

vm – Vectorized Movie class

The vectorized movie encapsulates a 3D dataset, provides read/write and display methods, and streamlines common movie operations. This class (in particular the `moviesc` method) is compatible with R2014b or later.

`+` `-` `*` `/` `^` `.*` `./` `.^` `'` `.'` Math and common functions are overloaded in the spirit of using the movie object as a regular variable. The design rule is that matrix operations are applied to the `[nrows*ncols nframes]`-dimension movie matrix, while scalar notation operations are `bsxfun`-ed (i.e. operand dimensions can either match or be 1) and applied to the `[nrows ncols nframes]`-dimension 3D movie. Matrix operations between two vectorized movies are customarily defined for particular cases.

Array indexing for access and assignment is overloaded to provide access to movie subsets. This only works in the first level of indexing, and all other indexing behavior is meant to be preserved. a movie object can be indexed with 1, 2, or 3 subscripts. 1 subscript: indexes frames of the movie like `vm(mov.data(:,frames))`. 2 subscripts: indexes the vectorized matrix data in (rowcol, frames), like `mov.tovec.data(rowcol,frames)`. 3 subscripts: indexes the movie data in (columns, rows, frames).

Tired of the tedious handling of movies dimensions for linear algebra and image processing? Not anymore -- this class encapsulates a movie and its dimensions in order to perform routine tasks more intuitively. This class should cover all the tasks that one wants to apply to a movie. As an evolving project, this class needs to be expanded for common needs of different users (if any), and bugs need to be fixed. Methods will change in future versions. Please contact me if you have suggestions for improvement.

Copyright 2016-2017 Vicente Parot

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

How to use it:

```
% create object from existing matrix, display, save movie for slides, transpose and
save binary file to open in ImageJ
mov = vm(my_3d_matrix)
mov.moviesc
mov.saveavi('A:\updates\for_presentation.avi')
mov.transpose.savebin('A:\analysis\processed_movie.bin')

%% hadamard optical sectioning reconstruction
cal = vm('A:\calibration\') % assumes directory includes a binary file and format of
the experimental parameters file for image dimensions
raw = vm('A:\experiment\Sq_camera.bin',512,512) % or, specify them
spots = raw*patterns
spots.moviesc
hadamard_img = cal*spots;
reference_img = raw.mean;

%% note that matlab accepts multiple notations. if you prefer, you can choose to use
the good old syntax and avoid dots ...
%% alternate notation for commands above
moviesc(spots)
reference_img = mean(raw);
```

More examples:

```
% normalize movie object mov by its time average
mov = mov./mov.mean
```

If mov was instead a 3D matrix, the following command would have worked:

```
% normalize 3D matrix movie mov by its time average
mov = bsxfun(@rdivide,mov,mean(mov,3));
```

No clicky functions are part of the vm class. Instead, external functions should be used:

```
clicky_faster
apply_clicky_faster
clicky_rects
nested_clicky_rects
```

To expand the class, please follow the same approach to make existing functions vm-compatible: just add three lines of code at the top like in clicky_rects.m for example.

If instead you make new functions that will be useful for anyone processing any movie, please feel encouraged to add them into the vm class.

Here is a short list of existing vm functions you may want to know about:

`blur(mov,sigma)` -- shorthand for `imfilter` with gaussian kernel
`pblc` -- photobleach correction dividing by fitted exponential
`montage` -- similar to `imageJ`
`correct_blank_marker` -- remove artifact from hamamatsu raw files
`imsz(mov)` -- image size of frames
`crop(mov,poly)` -- subregion of movie containing polygon
`trim(mov)` -- remove trailing constant borders
`blnfun(mov,@fun,n)` -- apply `fun` to blocks of `n` frames
`evnfun(mov,@fun,n)` -- apply `fun` to groups of every `n` frames
`moviesc` -- displays a movie with scaled colormap in each frame
`moviefixsc` -- displays a movie with fixed colormap over frames
`savebin`
`saveavi`
`savetiffstack` -- `saveastiff` wrapper

A complete list with brief explanations is found in the definition file `vm.m`