   %index-variable for the columns of D
    for i=1:numel(a)                       %iterate through all elements of the masks
        if(a(i)~=0)                        %if the element is nonzero, save the indices
            [CO{:}]=ind2sub(size(a),i);    %get the indices
            L(:,j)=[CO{:}]'-1;             %add converted cell to vector
            j=j+1;                         %increase counter
        end
    end
```

3.6 constructV

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

Syntax

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

Input

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

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

Options

`'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)`.

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

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 $\dim V_k > 0$.

Note

The functions `constructVt` and `constructU` construct the spaces $\tilde{V}_k(\Omega)$ and $U_k(S)$ 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 informations.

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, the function computes

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

Syntax

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

Input

`T` square matrix or cell array of square matrices, mandatory
The matrices which shall be restricted to the subspace `A`.

`A` rectangular matrix, mandatory
Matrix defining the subspace using column vectors.

Options

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

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

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

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 $dim_A \times dim_A$ matrix (or cell array of)
Restriction of T to A

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

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

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

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

Example Usage

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

3.8 transitionmatrix

Constructs transition matrices $T_{d,\Omega} = (\mathtt{a}(\alpha - \mathtt{M}\beta + d))_{\alpha,\beta\in\Omega}$ where $\mathtt{a} \in \mathtt{S}$ are subdivision masks, $M \in \mathtt{S}$ are dilation matrices, $d \in \mathtt{D} \in \mathtt{S}$, $D \simeq \mathbb{Z}^s/\mathtt{M}\mathbb{Z}^s$, are digit sets and $\Omega \subseteq \mathbb{Z}^s$ is an invariant set for these matrices [3].

Syntax

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

Input

S cell array of subdivision operators, mandatory

Options

'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.

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

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

'onlyindices' Function 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).

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

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

Output

T cell array of matrices
The transition matrices.

Om The set Ω used to compute the transition matrices.

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 of 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 $\tilde{V}_0$. To apply the joint spectral radius approach to compute the regularity of subdivision-schemes, the dimension of $V_0(\Omega)$ and $\tilde{V}_0(\Omega)$ 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=constructVt(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, which are: the *modified Gripenberg algorithm* as described the in the accompanying 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 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. The implementation of the Gripenberg algorithm and of the genetic algorithm only return *one* s.m.p.-candidate. Thus they are not suitable for finding s.m.p.-candidates as needed for the modified invariant polytope algorithm.

Syntax

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

Input

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

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

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

`'gripenberg'` Gripenbergs algorithm [4].

`'bruteforce'` Brute force algorithm.

`'genetic'` Genetic algorithm [1].

Options

Options available for all algorithms

`'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

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

Options for algorithm `'modifiedgripenberg'`

`'delta',val` double, default: 0.99, (experimental option, 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.

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

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

`'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 the Gripenberg algorithm.

`'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..

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

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

`'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.

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

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

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

Options for algorithm `'genetic'`

`'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.

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

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 ρ_c/`delta`, where ρ_c is the current upper bound for the JSR, are thrown away.

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

`'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.

'minJSR',val double, default: 0
Candidates must have at least **minJSR** spectral radius.

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

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.

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' )
```

Basic Implementation

```
function [c] = gripenberg_modified(M,N,D)
%Tries to find smp-candidates in a fast way.
%Ex: gripenberg_modified({[2 1; 0 -2],[2 1; -1 -2]}, 4, 10)
J = length(M);  %number of matrices
o = 1:J;        %the orderings of the products to be checked
c = {};         %list of candidates
r = 0;          %lower bound for JSR
for d = 1:D                     %do D iterations
    NR = zeros(2,size(o,2));    %norm and rho of candidates
    for i = 1:size(o,2)         %can be parallelised using parfor!
        P = buildProduct(M,o(:,i));             %construct matrices
        NR(:,i) = [norm(P); max(abs(eig(P)))];  %compute norm and rho
    end
    NR = NR.^(1/d);             %normalize norm and rho
    if r < max(NR(2,:))         %test if new bound was found
        c = {};                 %delete candidates
        r = max(NR(2,:));       %update lower bound for JSR
```

```
    end
    c = [c num2cell(o(:,NR(2,:) >= r),1)]; %add candidates to c
    idx = NR(1,:) < r;          %remove everything with norm less than JSR
    NR(:,idx) = [];
    o(:,idx) = [];
    [NR,idx] = sortrows(NR');   %sort correspdonding to norm
    NR = NR.'; idx = idx.'; nNR = size(NR,2);
    if nNR > 2*N                %keep highest and lowest norms
        o = o(:,[idx(1:N) idx(nNR-N+1:nNR)]);
    else                        %keep everything if N is too big
        o = o(:,idx);
    end
    o = [repmat(o,[1 J]);       %make new orderings of products
         reshape(repmat(1:J,[size(o,2) 1]),1,[])];
end

function M = buildProduct(A,prod)
% Constructs the product of matrices of A corresponding to prod.
M = eye(size(A{1},1));
for t = 1:length(prod); M = A{prod(t)}*M; end
```

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`.

Syntax

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

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.

Options

'bool' flag
Returns boolean matrices.

'int' flag
Returns integer matrices.

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

'pos' flag
Returns matrices with positive entries.

`'rho'` flag
Returns matrices with spectral radius 1.

`'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

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

Output

`val` cell array of square matrices
The returned matrices.

Note

Most options preserve do not preserve certain properties of the matrices, e.g. the entries in the matrix returned by `tgallery('rand_gauss',10,1,'pos')` are not normally distributed anymore.

Possible values for `what` and their mandatory arguments

This section lists a subset of the 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_bool',dim,N` Random matrices with values 0, 1.

`'rand_doublestochastic',dim,N` Random double-stochastic matrices.

`'rand_doublestochastic_neg',dim,N` Random double-stochastic matrices with pos. and neg. values.

`'rand_corr_1',dim,N` Random correlation matrices with pos. entries.

`'rand_corr_0',dim,N` Random correlation matrices with non-neg. entries.

`'rand',dim,N` Random matrices with equally distributed values in $[0,\ 1]$.

`'rand_gauss',dim,N` Random matrices with normally distributed values.

`'rand_hess',dim,N` Random orthogonal upper Hessenberg matrices.

`'rand_neg',dim,N` Random matrices with equally distributed values in $[-1,\ 1]$.

`'rand_normal',dim,N` Random matrices with normally distributed values.

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

`'rand_stochastic',dim,N` Random column-stochastic matrices.

`'rand_stochastic_neg',dim,N` Random column-stochastic matrices with positive and negative values.

`'rand_unitary',dim,N` Random unitary matrices.

`'rand_zero',dim,N` Random matrices with spectral radius 0.

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

Matrices from applications

`'binary',dim,N,k` Matrices whose linear entries equals the number `k` in base 2. E.g.:

$$\texttt{tgallery('binary',2,2,19)} = \left\{ \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \right\},$$

since $19_{[10]} = 00010011_{[2]}$.

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 may return the empty set.

`'cex'` Pair of 2x2 matrices which has no s.m.p. [10], 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 the source-code of `codecapacity` for more information. E.g.: `tgallery('code',{[1 1 0 1]})`.

`'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_p45'` Matrices $\{\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}\}$ from [4, p. 45].

`'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].

`'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 the Pascal rhombus [5, p. 35].

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

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

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

`'prot2016'` Matrices for the subdivision scheme [5, p. 33, p. 35, p. 50].

`'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_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 approximately $e \cdot x$.

Example Usage

```
tgallery('rand_gauss',5,2,100,'rho')
tgallery('mejstrik_119')
```

4.3 invariantsubspace

Searches for invariant subspaces of matrices $M \in$ `M`, i.e. a change of basis B such that all matrices $B^{-1}MB$, $M \in$ `M` have block-triangular form. The function uses three different algorithms: `permTriangul`, `jointTriangul` from [6] and an implementation of [2]. The returned matrices still may have invariant subspaces which can or cannot be found using this function.

Syntax

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

Input

`M` cell array of matrices, mandatory
The matrices.

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

`'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'` Tries to find subspaces generated by differences of basic limit functions as occurring in subdivision theory [2]. The algorithm first computes numerically, then symbolically, then using `vpa`.

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

Options

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

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.

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.

Syntax

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

Input

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

Important options

This is a list of the most important options (which should be sufficient for the standard-user). In a later section all available options are listed.

`'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).

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

`'invariantsubspace',string` string, default: `'auto'`
`string` is one of the following: `'none'`, `'perm'`, `'basis'`, `'trans'`, `'auto'`. See the documentation of `invariantsubspace` for more information.

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

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

`'nobalancing'` default: false
Disables balancing.

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

`'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)

`'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 fail in some cases.

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

Note

- 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 beforehand.
- 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.

Output

Screen Output

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

Output in front of the progress bar

- `Time: x/y` Time needed for the 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 the polytope
- `.` Vertex is machine-epsilon-near to the polytope
- `,` Vertex is 1000*machine-epsilon-near to the polytope
- `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 trees.

`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 algorithm 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`.

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` which returns example matrices. Some options used for these examples are described in detail in Section 4.4.

- 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 needs 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..

    Note that (*i*) *orderings of extra-vertices MUST be empty*, (*ii*) *orderings MUST be given as column vectors*, and (*iii*) 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)` Just starting the algorithm 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 to balance the trees. This option is useful when one wants to prove the invariance of the invariant polytope by hand.

  `tjsr(D7,'nobalancing')` This command disables the balancing (and the algorithm applied to this example will not terminate). Using 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 to the third cyclic-tree.

  `tjsr(D7,'autoextravertex',0)` This command disables the automatic 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_gauss',6,2,'rho','seed',100)`.

  `tjsr(T,'plot','norm')` Plots the computed norms of vertices and colours 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(T,'plot','tree')` 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 separated with an underscore `_`.

  `tjsr(T,'plot','info_normest_norm_rho','fastnorm',0)` Plots the estimated norms against the real norms against the spectral radii.

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

All options

Pre- and postprocessing options

These options control 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.

`'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.

`'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 subspaces (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*.

Preworker options

These options control the search for candidates, nearly-candidates and extra-vertices, etc..

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

`'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]`.

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

`0` Complex eigenvectors are kept.

`1` Complex eigenvectors are removed, whenever there is at least one real eigenvector corresponding to the same product.

`2` (default) Complex eigenvectors are removed, whenever 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.

`'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.

`'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. See the Example Usage-Section for more information.
E.g.: `'extravertex',{[0.1 0 0]', [0 0.1 0]'}`

`'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.

`'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' Tries to find subspaces generated by differences of basic limit functions as occurring in subdivision theory [2]. The algorithm first computes numerically, then symbolically, then using vpa.

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

'JSR',val 1x2-vector, default: empty, deprecated option
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.

'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.

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

'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.

'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).

'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.

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

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

'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. See the Example Usage-Section for more information.

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.

'noclassify' default: false
If false, all s.m.p. candidates (and their cyclic permutations) are examined whether they have equal leading 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.

Worker options

These options control 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.

`'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 is less then `epslinprog`, in particular `epspolytope` can be less then zero. See the Example Usage-Section for more information.

`'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` (not recommended) Check whether points are inside or outside. The behaviour of this option may be changed in a future release.

`'naturalselection',int` integer, default: depends on the number of available threads
Minimum number of vertices whose norms are computed in each level. The maximum number of vertices computed in each level is roughly the product of `naturalselection` and the number of available workers in the Matlab pool.
This option may be renamed in a future release.

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

`inf or -inf` (default: `+inf`) Use three times norm-estimate and 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` (not recommended) 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.

`'simplepolytope',val` integer, default: 10^{-8}
Vertices with norm less than $1 +$ `simplepolytope` may not be used in the norm computation. `simplepolytope` should be greater than `epslinprog`.

`'testoldvertex',val` integer, default: 1
Whether old vertices shall be estimated again 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

Termination options

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

`'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 more than `maxstepnumber` vertices are tested.

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

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

`'testeigenplane',val` double, default: $-\infty$ (i.e. this option is disabled)
Tests whether the distance of a vertex to the supporting eigenplanes defined by `v0` and `v0s` is more then approximately $1 -$ `testeigenplane`. If a vertex lies outside, the candidates are not s.m.ps. Positive values lead to 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 lead to false-positives. If `testspectralradius` $= 1$, the algorithm changes the value to -10^{-10}. This behaviour may be changed in a future release.

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

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

Output options

These options control the output during the computation, and partly also the return-values.

`'diary'` integer
Starts the Matlab diary and may change the default value for `save`.

`'plot',string` string, default: empty
`string` is one of the following: `'norm'`, `'polytope'`, `'L'` or an identifier beginning with `'info_'`.

`'norm'` Plots intermediate norms

`'polytope'` Plots the constructed polytope

`'L'` Plots the number of vertices left to compute (at the moment)

`'info_...'` A string beginning with `'info_'` and an arbitrary number of strings, which are names of fields in the output-struct `info`, separated by underscores `'_'`.
Fields in the sub-struct `info.cyclictree` do not need the prefix `'cyclictree.'`. All addressed fields are plotted when possible. E.g.: `'info_norm'`, `'info_cyclictree.norm'`, `'info_norm_normest_L'`.

See the Example Usage-Section and the source-code of `tjsr_plotoutput` for more information.

`'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 termination.

`2` Save output after each iteration.

`3` Save output after each iteration in a new file.

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

Debug options

These options are merely for testing the algorithm and should not be changed by the standard-user.

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

'memory' If set, the 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 a key-press after balancing.

'rholeqval',val If set, matrix products whose spectral radius is greater than `rholeqval` 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.

Output: 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`. See Pre- and postprocessing options for more information.

info.M_original cell array of matrices
Preprocessed 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 happened.

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.numextravertex integer
Number of extra-vertices.

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

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

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

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 main types of 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. Note that this sequence is not monotonic in general.

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 mean successful termination, all other values mean 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` < 1.
`0` Input error.
`inf` Unknown 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 occurred. The algorithm usually does not recover from that error and aborts. If that error happens, it is recommended to start the algorithm for each invariant subspace, which can be using with the function `invariantsubspace`.
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 the output text (and even more).

`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` contain aggregated informations from `info.block{:}`.

4.5 `tjsr_getpolytope`

Returns the vertices of the invariant polytope.

Syntax

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

Input

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

Output

`VV` matrix of column vectors
The invariant polytope.

Note

The function basically does the following:

```
VV=[info.cyclictree.V{:}]; %get all vertices
idx=[info.cyclictree.norm{:}]>1-info.opt.epspolytope; %choose those which are outside
VV=VV(:,idx);
```

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

Syntax

Simplifies sets of matrices, while preserving their joint spectral radii (when called with default-values).

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

Input

`M` cell-array of matrices
The input matrices

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 switched 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}` are 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.

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.

Example Usage

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

5 Copyright

5.1 Papers to cite

If you use the `tjsr`-package, please cite

- N. Guglielmi, V. Yu. Protasov, "Exact computation of joint spectral characteristics of linear operators", Comput. Math. 13 (2013)
- N. Guglielmi, V. Yu. Protasov, *Invariant polytopes of linear operators with applications to regularity of wavelets and of subdivisions*, SIAM J. Matrix Anal. Appl. 37 (2016)
- T. Mejstrik, *Improved invariant polytope algorithm and applications*, ArXiv (2018)

If you use the `subdivision`-package, please cite

- M. Charina, T Mejstrik, *Multiple multivariate subdivision schemes: Matrix and operator approaches*, J. Comp. Appl. Math. (2018)
- T. Mejstrik, *Improved invariant polytope algorithm and applications*, ArXiv (2018)

5.2 Cover-image

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

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] 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).