**SF\_Python**

Appunti di Python

Sergio Frasca

Ott 2022

Sommario

[Zen of Python 5](#_Toc120572191)

[Sistema 6](#_Toc120572192)

[Comandi 6](#_Toc120572193)

[Linguaggio 7](#_Toc120572194)

[Assegnazione 8](#_Toc120572195)

[Mutabili e immutabili 11](#_Toc120572196)

[Gestione 12](#_Toc120572197)

[Understanding Conda and Pip 12](#_Toc120572198)

[Path 14](#_Toc120572199)

[Exec & C 15](#_Toc120572200)

[**Reloading modules for >=Python3.4 and above** 15](#_Toc120572201)

[**Turn double square brackets to single** 15](#_Toc120572202)

[Dictionaries 16](#_Toc120572203)

[Functions 18](#_Toc120572204)

[Classi 20](#_Toc120572205)

[Kwargs in \_\_init\_\_ 22](#_Toc120572206)

[Packages 28](#_Toc120572207)

[External packages 28](#_Toc120572208)

[numpy 28](#_Toc120572209)

[scipy 29](#_Toc120572210)

[random 29](#_Toc120572211)

[copy 29](#_Toc120572212)

[matplotlib 29](#_Toc120572213)

[astropy 30](#_Toc120572214)

[Python time Module 30](#_Toc120572215)

[PyInstaller 30](#_Toc120572216)

[**Matlab** 31](#_Toc120572217)

[mat4py · PyPI 40](#_Toc120572218)

[**How to load Matlab .mat files in Python** 40](#_Toc120572219)

[**The data** 40](#_Toc120572220)

[**Loading .mat files** 41](#_Toc120572221)

[**Formatting the data** 41](#_Toc120572222)

[Matlab 43](#_Toc120572223)

[MATLAB commands in numerical Python (NumPy) 43](#_Toc120572224)

[1 Help 43](#_Toc120572225)

[3 Vectors 48](#_Toc120572226)

[6 File input and output 55](#_Toc120572227)

[7.1.1 Axes and titles 56](#_Toc120572228)

[7.1.4 Functions 57](#_Toc120572229)

[7.4.1 Contour and image plots 59](#_Toc120572230)

[7.4.2 Perspective plots of surfaces over the x-y plane 60](#_Toc120572231)

[10 Programming 63](#_Toc120572232)

[File IO (**scipy.io**) 65](#_Toc120572233)

[MATLAB files 65](#_Toc120572234)

[The basic functions 65](#_Toc120572235)

[How do I start? 65](#_Toc120572236)

[Altro 66](#_Toc120572237)

[**Installation:** 67](#_Toc120572238)

[**How to use this lib:** 67](#_Toc120572239)

[**To load the Matlab data file:** 68](#_Toc120572240)

[**Flag descriptions** 68](#_Toc120572241)

[**To Upload and Read mat files in python** 68](#_Toc120572242)

[Varie 70](#_Toc120572243)

[**Time** 71](#_Toc120572244)

[time — Time access and conversions 71](#_Toc120572245)

[Functions 72](#_Toc120572246)

[Astropy 83](#_Toc120572247)

# Zen of Python

by Tim Peters

Please, **import this**

* Beautiful is better than ugly.
* Explicit is better than implicit.
* Simple is better than complex.
* Complex is better than complicated.
* Flat is better than nested.
* Sparse is better than dense.
* Readability counts.
* Special cases aren't special enough to break the rules.
* Although practicality beats purity.
* Errors should never pass silently.
* Unless explicitly silenced.
* In the face of ambiguity, refuse the temptation to guess.
* There should be one– and preferably only one –obvious way to do it.[[a]](https://en.wikipedia.org/wiki/Zen_of_Python#cite_note-7)
* Although that way may not be obvious at first unless you're Dutch.
* Now is better than never.
* Although never is often better than right now.[[b]](https://en.wikipedia.org/wiki/Zen_of_Python#cite_note-8)
* If the implementation is hard to explain, it's a bad idea.
* If the implementation is easy to explain, it may be a good idea.
* Namespaces are one honking great idea – let's do more of those!

# Sistema

## Comandi

Immagine che contiene testo

Descrizione generata automaticamente

dir()

id()

isinstance(a,type)

# Linguaggio

## Assegnazione

Immagine che contiene testo

Descrizione generata automaticamente

## Mutabili e immutabili

# Gestione

Understanding Conda and Pip

**NOV 28, 2018**

jhelmus@anaconda.com

Conda and pip are often considered as being nearly identical. Although some of the functionality of these two tools overlap, they were designed and should be used for different purposes. [Pip](https://pip.pypa.io/en/stable/) is the Python Packaging Authority’s recommended tool for installing packages from the [Python Package Index](https://pypi.org/), PyPI. Pip installs Python software packaged as wheels or source distributions. The latter may require that the system have compatible compilers, and possibly libraries, installed before invoking pip to succeed.

[Conda](https://conda.io/docs/) is a cross platform package and environment manager that installs and manages conda packages from the [Anaconda repository](https://repo.anaconda.com/) as well as from the [Anaconda Cloud](https://anaconda.org/). Conda packages are binaries. There is never a need to have compilers available to install them. Additionally conda packages are not limited to Python software. They may also contain C or C++ libraries, R packages or any other software.

This highlights a key difference between conda and pip. Pip installs Python packages whereas conda installs packages which may contain software written in any language. For example, before using pip, a Python interpreter must be installed via a system package manager or by downloading and running an installer. Conda on the other hand can install Python packages as well as the Python interpreter directly.

Another key difference between the two tools is that conda has the ability to create isolated environments that can contain different versions of Python and/or the packages installed in them. This can be extremely useful when working with data science tools as different tools may contain conflicting requirements which could prevent them all being installed into a single environment. Pip has no built in support for environments but rather depends on other tools like [virtualenv](https://virtualenv.pypa.io/en/latest/) or [venv](https://docs.python.org/3/library/venv.html) to create isolated environments. Tools such as [pipenv](https://pipenv.readthedocs.io/en/latest/), [poetry](https://poetry.eustace.io/), and [hatch](https://github.com/ofek/hatch) wrap pip and virtualenv to provide a unified method for working with these environments.

Pip and conda also differ in how dependency relationships within an environment are fulfilled. When installing packages, pip installs dependencies in a recursive, serial loop. No effort is made to ensure that the dependencies of all packages are fulfilled simultaneously. This can lead to environments that are broken in subtle ways, if packages installed earlier in the order have incompatible dependency versions relative to packages installed later in the order. In contrast, conda uses a satisfiability (SAT) solver to verify that all requirements of all packages installed in an environment are met. This check can take extra time but helps prevent the creation of broken environments. As long as package metadata about dependencies is correct, conda will predictably produce working environments.

Given the similarities between conda and pip, it is not surprising that some try to combine these tools to create data science environments. A major reason for combining pip with conda is when one or more packages are only available to install via pip. Over 1,500 packages are available in the Anaconda repository, including the most popular data science, machine learning, and AI frameworks. These, along with thousands of additional packages available on Anaconda cloud from channeling including [conda-forge](https://conda-forge.org/) and [bioconda](https://bioconda.github.io/), can be installed using conda. Despite this large collection of packages, it is still small compared to the over 150,000 packages available on PyPI. Occasionally a package is needed which is not available as a conda package but is available on PyPI and can be installed with pip. In these cases, it makes sense to try to use both conda and pip.

Comparison of conda and pip

|  | **conda** | **pip** |
| --- | --- | --- |
| manages | binaries | wheel or source |
| can require compilers | no | yes |
| package types | any | Python-only |
| create environment | yes, built-in | no, requires virtualenv or venv |
| dependency checks | yes | no |
| package sources | Anaconda repo and cloud | PyPI |

## Path

Immagine che contiene testo

Descrizione generata automaticamente

## Exec & C

**exec(open('script1.py').read())**

**Reloading modules for >=Python3.4 and above**

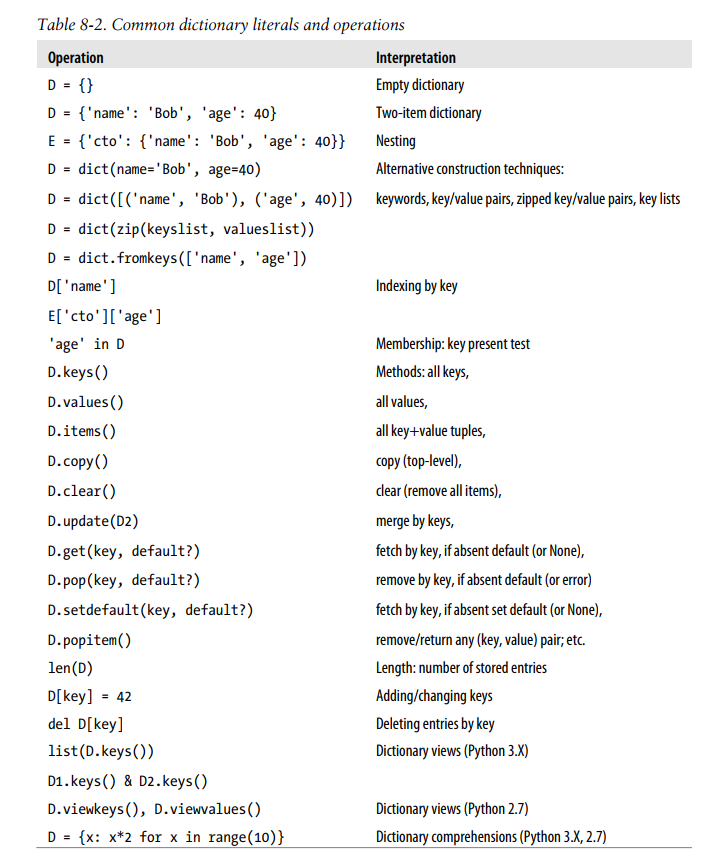
import importlib

importlib.reload(*module*)

## **Turn double square brackets to single**

If they are lists, as your example suggests, you can index the first element: [[0, 3.49, 0, 4.55]][0] -> [0, 3.49, 0, 4.55]

# Dictionaries



# Functions

**Argument Matching Syntax**Table 18-1 summarizes the syntax that invokes the special argument-matching modes.  
*Table 18-1. Function argument-matching forms*

Immagine che contiene testo

Descrizione generata automaticamente

The \*\* feature is similar, but it only works for keyword arguments—it collects them  
into a new dictionary, which can then be processed with normal dictionary tools. In a  
sense, the \*\* form allows you to convert from keywords to dictionaries, which you can  
then step through with keys calls, dictionary iterators, and the like (this is roughly what  
the dict call does when passed keywords, but it returns the new dictionary):  
>>> def f(\*\*args): print(args)>>> f(){}  
>>> f(a=1, b=2){'a': 1, 'b': 2}

# Classi

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

## Kwargs in \_\_init\_\_

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

# Packages

In addition to a module name, an import can name a directory path. A directory of Python code is said to be a *package*, so such imports are known as *package imports*. In  
effect, a package import turns a directory on your computer into another Python namespace, with attributes corresponding to the subdirectories and module files that the  
directory contains.

## External packages

### numpy

['ALLOW\_THREADS', 'AxisError', 'BUFSIZE', 'CLIP', 'ComplexWarning', 'DataSource', 'ERR\_CALL', 'ERR\_DEFAULT', 'ERR\_IGNORE', 'ERR\_LOG', 'ERR\_PRINT', 'ERR\_RAISE', 'ERR\_WARN', 'FLOATING\_POINT\_SUPPORT', 'FPE\_DIVIDEBYZERO', 'FPE\_INVALID', 'FPE\_OVERFLOW', 'FPE\_UNDERFLOW', 'False\_', 'Inf', 'Infinity', 'MAXDIMS', 'MAY\_SHARE\_BOUNDS', 'MAY\_SHARE\_EXACT', 'ModuleDeprecationWarning', 'NAN', 'NINF', 'NZERO', 'NaN', 'PINF', 'PZERO', 'RAISE', 'RankWarning', 'SHIFT\_DIVIDEBYZERO', 'SHIFT\_INVALID', 'SHIFT\_OVERFLOW', 'SHIFT\_UNDERFLOW', 'ScalarType', 'Tester', 'TooHardError', 'True\_', 'UFUNC\_BUFSIZE\_DEFAULT', 'UFUNC\_PYVALS\_NAME', 'VisibleDeprecationWarning', 'WRAP', '\_CopyMode', '\_NoValue', '\_UFUNC\_API', '\_\_NUMPY\_SETUP\_\_', '\_\_all\_\_', '\_\_builtins\_\_', '\_\_cached\_\_', '\_\_config\_\_', '\_\_deprecated\_attrs\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_expired\_functions\_\_', '\_\_file\_\_', '\_\_getattr\_\_', '\_\_git\_version\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_path\_\_', '\_\_spec\_\_', '\_\_version\_\_', '\_add\_newdoc\_ufunc', '\_distributor\_init', '\_financial\_names', '\_from\_dlpack', '\_globals', '\_mat', '\_pytesttester', '\_version', 'abs', 'absolute', 'add', 'add\_docstring', 'add\_newdoc', 'add\_newdoc\_ufunc', 'alen', 'all', 'allclose', 'alltrue', 'amax', 'amin', 'angle', 'any', 'append', 'apply\_along\_axis', 'apply\_over\_axes', 'arange', 'arccos', 'arccosh', 'arcsin', 'arcsinh', 'arctan', 'arctan2', 'arctanh', 'argmax', 'argmin', 'argpartition', 'argsort', 'argwhere', 'around', 'array', 'array2string', 'array\_equal', 'array\_equiv', 'array\_repr', 'array\_split', 'array\_str', 'asanyarray', 'asarray', 'asarray\_chkfinite', 'ascontiguousarray', 'asfarray', 'asfortranarray', 'asmatrix', 'asscalar', 'atleast\_1d', 'atleast\_2d', 'atleast\_3d', 'average', 'bartlett', 'base\_repr', 'binary\_repr', 'bincount', 'bitwise\_and', 'bitwise\_not', 'bitwise\_or', 'bitwise\_xor', 'blackman', 'block', 'bmat', 'bool8', 'bool\_', 'broadcast', 'broadcast\_arrays', 'broadcast\_shapes', 'broadcast\_to', 'busday\_count', 'busday\_offset', 'busdaycalendar', 'byte', 'byte\_bounds', 'bytes0', 'bytes\_', 'c\_', 'can\_cast', 'cast', 'cbrt', 'cdouble', 'ceil', 'cfloat', 'char', 'character', 'chararray', 'choose', 'clip', 'clongdouble', 'clongfloat', 'column\_stack', 'common\_type', 'compare\_chararrays', 'compat', 'complex128', 'complex64', 'complex\_', 'complexfloating', 'compress', 'concatenate', 'conj', 'conjugate', 'convolve', 'copy', 'copysign', 'copyto', 'core', 'corrcoef', 'correlate', 'cos', 'cosh', 'count\_nonzero', 'cov', 'cross', 'csingle', 'ctypeslib', 'cumprod', 'cumproduct', 'cumsum', 'datetime64', 'datetime\_as\_string', 'datetime\_data', 'deg2rad', 'degrees', 'delete', 'deprecate', 'deprecate\_with\_doc', 'diag', 'diag\_indices', 'diag\_indices\_from', 'diagflat', 'diagonal', 'diff', 'digitize', 'disp', 'divide', 'divmod', 'dot', 'double', 'dsplit', 'dstack', 'dtype', 'e', 'ediff1d', 'einsum', 'einsum\_path', 'emath', 'empty', 'empty\_like', 'equal', 'errstate', 'euler\_gamma', 'exp', 'exp2', 'expand\_dims', 'expm1', 'extract', 'eye', 'fabs', 'fastCopyAndTranspose', 'fft', 'fill\_diagonal', 'find\_common\_type', 'finfo', 'fix', 'flatiter', 'flatnonzero', 'flexible', 'flip', 'fliplr', 'flipud', 'float16', 'float32', 'float64', 'float\_', 'float\_power', 'floating', 'floor', 'floor\_divide', 'fmax', 'fmin', 'fmod', 'format\_float\_positional', 'format\_float\_scientific', 'format\_parser', 'frexp', 'frombuffer', 'fromfile', 'fromfunction', 'fromiter', 'frompyfunc', 'fromregex', 'fromstring', 'full', 'full\_like', 'gcd', 'generic', 'genfromtxt', 'geomspace', 'get\_array\_wrap', 'get\_incoll', 'rollaxis', 'roots', 'rot90', 'round', 'round\_', 'row\_stack', 's\_', 'safe\_eval', 'save', 'savetxt', 'savez', 'savez\_compressed', 'sctype2char', 'sctypeDict', 'sctypes', 'searchsorted', 'select', 'set\_numeric\_ops', 'set\_printoptions', 'set\_string\_function', 'setbufsize', 'setdiff1d', 'seterr', 'seterrcall', 'seterrobj', 'setxor1d', 'shape', 'shares\_memory', 'short', 'show\_config', 'sign', 'signbit', 'signedinteger', 'sin', 'sinc', 'single', 'singlecomplex', 'sinh', 'size', 'sometrue', 'sort', 'sort\_complex', 'source', 'spacing', 'split', 'sqrt', 'square', 'squeeze', 'stack', 'std', 'str0', 'str\_', 'string\_', 'subtract', 'sum', 'swapaxes', 'sys', 'take', 'take\_along\_axis', 'tan', 'tanh', 'tensordot', 'test', 'testing', 'tile', 'timedelta64', 'trace', 'tracemalloc\_domain', 'transpose', 'trapz', 'tri', 'tril', 'tril\_indices', 'tril\_indices\_from', 'trim\_zeros', 'triu', 'triu\_indices', 'triu\_indices\_from', 'true\_divide', 'trunc', 'typecodes', 'typename', 'ubyte', 'ufunc', 'uint', 'uint0', 'uint16', 'uint32', 'uint64', 'uint8', 'uintc', 'uintp', 'ulonglong', 'unicode\_', 'union1d', 'unique', 'unpackbits', 'unravel\_index', 'unsignedinteger', 'unwrap', 'use\_hugepage', 'ushort', 'vander', 'var', 'vdot', 'vectorize', 'version', 'void', 'void0', 'vsplit', 'vstack', 'warnings', 'where', 'who', 'zeros', 'zeros\_like']

### scipy

## random

## copy

## matplotlib

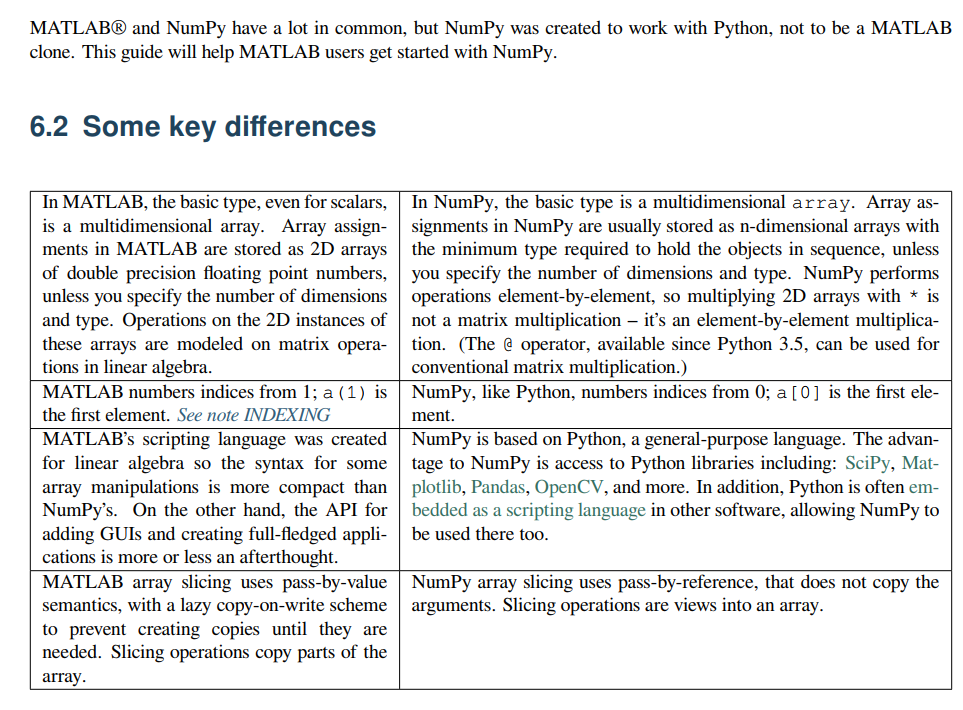
## astropy

## Python time Module

## PyInstaller

[PyInstaller Manual — PyInstaller 5.6.2 documentation](https://pyinstaller.org/en/stable/)

# **Matlab**

****

**6.3.1 General purpose equivalents**

|  |  |  |
| --- | --- | --- |
| MATLAB | NumPy | Notes |
| help func | info(func) or help(func) or func? (in IPython) | get help on the function *func* |
| which func | *see note HELP* | find out where *func* is defined |
| type func | np.source(func) or func?? (in IPython) | print source for *func* (if not a native function) |
| % comment | # comment | comment a line of code with the text comment |
| **for** i=1:3 fprintf('*%i***\n**',i) end | **for** i **in** range(1, 4): print(i) | use a for-loop to print the numbers 1, 2, and 3 using range |
| a && b | a and b | short-circuiting logical AND opera tor (Python native operator); scalar arguments only |
| a || b | a or b | short-circuiting logical OR operator (Python native operator); scalar argu ments only |
| >> 4 == 4 ans = 1 >> 4 == 5 ans = 0 | **>>>** 4 == 4 True **>>>** 4 == 5 False | The boolean objects in Python are True and False, as opposed to MATLAB logical types of 1 and 0. |
| a=4 **if** a==4 fprintf('a = 4\n') **elseif** a==5 fprintf('a = 5\n') **end** | a = 4 **if** a == 4: print('a = 4') **elif** a == 5: print('a = 5') | create an if-else statement to check if a is 4 or 5 and print result |
| 1\*i, 1\*j, 1i, 1j | 1j | complex numbers |
| eps | np.finfo(float).eps or np. spacing(1) | Upper bound to relative error due to rounding in 64-bit floating point arithmetic. |
| load data.mat | io.loadmat('data.mat') | Load MATLAB variables saved to the file data.mat. (Note: When saving arrays to data. mat in MATLAB/Octave, use a recent binary format. scipy.io. loadmat will create a dictionary with the saved arrays and further information.) |
| ode45 | integrate.solve\_ivp(f) | integrate an ODE with Runge-Kutta 4,5 |
| ode15s | integrate.solve\_ivp(f, method='BDF') | integrate an ODE with BDF method |

**6.3.2 Linear algebra equivalents**

|  |  |  |
| --- | --- | --- |
| MATLAB | NumPy | Notes |
| ndims(a) | np.ndim(a) or a.ndim | number of dimensions of array a |
| numel(a) | np.size(a) or a.size | number of elements of array a |
| size(a) | np.shape(a) or a.shape | “size” of array a |
| size(a,n) | a.shape[n-1] | get the number of elements of the n-th dimension of array a. (Note that MATLAB uses 1 based indexing while Python uses 0 based indexing, See note *INDEXING*) |
| [ 1 2 3; 4 5 6 ] | np.array([[1. ,2. ,3.], [4. ,5. ,6.]]) | define a 2x3 2D array |
| [ a b; c d ] | np.block([[a, b], [c, d]]) | construct a matrix from blocks a, b, c, and d |
| a(end) | a[-1] | access last element in MATLAB vec tor (1xn or nx1) or 1D NumPy array a (length n) |
| a(2,5) | a[1, 4] | access element in second row, fifth column in 2D array a |
| a(2,:) | a[1] or a[1, :] | entire second row of 2D array a |
| a(1:5,:) | a[0:5] or a[:5] or a[0:5, :] | first 5 rows of 2D array a |
| a(end-4:end,:) | a[-5:] | last 5 rows of 2D array a |
| a(1:3,5:9) | a[0:3, 4:9] | The first through third rows and fifth through ninth columns of a 2D array, a. |
| a([2,4,5],[1,3]) | a[np.ix\_([1, 3, 4], [0, 2])] | rows 2,4 and 5 and columns 1 and 3. This allows the matrix to be mod ified, and doesn’t require a regular slice. |
| a(3:2:21,:) | a[2:21:2,:] | every other row of a, starting with the third and going to the twenty-first |
| a(1:2:end,:) | a[ ::2,:] | every other row of a, starting with the first |
| a(end:-1:1,:) or flipud(a) | a[::-1,:] | a with rows in reverse order |
| a([1:end 1],:) | a[np.r\_[:len(a),0]] | a with copy of the first row appended to the end |
| a.' | a.transpose() or a.T | transpose of a |
| a' | a.conj().transpose() or a. conj().T | conjugate transpose of a |
| a \* b | a @ b | matrix multiply |
| a .\* b | a \* b | element-wise multiply |
| a./b | a/b | element-wise divide |
| a.^3 | a\*\*3 | element-wise exponentiation |
| (a > 0.5) | (a > 0.5) | matrix whose i,jth element is (a\_ij > 0.5). The MATLAB result is an array of logical values 0 and 1. The NumPy result is an array of the boolean values False and True. |
| find(a > 0.5) | np.nonzero(a > 0.5) | find the indices where (a > 0.5) |

|  |  |  |
| --- | --- | --- |
| MATLAB | NumPy | Notes |
| a(:,find(v > 0.5)) | a[:,np.nonzero(v > 0. 5)[0]] | extract the columns of a where vector v > 0.5 |
| a(:,find(v>0.5)) | a[:, v.T > 0.5] | extract the columns of a where col umn vector v > 0.5 |
| a(a<0.5)=0 | a[a < 0.5]=0 | a with elements less than 0.5 zeroed out |
| a .\* (a>0.5) | a \* (a > 0.5) | a with elements less than 0.5 zeroed out |
| a(:) = 3 | a[:] = 3 | set all values to the same scalar value |
| y=x | y = x.copy() | NumPy assigns by reference |
| y=x(2,:) | y = x[1, :].copy() | NumPy slices are by reference |
| y=x(:) | y = x.flatten() | turn array into vector (note that this forces a copy). To obtain the same data ordering as in MATLAB, use x. flatten('F'). |
| 1:10 | np.arange(1., 11.) or np. | create an increasing vector (see note *RANGES*) |
| 0:9 | np.arange(10.) or np. r\_[:10.] or np.r\_[:9:10j] | create an increasing vector (see note *RANGES*) |
| [1:10]' | np.arange(1.,11.)[:, np.newaxis] | create a column vector |
| zeros(3,4) | np.zeros((3, 4)) | 3x4 two-dimensional array full of 64- bit floating point zeros |
| zeros(3,4,5) | np.zeros((3, 4, 5)) | 3x4x5 three-dimensional array full of 64-bit floating point zeros |
| ones(3,4) | np.ones((3, 4)) | 3x4 two-dimensional array full of 64- bit floating point ones |
| eye(3) | np.eye(3) | 3x3 identity matrix |
| diag(a) | np.diag(a) | returns a vector of the diagonal ele ments of 2D array, a |
| diag(v,0) | np.diag(v, 0) | returns a square diagonal matrix whose nonzero values are the ele ments of vector, v |
| rng(42,'twister') rand(3,4) | **from numpy.random import**␣ *,→*default\_rng rng = default\_rng(42) rng.random(3, 4) or older version: random. rand((3, 4)) | generate a random 3x4 array with de fault random number generator and seed = 42 |
| linspace(1,3,4) | np.linspace(1,3,4) | 4 equally spaced samples between 1 and 3, inclusive |
| [x,y]=meshgrid(0:8,0:5) | np.mgrid[0:9.,0:6.] or np.meshgrid(r\_[0:9.], r\_[0:6.] | two 2D arrays: one of x values, the other of y values |
| ogrid[0:9.,0:6.] or np. ix\_(np.r\_[0:9.],np. r\_[0:6.] | the best way to eval functions on a grid |  |

np.r\_[1.:11.] or r\_[1:10:10j]

|  |  |  |
| --- | --- | --- |
| MATLAB | NumPy | Notes |
| [x,y]=meshgrid([1,2,4], [2,4,5]) | np.meshgrid([1,2,4],[2, 4,5]) |  |
| ix\_([1,2,4],[2,4,5]) | the best way to eval functions on a grid |  |
| repmat(a, m, n) | np.tile(a, (m, n)) | create m by n copies of a |
| [a b] | np.concatenate((a,b), 1) or np.hstack((a,b)) or np.column\_stack((a,b)) or np.c\_[a,b] | concatenate columns of a and b |
| [a; b] | np.concatenate((a,b)) or np.vstack((a,b)) or np.r\_[a,b] | concatenate rows of a and b |
| max(max(a)) | a.max() or np.nanmax(a) | maximum element of a (with ndims(a)<=2 for MATLAB, if there are NaN’s, nanmax will ignore these and return largest value) |
| max(a) | a.max(0) | maximum element of each column of array a |
| max(a,[],2) | a.max(1) | maximum element of each row of ar ray a |
| max(a,b) | np.maximum(a, b) | compares a and b element-wise, and returns the maximum value from each pair |
| norm(v) | np.sqrt(v @ v) or np. linalg.norm(v) | L2 norm of vector v |
| a & b | logical\_and(a,b) | element-by-element AND operator (NumPy ufunc) *See note LOGICOPS* |
| a | b | np.logical\_or(a,b) | element-by-element OR operator (NumPy ufunc) *See note LOGICOPS* |
| bitand(a,b) | a & b | bitwise AND operator (Python native and NumPy ufunc) |
| bitor(a,b) | a | b | bitwise OR operator (Python native and NumPy ufunc) |
| inv(a) | linalg.inv(a) | inverse of square 2D array a |
| pinv(a) | linalg.pinv(a) | pseudo-inverse of 2D array a |
| rank(a) | linalg.matrix\_rank(a) | matrix rank of a 2D array a |
| a\b | linalg.solve(a, b) if a is square; linalg.lstsq(a, b) otherwise | solution of a x = b for x |
| b/a | Solve a.T x.T = b.T instead | solution of x a = b for x |
| [U,S,V]=svd(a) | U, S, Vh = linalg. svd(a), V = Vh.T | singular value decomposition of a |
| c=chol(a) where a==c'\*c | c = linalg.cholesky(a) where a == c@c.T | Cholesky factorization of a 2D ar ray (chol(a) in MATLAB returns an upper triangular 2D array, but cholesky returns a lower triangu lar 2D array) |
| [V,D]=eig(a) | D,V = linalg.eig(a) | eigenvalues *λ* and eigenvectors *v*¯ of a, where *λv*¯ = **a***v*¯ |

|  |  |  |
| --- | --- | --- |
| MATLAB | NumPy | Notes |
| [V,D]=eig(a,b) | D,V = linalg.eig(a, b) | eigenvalues *λ* and eigenvectors *v*¯ of a, b where *λ***b***v*¯ = **a***v*¯ |
| [V,D]=eigs(a,3) | D,V = eigs(a, k = 3) | find the k=3 largest eigenvalues and eigenvectors of 2D array, a |
| [Q,R,P]=qr(a,0) | Q,R = linalg.qr(a) | QR decomposition |
| [L,U,P]=lu(a) where a==P'\*L\*U | P,L,U = linalg.lu(a) where a == P@L@U | (note: trans |
| conjgrad | cg | Conjugate gradients solver |
| fft(a) | np.fft(a) | Fourier transform of a |
| ifft(a) | np.ifft(a) | inverse Fourier transform of a |
| sort(a) | np.sort(a) or a. sort(axis=0) | sort each column of a 2D array, a |
| sort(a, 2) | np.sort(a, axis = 1) or a. sort(axis = 1) | sort the each row of 2D array, a |
| [b,I]=sortrows(a,1) | I = np.argsort(a[:, 0]); b = a[I,:] | save the array a as array b with rows sorted by the first column |
| x = Z\y | x = linalg.lstsq(Z, y) | perform a linear regression of the form **Zx** = **y** |
| decimate(x, q) | signal.resample(x, np. ceil(len(x)/q)) | downsample with low-pass filtering |
| unique(a) | np.unique(a) | a vector of unique values in array a |
| squeeze(a) | a.squeeze() | remove singleton dimensions of array a. Note that MATLAB will always return arrays of 2D or higher while NumPy will return arrays of 0D or higher |

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

[mat4py · PyPI](https://pypi.org/project/mat4py/)

[https://pypi.org › project](https://pypi.org/project/mat4py/)

[Traduci questa pagina](https://translate.google.it/translate?hl=it&sl=en&u=https://pypi.org/project/mat4py/&prev=search&pto=aue)

**Matlab structs** and cells are represented using **Python** dicts. ... MAT-file into a simple **Python** data **structure**, using only **Python's dict** and list objects.

**How to load Matlab .mat files in Python**

Matlab is a really popular platform for scientific computing in the academia. I’ve used it my throughout my engineering degree and chances are, you will come across .mat files for datasets released by the universities.

This is a brief post which explains how to load these files using python, the most popular language for machine learning today.

**The data**

I wanted to build a classifier for detecting cars of different models and makes and so the [Stanford Cars Dataset](https://ai.stanford.edu/~jkrause/cars/car_dataset.html) appeared to be a great starting point. Coming from the academia, the annotations for the dataset was in the .mat format. You can get the file used in this post [here](https://ai.stanford.edu/~jkrause/cars/car_devkit.tgz).

**Loading .mat files**

Scipy is a really popular python library used for scientific computing and quite naturally, they have a method which lets you read in .mat files. Reading them in is definitely the easy part. You can get it done in one line of code:

from scipy.io import loadmat  
annots = loadmat('cars\_train\_annos.mat')

Well, it’s really that simple. But let’s go on and actually try to get the data we need out of this dictionary.

**Formatting the data**

The loadmat method returns a more familiar data structure, a python dictionary. If we peek into the keys, we’ll see how at home we feel now compared to dealing with a .mat file:

annots.keys()  
> dict\_keys(['\_\_header\_\_', '\_\_version\_\_', '\_\_globals\_\_', 'annotations'])

Looking at the documentation for this dataset, we’ll get to learn what this is really made of. The README.txt gives us the following information:

This file gives documentation for the cars 196 dataset.  
(<http://ai.stanford.edu/~jkrause/cars/car_dataset.html>) — — — — — — — — — — — — — — — — — — — —   
Metadata/Annotations  
 — — — — — — — — — — — — — — — — — — — —   
Descriptions of the files are as follows:-cars\_meta.mat:  
 Contains a cell array of class names, one for each class.-cars\_train\_annos.mat:  
 Contains the variable ‘annotations’, which is a struct array of length  
 num\_images and where each element has the fields:  
 bbox\_x1: Min x-value of the bounding box, in pixels  
 bbox\_x2: Max x-value of the bounding box, in pixels  
 bbox\_y1: Min y-value of the bounding box, in pixels  
 bbox\_y2: Max y-value of the bounding box, in pixels  
 class: Integral id of the class the image belongs to.  
 fname: Filename of the image within the folder of images.-cars\_test\_annos.mat:  
 Same format as ‘cars\_train\_annos.mat’, except the class is not provided.

# Matlab

# MATLAB commands in numerical Python (NumPy)

Copyright Vidar Bronken Gundersen

Permission is granted to copy, distribute and/or modify this document as long as the above attribution is kept and the resulting work is distributed under a license identical to this one.

The idea of this document (and the corresponding xml instance) is to provide a quick reference*[[1]](#footnote-1)* for switching from matlab to an open-source environment, such as Python, Scilab, Octave and Gnuplot, or R for numeric processing and data visualisation.

Where Octave and Scilab commands are omitted, expect Matlab compatibility, and similarly where non given use the generic command. Time-stamp: **-**-**T**:**:** vidar

# 1 Help

|  |  |  |  |
| --- | --- | --- | --- |
| Desc. matlab/Octave | | Python | R |
| Browse help interactively doc  Octave:help -i % browse with Info | | help() | help.start() |
| Help on using help help help or doc doc | | help | help() |
| Help for a function help plot | | help(plot) or ?plot | help(plot) or ?plot |
| Help for a toolbox/library package help splines or doc splines | | help(pylab) | help(package=’splines’) |
| Demonstration examples demo | |  | demo() |
| Example using a function  1.1 Searching available documentation | |  | example(plot) |
| Desc. | matlab/Octave | Python | R |
| Search help files | lookfor plot |  | help.search(’plot’) |
| Find objects by partial name |  |  | apropos(’plot’) |
| List available packages | help | help(); modules [Numeric] | library() |
| Locate functions | which plot | help(plot) | find(plot) |
| List available methods for a function  1.2 Using interactively |  |  | methods(plot) |
| Desc. | matlab/Octave | Python | R |
| Start session | Octave:octave -q | ipython -pylab | Rgui |
| Auto completion | Octave:TAB or M-? | TAB |  |
| Run code from file | foo(.m) | execfile(’foo.py’) or run foo.py | source(’foo.R’) |
| Command history | Octave:history | hist -n | history() |
| Save command history | diary on [..] diary off |  | savehistory(file=".Rhistory") |
| End session  2 Operators | exit or quit | CTRL-D  CTRL-Z # windows sys.exit() | q(save=’no’) |
| Desc. | matlab/Octave | Python | R |
| Help on operator syntax | help - |  | help(Syntax) |

2.1 Arithmetic operators

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | Assignment; defining a number | a=1; b=2; | a=1; b=1 | a<-1; b<-2 | | Addition | a + b | a + b or add(a,b) | a + b | | Subtraction | a - b | a - b or subtract(a,b) | a - b | | Multiplication | a \* b | a \* b or multiply(a,b) | a \* b | | Division | a / b | a / b or divide(a,b) | a / b | | Power, *ab* | a .^ b | a \*\* b  power(a,b) pow(a,b) | a ^ b | | Remainder | rem(a,b) | a % b  remainder(a,b) fmod(a,b) | a %% b | | Integer division |  |  | a %/% b | | In place operation to save array creation overhead | Octave:a+=1 | a+=b or add(a,b,a) |  | | Factorial, *n*!  2.2 Relational operators | factorial(a) |  | factorial(a) | | Desc. | matlab/Octave | Python | R | | Equal | a == b | a == b or equal(a,b) | a == b | | Less than | a < b | a < b or less(a,b) | a < b | | Greater than | a > b | a > b or greater(a,b) | a > b | | Less than or equal | a <= b | a <= b or less\_equal(a,b) | a <= b | | Greater than or equal | a >= b | a >= b or greater\_equal(a,b) | a >= b | | Not Equal  2.3 Logical operators | a ~= b | a != b or not\_equal(a,b) | a != b | | Desc. | matlab/Octave | Python | R | | Short-circuit logical AND | a && b | a and b | a && b | | Short-circuit logical OR | a || b | a or b | a || b | | Element-wise logical AND | a & b or and(a,b) | logical\_and(a,b) or a and b | a & b | | Element-wise logical OR | a | b or or(a,b) | logical\_or(a,b) or a or b | a | b | | Logical EXCLUSIVE OR | xor(a, b) | logical\_xor(a,b) | xor(a, b) | | Logical NOT | ~a or not(a)  Octave:~a or !a | logical\_not(a) or not a | !a | | True if any element is nonzero | any(a) |  |  | | True if all elements are nonzero  2.4 root and logarithm | all(a) |  |  | | Desc. | matlab/Octave | Python | R | | Square root | sqrt(a) | math.sqrt(a) | sqrt(a) | | Logarithm, base *e* (natural) | log(a) | math.log(a) | log(a) | | Logarithm, base ** | log10(a) | math.log10(a) | log10(a) | | Logarithm, base ** (binary) | log2(a) | math.log(a, 2) | log2(a) | | Exponential function | exp(a) | math.exp(a) | exp(a) | |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Round round(a) | | around(a) or math.round(a) | round(a) |  |
| Round up ceil(a) | | ceil(a) | ceil(a) |  |
| Round down floor(a) | | floor(a) | floor(a) |  |
| Round towards zero fix(a)  2.6 Mathematical constants | | fix(a) |  |  |
| Desc. matlab/Octave | | Python | R |  |
| *π* = 3*.*141592 pi | | math.pi | pi |  |
| *e* = 2*.*718281 exp(1)  2.6.1 Missing values; IEEE-754 floating point status flags | | math.e or math.exp(1) | exp(1) |  |
| Desc. | matlab/Octave | Python | R |  |
| Not a Number | NaN | nan |  |  |
| Infinity, ∞ | Inf | inf |  |  |
| Infinity, +∞ |  | plus\_inf |  |  |
| Infinity, −∞ |  | minus\_inf |  |  |
| Plus zero, +0 |  | plus\_zero |  |  |
| Minus zero, −0  2.7 Complex numbers |  | minus\_zero |  |  |
| Desc.  Imaginary unit | matlab/Octave i | Python z = 1j | R  1i | √ |
| *i* = −1 |
| A complex number, 3 + 4*i* | z = 3+4i | z = 3+4j or z = complex(3,4) | z <- 3+4i |  |
| Absolute value (modulus) | abs(z) | abs(3+4j) | abs(3+4i) or Mod(3+4i) |  |
| Real part | real(z) | z.real | Re(3+4i) |  |
| Imaginary part | imag(z) | z.imag | Im(3+4i) |  |
| Argument | arg(z) |  | Arg(3+4i) |  |
| Complex conjugate  2.8 Trigonometry | conj(z) | z.conj(); z.conjugate() | Conj(3+4i) |  |
| Desc. | matlab/Octave | Python | R |  |
| Arctangent, arctan(*b/a*) | atan(a,b) | atan2(b,a) | atan2(b,a) |  |
| Hypotenus; Euclidean distance |  | hypot(x,y) |  | p 2 + *y*2 *x* |
| 2.9 Generate random numbers | |
| Desc. matlab/Octave | | Python | R |  |
| Uniform distribution rand(1,10) | | random.random((10,)) random.uniform((10,)) | runif(10) |  |
| Uniform: Numbers between ** and ** 2+5\*rand(1,10) | | random.uniform(2,7,(10,)) | runif(10, min=2, max=7) |  |
| Uniform: **,** array rand(6) | | random.uniform(0,1,(6,6)) | matrix(runif(36),6) |  |
| Normal distribution randn(1,10) | | random.standard\_normal((10,)) | rnorm(10) |  |

2.5 Round off

# 3 Vectors

|  |  |  |  |
| --- | --- | --- | --- |
| Row vector, 1 × *n*-matrix | a=[2 3 4 5]; | a=array([2,3,4,5]) | a <- c(2,3,4,5) |
| Column vector, *m* × 1-matrix  3.1 Sequences | adash=[2 3 4 5]’; | array([2,3,4,5])[:,NewAxis] array([2,3,4,5]).reshape(-1,1) r\_[1:10,’c’] | adash <- t(c(2,3,4,5)) |
| Desc. | matlab/Octave | Python | R |
| **,**,**, ... ,** | 1:10 | arange(1,11, dtype=Float) range(1,11) | seq(10) or 1:10 |
| **.**,**.**,**.**, ... ,**.** | 0:9 | arange(10.) | seq(0,length=10) |
| **,**,**,** | 1:3:10 | arange(1,11,3) | seq(1,10,by=3) |
| **,**,**, ... ,** | 10:-1:1 | arange(10,0,-1) | seq(10,1) or 10:1 |
| **,**,**,** | 10:-3:1 | arange(10,0,-3) | seq(from=10,to=1,by=-3) |
| Linearly spaced vector of n=** points | linspace(1,10,7) | linspace(1,10,7) | seq(1,10,length=7) |
| Reverse | reverse(a) | a[::-1] or | rev(a) |
| Set all values to same scalar value | a(:) = 3 | a.fill(3), a[:] = 3 |  |
| 3.2 Concatenation (vectors) | |
| Desc. matlab/Octave | | Python | R |
| Concatenate two vectors [a a] | | concatenate((a,a)) | c(a,a) |
| [1:4 a]  3.3 Repeating | | concatenate((range(1,5),a), axis=1) c(1:4,a) | |
| Desc. matlab/Octave | | Python R | |
| *  *, *  * [a a] | | concatenate((a,a)) rep(a,times=2) | |
| *  *, *  *, *  * | | a.repeat(3) or rep(a,each=3) | |
| **, * *, *  *  3.4 Miss those elements out | | a.repeat(a) or rep(a,a) | |
| Desc. matlab/Octave | | Python R | |
| miss the first element a(2:end) | | a[1:] a[-1] | |
| miss the tenth element a([1:9]) | | a[-10] | |
| miss **,**,**, ... | | a[-seq(1,50,3)] | |
| last element a(end) | | a[-1] | |
| last two elements a(end-1:end)  3.5 Maximum and minimum | | a[-2:] | |
| Desc. matlab/Octave | | Python R | |
| pairwise max max(a,b) | | maximum(a,b) pmax(a,b) | |
| max of all values in two vectors max([a b]) | | concatenate((a,b)).max() max(a,b) | |
| [v,i] = max(a) | | v,i = a.max(0),a.argmax(0) v <- max(a) ; i <- which.max(a) | |

3.6 Vector multiplication

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | Multiply two vectors a.\*a | | a\*a | a\*a | | Vector dot product, *u* · *v* dot(u,v)  4 Matrices | | dot(u,v) |  | | Desc. matlab/Octave | | Python | R | | Define a matrix a = [2 3;4 5]  4.1 Concatenation (matrices); rbind and cbind | | a = array([[2,3],[4,5]]) | rbind(c(2,3),c(4,5)) array(c(2,3,4,5), dim=c(2,2)) | | Desc. | matlab/Octave | Python | R | | Bind rows | [a ; b] | concatenate((a,b), axis=0) vstack((a,b)) | rbind(a,b) | | Bind columns | [a , b] | concatenate((a,b), axis=1) hstack((a,b)) | cbind(a,b) | | Bind slices (three-way arrays) |  | concatenate((a,b), axis=2) dstack((a,b)) |  | | Concatenate matrices into one vector | [a(:), b(:)] | concatenate((a,b), axis=None) |  | | Bind rows (from vectors) | [1:4 ; 1:4] | concatenate((r\_[1:5],r\_[1:5])).reshape(2,-1)rbind(1:4,1:4) | | | vstack((r\_[1:5],r\_[1:5])) |  | | Bind columns (from vectors)  4.2 Array creation | [1:4 ; 1:4]’ |  | cbind(1:4,1:4) | | Desc. | matlab/Octave | Python | R | | ** filled array | zeros(3,5) | zeros((3,5),Float) | matrix(0,3,5) or array(0,c(3,5)) | | ** filled array of integers |  | zeros((3,5)) |  | | ** filled array | ones(3,5) | ones((3,5),Float) | matrix(1,3,5) or array(1,c(3,5)) | | Any number filled array | ones(3,5)\*9 |  | matrix(9,3,5) or array(9,c(3,5)) | | Identity matrix | eye(3) | identity(3) | diag(1,3) | | Diagonal | diag([4 5 6]) | diag((4,5,6)) | diag(c(4,5,6)) | | Magic squares; Lo Shu | magic(3) |  |  | | Empty array |  | a = empty((3,3)) |  | |  |
|  |

4.3 Reshape and flatten matrices

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | Reshaping (rows first) reshape(1:6,3,2)’; arange(1,7).reshape(2,-1) matrix(1:6,nrow=3,byrow=T) a.setshape(2,3)  Reshaping (columns first) reshape(1:6,2,3); arange(1,7).reshape(-1,2).transpose()matrix(1:6,nrow=2) | | | | | |  | | | array(1:6,c(2,3)) | | | Flatten to vector (by rows, like comics) a’(:) a.flatten() or | | | as.vector(t(a)) | | | Flatten to vector (by columns) a(:) a.flatten(1) | | | as.vector(a) | | | Flatten upper triangle (by columns) vech(a)  4.4 Shared data (slicing) | | | a[row(a) <= col(a)] | | | Desc. matlab/Octave Python | | | R | | | Copy of a b = a b = a.copy()  4.5 Indexing and accessing elements (Python: slicing) | | | b = a | | | Desc. | matlab/Octave | Python | R | | | Input is a **,** array | a = [ 11 12 13 14 ... 21 22 23 24 ...  31 32 33 34 ] | a = array([[ 11, 12, 13, 14 ], [ 21, 22, 23, 24 ],  [ 31, 32, 33, 34 ]]) | a <- rbind(c(11, 12, 13, 14), | | |  | c(21, 22, 23, 24), c(31, 32, 33, 34)) | | Element **,** (row,col) | a(2,3) | a[1,2] | a[2,3] |  | | First row | a(1,:) | a[0,] | a[1,] |  | | First column | a(:,1) | a[:,0] | a[,1] |  | | Array as indices | a([1 3],[1 4]); | a.take([0,2]).take([0,3], axis=1) |  |  | | All, except first row | a(2:end,:) | a[1:,] | a[-1,] |  | | Last two rows | a(end-1:end,:) | a[-2:,] |  |  | | Strides: Every other row | a(1:2:end,:) | a[::2,:] |  |  | | Third in last dimension (axis) |  | a[...,2] |  |  | | All, except row,column (**,**) |  |  | a[-2,-3] |  | | Remove one column | a(:,[1 3 4]) | a.take([0,2,3],axis=1) | a[,-2] |  | | Diagonal |  | a.diagonal(offset=0) |  |  | |  |
|  |

4.6 Assignment

|  |  |  |  |
| --- | --- | --- | --- |
| Desc. | matlab/Octave | Python | R |
|  | a(:,1) = 99 | a[:,0] = 99 | a[,1] <- 99 |
|  | a(:,1) = [99 98 97]’ | a[:,0] = array([99,98,97]) | a[,1] <- c(99,98,97) |
| Clipping: Replace all elements over ** | a(a>90) = 90; | (a>90).choose(a,90)  a.clip(min=None, max=90) | a[a>90] <- 90 |
| Clip upper and lower values  4.7 Transpose and inverse |  | a.clip(min=2, max=5) |  |
| Desc. | matlab/Octave | Python | R |
| Transpose | a’ | a.conj().transpose() | t(a) |
| Non-conjugate transpose | a.’ or transpose(a) | a.transpose() |  |
| Determinant | det(a) | linalg.det(a) or | det(a) |
| Inverse | inv(a) | linalg.inv(a) or | solve(a) |
| Pseudo-inverse | pinv(a) | linalg.pinv(a) | ginv(a) |
| Norms | norm(a) | norm(a) |  |
| Eigenvalues | eig(a) | linalg.eig(a)[0] | eigen(a)$values |
| Singular values | svd(a) | linalg.svd(a) | svd(a)$d |
| Cholesky factorization | chol(a) | linalg.cholesky(a) |  |
| Eigenvectors | [v,l] = eig(a) | linalg.eig(a)[1] | eigen(a)$vectors |
| Rank  4.8 Sum | rank(a) | rank(a) | rank(a) |
| Desc. | matlab/Octave | Python | R |
| Sum of each column | sum(a) | a.sum(axis=0) | apply(a,2,sum) |
| Sum of each row | sum(a’) | a.sum(axis=1) | apply(a,1,sum) |
| Sum of all elements | sum(sum(a)) | a.sum() | sum(a) |
| Sum along diagonal |  | a.trace(offset=0) |  |
| Cumulative sum (columns) | cumsum(a) | a.cumsum(axis=0) | apply(a,2,cumsum) |

4.9 Sorting

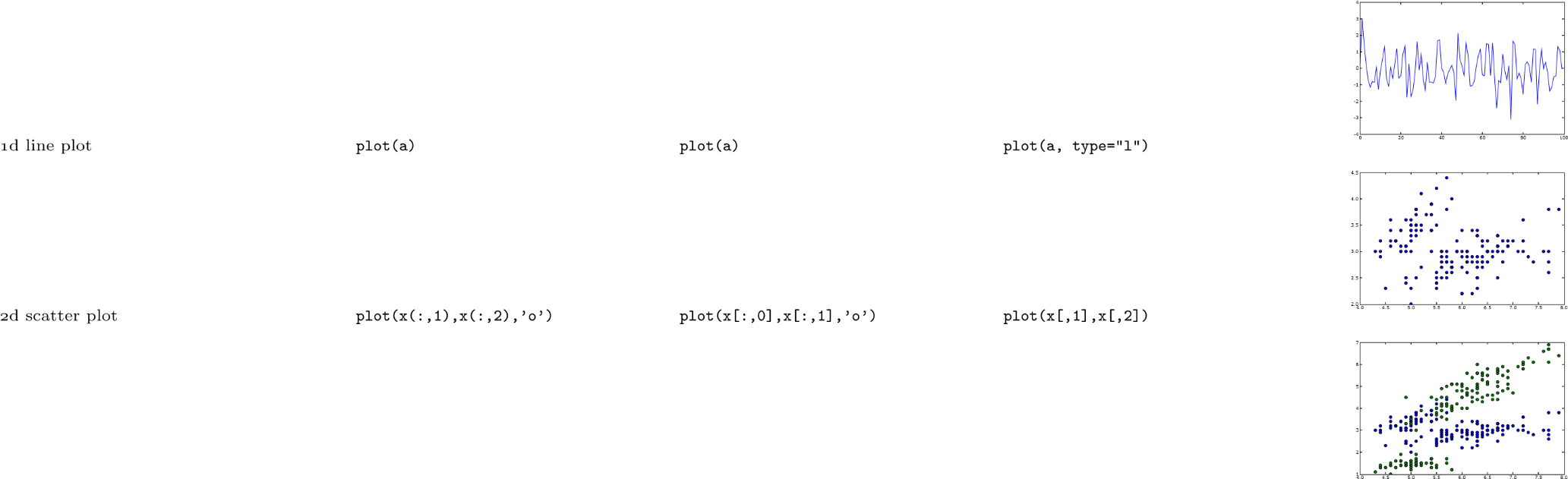
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | Example data a = [ 4 3 2 ; 2 8 6 ; 1 4 7 ] | | a = array([[4,3,2],[2,8,6],[1,4,7]]) |  | | Flat and sorted sort(a(:)) | | a.ravel().sort() or | t(sort(a)) | | Sort each column sort(a) | | a.sort(axis=0) or msort(a) | apply(a,2,sort) | | Sort each row sort(a’)’ | | a.sort(axis=1) | t(apply(a,1,sort)) | | Sort rows (by first row) sortrows(a,1) | | a[a[:,0].argsort(),] |  | | Sort, return indices | | a.ravel().argsort() | order(a) | | Sort each column, return indices | | a.argsort(axis=0) |  | | Sort each row, return indices  4.10 Maximum and minimum | | a.argsort(axis=1) |  | | Desc. | matlab/Octave | Python | R | | max in each column | max(a) | a.max(0) or amax(a [,axis=0]) | apply(a,2,max) | | max in each row | max(a’) | a.max(1) or amax(a, axis=1) | apply(a,1,max) | | max in array | max(max(a)) | a.max() or | max(a) | | return indices, i | [v i] = max(a) |  | i <- apply(a,1,which.max) | | pairwise max | max(b,c) | maximum(b,c) | pmax(b,c) | |  | cummax(a) |  | apply(a,2,cummax) | | max-to-min range  4.11 Matrix manipulation |  | a.ptp(); a.ptp(0) |  | | Desc. | matlab/Octave | Python | R | | Flip left-right | fliplr(a) | fliplr(a) or a[:,::-1] | a[,4:1] | | Flip up-down | flipud(a) | flipud(a) or a[::-1,] | a[3:1,] | | Rotate ** degrees | rot90(a) | rot90(a) |  | | Repeat matrix: [ a a a ; a a a ] | repmat(a,2,3)  Octave:kron(ones(2,3),a) | kron(ones((2,3)),a) | kronecker(matrix(1,2,3),a) | | Triangular, upper | triu(a) | triu(a) | a[lower.tri(a)] <- 0 | | Triangular, lower  4.12 Equivalents to ”size” | tril(a) | tril(a) | a[upper.tri(a)] <- 0 | | Desc. | matlab/Octave | Python | R | | Matrix dimensions | size(a) | a.shape or a.getshape() | dim(a) | | Number of columns | size(a,2) or length(a) | a.shape[1] or size(a, axis=1) | ncol(a) | | Number of elements | length(a(:)) | a.size or size(a[, axis=None]) | prod(dim(a)) | | Number of dimensions | ndims(a) | a.ndim |  | | Number of bytes used in memory |  | a.nbytes | object.size(a) | |  |

4.13 Matrix- and elementwise- multiplication

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Desc. | matlab/Octave |  |  |  |  | |  |
| Elementwise operations | a .\* b | a \* b or multiply(a,b) | a \* b | h 1  9 |  | |  |
| Matrix product (dot product) | a \* b | matrixmultiply(a,b) | a %\*% b | h 7  15 |  | |  |
| Inner matrix vector multiplication *a* · *b*0 |  | inner(a,b) or |  | h 5  11 |  | |  |
| Outer product |  | outer(a,b) or | outer(a,b) or a %o% b | " 1  2  3  4 | 2 3  4 6  6 9  8 12 | |  |
| Cross product |  |  | crossprod(a,b) or t(a) %\*% b | h 10  14 | 14 i | |  |
| 20 |  |
| Kronecker product | kron(a,b) | kron(a,b) | kronecker(a,b) | " 1  3  3  9 | 2  4  6  12 | 2  6  4  12 |  |
| Matrix division, *b*·*a*−1 | a / b |  |  |  | |  |  |
| Left matrix division, *b*−1·*a*  (solve linear equations) | a \ b | linalg.solve(a,b) | solve(a,b) | *Ax* = *b* | |  |  |
| Vector dot product |  | vdot(a,b) |  |  | |  |  |
| Cross product |  | cross(a,b) |  |  | |  |  |
| 4.14 Find; conditional indexing | |
| Desc. matlab/Octave | | Python | R |  | |  |  |
| Non-zero elements, indices find(a) | | a.ravel().nonzero() | which(a != 0) |  | |  |  |
| Non-zero elements, array indices [i j] = find(a) | | (i,j) = a.nonzero()  (i,j) = where(a!=0) | which(a != 0, arr.ind=T) |  | |  |  |
| Vector of non-zero values [i j v] = find(a) | | v = a.compress((a!=0).flat) v = extract(a!=0,a) | ij <- which(a != 0, arr.ind=T); v <- a[ij] | | |  |  |
| Condition, indices find(a>5.5) | | (a>5.5).nonzero() | which(a>5.5) | | |  |  |
| Return values | | a.compress((a>5.5).flat) | ij <- which(a>5.5, arr.ind=T); v <- a[ij] | | |  |  |
| Zero out elements above **.** a .\* (a>5.5) | | where(a>5.5,0,a) or a \* (a>5.5) |  | | |  |  |
| Replace values  5 Multi-way arrays | | a.put(2,indices) |  | | |  |  |
| Desc. matlab/Octave | | Python | R | | |  |  |
| Define a **-way array a = cat(3, [1 2; 1 2],[3 4; 3 4]); | | a = array([[[1,2],[1,2]], [[3,4],[3,4]]]) | | | |  |  |
| a(1,:,:) | | a[0,...] | | | |  |  |

# 6 File input and output

|  |  |  |
| --- | --- | --- |
| Reading from a file (**d) | f = load(’data.txt’) | f = fromfile("data.txt") f <- read.table("data.txt") f = load("data.txt") |
| Reading from a file (**d) | f = load(’data.txt’) | f = load("data.txt") f <- read.table("data.txt") |
| Reading fram a CSV file (**d) | x = dlmread(’data.csv’, ’;’) | f = load(’data.csv’, delimiter=’;’) f <- read.table(file="data.csv", sep=";") |
| Writing to a file (**d) | save -ascii data.txt f | save(’data.csv’, f, fmt=’%.6f’, delimiwtreirt=e’(;f’,)file="data.txt") |
| Writing to a file (**d) |  | f.tofile(file=’data.csv’, format=’%.6f’, sep=’;’) |
| Reading from a file (**d)   1. Plotting   7.1 Basic x-y plots |  | f = fromfile(file=’data.csv’, sep=’;’) |
| Desc. | matlab/Octave | Python R |



|  |  |  |  |
| --- | --- | --- | --- |
| Two graphs in one plot | plot(x1,y1, x2,y2) | plot(x1,y1,’bo’, x2,y2,’go’) |  |
| Overplotting: Add new plots to current | plot(x1,y1) hold on plot(x2,y2) | plot(x1,y1,’o’) plot(x2,y2,’o’) show() # as normal | plot(x1,y1) matplot(x2,y2,add=T) |
| subplots | subplot(211) | subplot(211) |  |
| Plotting symbols and color | plot(x,y,’ro-’) | plot(x,y,’ro-’) | plot(x,y,type="b",col="red") |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Plot a function for given range | ezplot(f,[0,40])  fplot(’sin(x/3) - cos(x/5)’,[0,40]) Octave:% no ezplot | = arrayrange(0,40,.5)y = sin(x/3) - cos(x/5) plot(x,y, ’o’) | plot(f, xlim=c(0,40), type=’p’) | x |

## 7.1.1 Axes and titles

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

●

0

10

20

30

40

−2.0

−1.5

−1.0

−0.5

0.0

0.5

1.0

f (x)

|  |  |  |  |
| --- | --- | --- | --- |
| Desc. | matlab/Octave |  |  |
| Turn on grid lines | grid on | grid() | grid() |
| **:** aspect ratio | axis equal Octave: axis(’equal’) replot | figure(figsize=(6,6)) | plot(c(1:10,10:1), asp=1) |
| Set axes manually | axis([ 0 10 0 5 ]) | axis([ 0, 10, 0, 5 ]) | plot(x,y, xlim=c(0,10), ylim=c(0,5)) |
| Axis labels and titles | title(’title’) xlabel(’x-axis’) ylabel(’y-axis’) |  | plot(1:10, main="title", xlab="x-axis", ylab="y-axis") |
| Insert text  7.1.2 Log plots |  | text(2,25,’hello’) |  |
| Desc. | matlab/Octave | Python | R |
| logarithmic y-axis | semilogy(a) | semilogy(a) | plot(x,y, log="y") |
| logarithmic x-axis | semilogx(a) | semilogx(a) | plot(x,y, log="x") |
| logarithmic x and y axes  7.1.3 Filled plots and bar plots | loglog(a) | loglog(a) | plot(x,y, log="xy") |
| Desc. | matlab/Octave | Python | R |

Filled plot fill(t,s,’b’, t,c,’g’) fill(t,s,’b’, t,c,’g’, alpha=0.2) plot(t,s, type="n", xlab="", ylab="")Immagine che contiene testo, coltello

Descrizione generata automaticamente

Octave:% fill has a bug? polygon(t,s, col="lightblue")

polygon(t,c, col="lightgreen")

1. 5
2. 71

Stem-and-Leaf plot stem(x[,3]) 7 0338 00113345567889

1. 0133566677788
2. 32674

## 7.1.4 Functions

Desc. matlab/Octave Python R

Defining functions f = inline(’sin(x/3) - cos(x/5)’) f <- function(x) sin(x/3) - cos(x/5) 

7.2 Polar plots

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Desc. | matlab/Octave | Python | R |  |
|  | theta = 0:.001:2\*pi; r = sin(2\*theta); | theta = arange(0,2\*pi,0.001) r = sin(2\*theta) |  | *ρ*(*θ*) = sin(2*θ*) |

0

45

90

135

180

225

270

315

|  |  |  |  |
| --- | --- | --- | --- |
| 7.3 Histogram plots | polar(theta, rho) | polar(theta, rho) |  |
| Desc. | matlab/Octave | Python | R |
|  | hist(randn(1000,1)) |  | hist(rnorm(1000)) |
|  | hist(randn(1000,1), -4:4) |  | hist(rnorm(1000), breaks= -4:4)  hist(rnorm(1000), breaks=c(seq(-5,0,0.25), seq(0.5,5,0.5)), freq=F) |
|  | plot(sort(a)) |  | plot(apply(a,1,sort),type="l") |

7.4 3d data

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | -2 | -1 | 0 | 1 | 2 |
| Filled contour plot | contourf(z); colormap(gray) | contourf(Z, V, cmap=cm.gray, origin=’lower’, extent=(-3,3,-3,3)) | filled.contour(x,y,z, nlevels=7, color=gray.colors) |  |  |  |  |  |

## 7.4.1 Contour and image plots

|  |  |  |  |
| --- | --- | --- | --- |
| Desc. | matlab/Octave | Python | R |

-2

-1

0

1

2

-0.6

-0.4

-0.2

-0.2

0.0

0.2

0.4

0.6

0.6

0.8

0.8

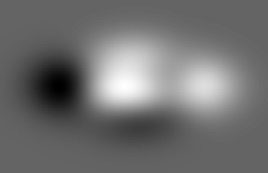
1.0

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | -2 | -1 | 0 | 1 | 2 |
| Contour plot | contour(z) | levels, colls = contour(Z, V, contour(z) origin=’lower’, extent=(-3,3,-3,3))  clabel(colls, levels, inline=1, fmt=’%1.1f’, fontsize=10) |  |  |  |  |  |

Plot image data image(z) im = imshow(Z, image(z, col=gray.colors(256)) Immagine che contiene testo, elettronico

Descrizione generata automaticamente

|  |  |
| --- | --- |
| colormap(gray) | interpolation=’bilinear’, origin=’lower’, extent=(-3,3,-3,3)) |



-2

-1

0

1

2

-0.6

-0.4

-0.2

-0.2

0.0

0.2

0.4

0.6

0.6

0.8

0.8

1.0

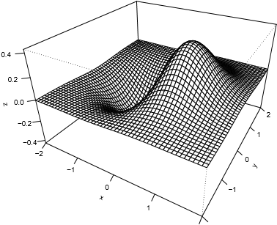
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Image with contours |  | # imshow() and contour() as above | -2 | -1 | 0 | 1 | 2 |
| Direction field vectors | quiver() | quiver() |  |  |  |  |  |

## 7.4.2 Perspective plots of surfaces over the x-y plane

Desc. matlab/Octave

n=-2:.1:2; n=arrayrange(-2,2,.1) f <- function(x,y) x\*exp(-x^2-y^2) *f*(*x,y*) = *xe*−*x*2−*y*2

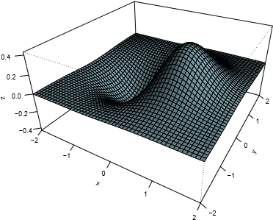
[x,y] = meshgrid(n,n); [x,y] = meshgrid(n,n) n <- seq(-2,2, length=40) z=x.\*exp(-x.^2-y.^2); z = x\*power(math.e,-x\*\*2-y\*\*2) z <- outer(n,n,f)



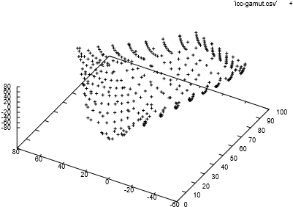
2 −2

Mesh plot mesh(z) persp(x,y,z,

theta=30, phi=30, expand=0.6, ticktype=’detailed’)



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Surface plot | | surf(x,y,z) or surfl(x,y,z) Octave:% no surfl() |  | persp(x,y,z, theta=30, phi=30, expand=0.6, col=’lightblue’, shade=0.75, ltheta=120, ticktype=’detailed’) |
| 7.4.3 | Scatter (cloud) plots |  |  |  |
| Desc. |  | matlab/Octave | Python | R |



**d scatter plot plot3(x,y,z,’k+’) cloud(z~x\*y)

7.5 Save plot to a graphics file

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | PostScript plot(1:10)  print -depsc2 foo.eps Octave:  gset output "foo.eps" gset terminal postscript eps plot(1:10) | | savefig(’foo.eps’) | postscript(file="foo.eps") plot(1:10) dev.off() | | PDF | | savefig(’foo.pdf’) | pdf(file=’foo.pdf’) | | SVG (vector graphics for www) | | savefig(’foo.svg’) | devSVG(file=’foo.svg’) | | PNG (raster graphics) print -dpng foo.png   1. Data analysis   8.1 Set membership operators | | savefig(’foo.png’) | png(filename = "Rplot%03d.png" | | Desc. | matlab/Octave | Python | R | | Create sets | a = [ 1 2 2 5 2 ]; b = [ 2 3 4 ]; | a = array([1,2,2,5,2]) b = array([2,3,4]) a = set([1,2,2,5,2]) b = set([2,3,4]) | a <- c(1,2,2,5,2) b <- c(2,3,4) | | Set unique | unique(a) | unique1d(a) unique(a) set(a) | unique(a) | | Set union | union(a,b) | union1d(a,b)  a.union(b) | union(a,b) | | Set intersection | intersect(a,b) | intersect1d(a)  a.intersection(b) | intersect(a,b) | | Set difference | setdiff(a,b) | setdiff1d(a,b)  a.difference(b) | setdiff(a,b) | | Set exclusion | setxor(a,b) | setxor1d(a,b)  a.symmetric\_difference(b) | setdiff(union(a,b),intersect(a,b)) | | True for set member | ismember(2,a) | 2 in a  setmember1d(2,a) contains(a,2) | is.element(2,a) or 2 %in% a | |  |
|  |

8.2 Statistics

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | Average mean(a) | | a.mean(axis=0) mean(a [,axis=0]) | apply(a,2,mean) | | Median median(a) | | median(a) or median(a [,axis=0]) | apply(a,2,median) | | Standard deviation std(a) | | a.std(axis=0) or std(a [,axis=0]) | apply(a,2,sd) | | Variance var(a) | | a.var(axis=0) or var(a) | apply(a,2,var) | | Correlation coefficient corr(x,y) | | correlate(x,y) or corrcoef(x,y) | cor(x,y) | | Covariance cov(x,y)  8.3 Interpolation and regression | | cov(x,y) | cov(x,y) | | Desc. | matlab/Octave | Python | R | | Straight line fit | z = polyval(polyfit(x,y,1),x) plot(x,y,’o’, x,z ,’-’) | (a,b) = polyfit(x,y,1) plot(x,y,’o’, x,a\*x+b,’-’) | z <- lm(y~x) plot(x,y) abline(z) | | Linear least squares *y* = *ax* + *b* | a = x\y | linalg.lstsq(x,y) | solve(a,b) | | Polynomial fit  8.4 Non-linear methods  8.4.1 Polynomials, root finding | polyfit(x,y,3) | polyfit(x,y,3) |  | | Desc. | matlab/Octave | Python | R | | Polynomial |  | poly() |  | | Find zeros of polynomial | roots([1 -1 -1]) | roots() | polyroot(c(1,-1,-1)) | | Find a zero near *x* = 1 | f = inline(’1/x - (x-1)’) fzero(f,1) |  |  | | Solve symbolic equations | solve(’1/x = x-1’) |  |  | | Evaluate polynomial  8.4.2 Differential equations | polyval([1 2 1 2],1:10) | polyval(array([1,2,1,2]),arange(1,11)) | | | Desc. | matlab/Octave | Python R | | | Discrete difference function and approxi-  mate derivative  Solve differential equations  8.5 Fourier analysis | diff(a) | diff(x, n=1, axis=0) | | | Desc. | matlab/Octave | Python R | | | Fast fourier transform | fft(a) | fft(a) or fft(a) | | | Inverse fourier transform | ifft(a) | ifft(a) or fft(a, inverse=TRUE) | | | Linear convolution |  | convolve(x,y) | | | 9 Symbolic algebra; calculus | | | Desc. matlab/Octave  Factorization factor() | | Python R | | |  |

# 10 Programming

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Script file extension | .m | .py | .R |  |
| Comment symbol (rest of line) | %  Octave:% or # | # | # |  |
| Import library functions | % must be in MATLABPATH  Octave:% must be in LOADPATH | from pylab import \* | library(RSvgDevice) |  |
| Eval  10.1 Loops | string=’a=234’; eval(string) | string="a=234" eval(string) | string <- "a <- 234"  eval(parse(text=string)) |  |
| Desc. | matlab/Octave | Python | R |  |
| for-statement | for i=1:5; disp(i); end | for i in range(1,6): print(i) | for(i in 1:5) print(i) |  |
| Multiline for statements  10.2 Conditionals | for i=1:5 disp(i) disp(i\*2) end | for i in range(1,6):  print(i) print(i\*2) | for(i in 1:5) {  print(i) print(i\*2)  } |  |
| Desc. | matlab/Octave | Python | R |  |
| if-statement | if 1>0 a=100; end | if 1>0: a=100 | if (1>0) a <- 100 |  |
| if-else-statement | if 1>0 a=100; else a=0; end |  |  |  |
| Ternary operator (if?true:false)  10.3 Debugging |  |  | ifelse(a>0,a,0) | *a >* 0?*a* : 0 |
| Desc. | matlab/Octave | Python | R |  |
| Most recent evaluated expression | ans |  | .Last.value |  |
| List variables loaded into memory | whos or who |  | objects() |  |
| Clear variable *x* from memory | clear x or clear [all] |  | rm(x) |  |
| Print | disp(a) | print a | print(a) |  |
| 10.4 Working directory and OS | |
| Desc. matlab/Octave | | Python | R |  |
| List files in directory dir or ls | | os.listdir(".") | list.files() or dir() |  |
| List script files in directory what | | grep.grep("\*.py") | list.files(pattern="\.r$") |  |
| Displays the current working directory pwd | | os.getcwd() | getwd() |  |
| Change working directory cd foo | | os.chdir(’foo’) | setwd(’foo’) |  |
| Invoke a System Command !notepad  Octave:system("notepad") | | os.system(’notepad’) os.popen(’notepad’) | system("notepad") |  |

**This document is still draft quality. Most shown **d plots are made using Matplotlib, and **d plots using R and Gnuplot, provided as examples only.

**Version numbers and download url for software used: Python **.**.**, http://www.python.org/; NumPy **.**.**, http://numeric.scipy.org/; Matplotlib **.**, http://matplotlib.sf.net/; IPython **.**.**, http://ipython.scipy.org/; R **.**.**, http://www.r-project.org/; Octave **.**.**, http://www.octave.org/; Scilab **.**, http://www.scilab.org/; Gnuplot **.**, http://www.gnuplot.info/.

**For referencing: Gundersen, Vidar Bronken. *MATLAB commands in numerical Python* (Oslo/Norway, **), available from: http://mathesaurus.sf.net/ **Contributions are appreciated: The best way to do this is to edit the xml and submit patches to our tracker or forums.

File IO (**[scipy.io](https://docs.scipy.org/doc/scipy/reference/io.html" \l "module-scipy.io" \o "scipy.io)**)

**See also**

[NumPy IO routines](https://www.numpy.org/devdocs/reference/routines.io.html)

MATLAB files

|  |  |
| --- | --- |
| [**loadmat**](https://docs.scipy.org/doc/scipy/reference/generated/scipy.io.loadmat.html#scipy.io.loadmat)(file\_name[, mdict, appendmat]) | Load MATLAB file. |
| [**savemat**](https://docs.scipy.org/doc/scipy/reference/generated/scipy.io.savemat.html#scipy.io.savemat)(file\_name, mdict[, appendmat, ...]) | Save a dictionary of names and arrays into a MATLAB-style .mat file. |
| [**whosmat**](https://docs.scipy.org/doc/scipy/reference/generated/scipy.io.whosmat.html#scipy.io.whosmat)(file\_name[, appendmat]) | List variables inside a MATLAB file. |

The basic functions

We’ll start by importing [**scipy.io**](https://docs.scipy.org/doc/scipy/reference/io.html#module-scipy.io) and calling it sio for convenience:

>>> **import** scipy.io **as** sio

If you are using IPython, try tab-completing on sio. Among the many options, you will find:

sio**.**loadmat

sio**.**savemat

sio**.**whosmat

These are the high-level functions you will most likely use when working with MATLAB files. You’ll also find:

sio**.**matlab

This is the package from which loadmat, savemat, and whosmat are imported. Within sio.matlab, you will find the mio module This module contains the machinery that loadmat and savemat use. From time to time you may find yourself re-using this machinery.

How do I start?

You may have a .mat file that you want to read into SciPy. Or, you want to pass some variables from SciPy / NumPy into MATLAB.

To save us using a MATLAB license, let’s start in [Octave](https://www.gnu.org/software/octave). Octave has MATLAB-compatible save and load functions. Start Octave (octave at the command line for me):

## Altro

Neither scipy.io.savemat, nor scipy.io.loadmat work for MATLAB arrays version 7.3. But the good part is that MATLAB version 7.3 files are hdf5 datasets. So they can be read using a number of tools, including [NumPy](http://en.wikipedia.org/wiki/NumPy).

For Python, you will need the h5py extension, which requires HDF5 on your system.

import numpy as np

import h5py

f = h5py.File('somefile.mat','r')

data = f.get('data/variable1')

data = np.array(data) # For converting to a NumPy array

There is a nice package called [mat4py](https://pypi.org/project/mat4py/) which can easily be installed using

pip install mat4py

It is straightforward to use (from the website):

**Load data from a MAT-file**

The function loadmat loads all variables stored in the MAT-file into a simple Python data structure, using only Python’s dict and list objects. Numeric and cell arrays are converted to row-ordered nested lists. Arrays are squeezed to eliminate arrays with only one element. The resulting data structure is composed of simple types that are compatible with the [JSON](http://en.wikipedia.org/wiki/JSON) format.

Example: Load a MAT-file into a Python data structure:

from mat4py import loadmat

data = loadmat('datafile.mat')

The variable data is a dict with the variables and values contained in the MAT-file.

**Save a Python data structure to a MAT-file**

Python data can be saved to a MAT-file, with the function savemat. Data has to be structured in the same way as for loadmat, i.e. it should be composed of simple data types, like dict, list, str, int, and float.

Example: Save a Python data structure to a MAT-file:

from mat4py import savemat

savemat('datafile.mat', data)

The parameter data shall be a dict with the variables.

[Call MATLAB from Python - MATLAB & Simulink - MathWorks Italia](https://it.mathworks.com/help/matlab/matlab-engine-for-python.html)

Can also use the hdf5storage library. official documentation [here](https://pythonhosted.org/hdf5storage/information.html#matlab-mat-v7-3-file-support) for details on matlab version support.

import hdf5storage

label\_file = "./LabelTrain.mat"

out = hdf5storage.loadmat(label\_file)

print(type(out)) # <class 'dict'>

from os.path import dirname, join as pjoin

import scipy.io as sio

data\_dir = pjoin(dirname(sio.\_\_file\_\_), 'matlab', 'tests', 'data')

mat\_fname = pjoin(data\_dir, 'testdouble\_7.4\_GLNX86.mat')

mat\_contents = sio.loadmat(mat\_fname)

You can use above code to read the default saved .mat file in Python.

scipy will work perfectly to load the .mat files. And we can use the get() function to convert it to a numpy array.

mat = scipy.io.loadmat('point05m\_matrix.mat')

x = mat.get("matrix")

print(type(x))

print(len(x))

plt.imshow(x, extent=[0,60,0,55], aspect='auto')

plt.show()

After struggling with this problem myself and trying other libraries (I have to say mat4py is a good one as well but with a few limitations) I have built this library ("[matdata2py](https://pypi.org/project/matdata2py/)") that can handle most variable types and most importantly for me the "string" type. The .mat file needs to be saved in the -V7.3 version. I hope this can be useful for the community.

## **Installation:**

pip install matdata2py

## **How to use this lib:**

import matdata2py as mtp

## **To load the Matlab data file:**

Variables\_output = mtp.loadmatfile(file\_Name, StructsExportLikeMatlab = True, ExportVar2PyEnv = False)

print(Variables\_output.keys()) # with ExportVar2PyEnv = False the variables are as elements of the Variables\_output dictionary.

with ExportVar2PyEnv = True you can see each variable separately as python variables with the same name as saved in the Mat file.

## **Flag descriptions**

StructsExportLikeMatlab = True/False structures are exported in dictionary format (False) or dot-based format similar to Matlab (True)

ExportVar2PyEnv = True/False export all variables in a single dictionary (True) or as separate individual variables into the python environment (False)

## **To Upload and Read mat files in python**

1. Install mat4py in python.On successful installation we get:
2. Successfully installed mat4py-0.5.0.
3. Importing loadmat from mat4py.
4. Save file actual location inside a variable.
5. Load mat file format to a data value using python  
   pip install mat4py  
   from mat4py import loadmat  
   boston = r"E:\Downloads\boston.mat" data = loadmat(boston, meta=False)

MOLTO INTERESSANTE:  
[How to load Matlab .mat files in Python | by Ashique Mahmood | Towards Data Science](https://towardsdatascience.com/how-to-load-matlab-mat-files-in-python-1f200e1287b5)

[How to open Matlab's "mat" files in Python | Import and plot from mat - YouTube](https://www.youtube.com/watch?v=Ev_dyZ1394s)

[How to read .mat file in python scipy | How to read MATLAB files in Python | Python Tutorial - YouTube](https://www.youtube.com/watch?v=KOByPHHFwuI)

[numpy - reading v 7.3 mat file in python - Stack Overflow](https://stackoverflow.com/questions/17316880/reading-v-7-3-mat-file-in-python)

# Varie

**cstruct2py**

[Pdfrw :: Anaconda.org](https://anaconda.org/conda-forge/pdfrw) legge e scrive pdf

# **Time**

[time](https://docs.python.org/3/library/time.html#module-time) — Time access and conversions

This module provides various time-related functions. For related functionality, see also the [datetime](https://docs.python.org/3/library/datetime.html#module-datetime) and [calendar](https://docs.python.org/3/library/calendar.html#module-calendar) modules.

Although this module is always available, not all functions are available on all platforms. Most of the functions defined in this module call platform C library functions with the same name. It may sometimes be helpful to consult the platform documentation, because the semantics of these functions varies among platforms.

An explanation of some terminology and conventions is in order.

* The *epoch* is the point where the time starts, the return value of time.gmtime(0). It is January 1, 1970, 00:00:00 (UTC) on all platforms.
* The term *seconds since the epoch* refers to the total number of elapsed seconds since the epoch, typically excluding [leap seconds](https://en.wikipedia.org/wiki/Leap_second). Leap seconds are excluded from this total on all POSIX-compliant platforms.
* The functions in this module may not handle dates and times before the [epoch](https://docs.python.org/3/library/time.html#epoch) or far in the future. The cut-off point in the future is determined by the C library; for 32-bit systems, it is typically in 2038.
* Function [strptime()](https://docs.python.org/3/library/time.html" \l "time.strptime" \o "time.strptime) can parse 2-digit years when given %y format code. When 2-digit years are parsed, they are converted according to the POSIX and ISO C standards: values 69–99 are mapped to 1969–1999, and values 0–68 are mapped to 2000–2068.
* UTC is Coordinated Universal Time (formerly known as Greenwich Mean Time, or GMT). The acronym UTC is not a mistake but a compromise between English and French.
* DST is Daylight Saving Time, an adjustment of the timezone by (usually) one hour during part of the year. DST rules are magic (determined by local law) and can change from year to year. The C library has a table containing the local rules (often it is read from a system file for flexibility) and is the only source of True Wisdom in this respect.
* The precision of the various real-time functions may be less than suggested by the units in which their value or argument is expressed. E.g. on most Unix systems, the clock “ticks” only 50 or 100 times a second.
* On the other hand, the precision of [time()](https://docs.python.org/3/library/time.html#time.time) and [sleep()](https://docs.python.org/3/library/time.html#time.sleep) is better than their Unix equivalents: times are expressed as floating point numbers, [time()](https://docs.python.org/3/library/time.html#time.time) returns the most accurate time available (using Unix gettimeofday() where available), and [sleep()](https://docs.python.org/3/library/time.html#time.sleep) will accept a time with a nonzero fraction (Unix select() is used to implement this, where available).
* The time value as returned by [gmtime()](https://docs.python.org/3/library/time.html" \l "time.gmtime" \o "time.gmtime), [localtime()](https://docs.python.org/3/library/time.html" \l "time.localtime" \o "time.localtime), and [strptime()](https://docs.python.org/3/library/time.html" \l "time.strptime" \o "time.strptime), and accepted by [asctime()](https://docs.python.org/3/library/time.html" \l "time.asctime" \o "time.asctime), [mktime()](https://docs.python.org/3/library/time.html" \l "time.mktime" \o "time.mktime) and [strftime()](https://docs.python.org/3/library/time.html" \l "time.strftime" \o "time.strftime), is a sequence of 9 integers. The return values of [gmtime()](https://docs.python.org/3/library/time.html" \l "time.gmtime" \o "time.gmtime), [localtime()](https://docs.python.org/3/library/time.html" \l "time.localtime" \o "time.localtime), and [strptime()](https://docs.python.org/3/library/time.html" \l "time.strptime" \o "time.strptime) also offer attribute names for individual fields.

See [struct\_time](https://docs.python.org/3/library/time.html" \l "time.struct_time" \o "time.struct_time) for a description of these objects.

*Changed in version 3.3:*The [struct\_time](https://docs.python.org/3/library/time.html" \l "time.struct_time" \o "time.struct_time) type was extended to provide the tm\_gmtoff and tm\_zone attributes when platform supports corresponding struct tm members.

*Changed in version 3.6:*The [struct\_time](https://docs.python.org/3/library/time.html" \l "time.struct_time" \o "time.struct_time) attributes tm\_gmtoff and tm\_zone are now available on all platforms.

* Use the following functions to convert between time representations:

| **From** | **To** | **Use** |
| --- | --- | --- |
| seconds since the epoch | [struct\_time](https://docs.python.org/3/library/time.html#time.struct_time) in UTC | [gmtime()](https://docs.python.org/3/library/time.html#time.gmtime) |
| seconds since the epoch | [struct\_time](https://docs.python.org/3/library/time.html#time.struct_time) in local time | [localtime()](https://docs.python.org/3/library/time.html#time.localtime) |
| [struct\_time](https://docs.python.org/3/library/time.html#time.struct_time) in UTC | seconds since the epoch | [calendar.timegm()](https://docs.python.org/3/library/calendar.html#calendar.timegm) |
| [struct\_time](https://docs.python.org/3/library/time.html#time.struct_time) in local time | seconds since the epoch | [mktime()](https://docs.python.org/3/library/time.html#time.mktime) |

Functions

time.**asctime**([*t*])

Convert a tuple or [struct\_time](https://docs.python.org/3/library/time.html" \l "time.struct_time" \o "time.struct_time) representing a time as returned by [gmtime()](https://docs.python.org/3/library/time.html" \l "time.gmtime" \o "time.gmtime) or [localtime()](https://docs.python.org/3/library/time.html" \l "time.localtime" \o "time.localtime) to a string of the following form: 'Sun Jun 20 23:21:05 1993'. The day field is two characters long and is space padded if the day is a single digit, e.g.: 'Wed Jun  9 04:26:40 1993'.

If *t* is not provided, the current time as returned by [localtime()](https://docs.python.org/3/library/time.html" \l "time.localtime" \o "time.localtime) is used. Locale information is not used by [asctime()](https://docs.python.org/3/library/time.html" \l "time.asctime" \o "time.asctime).

**Note**

Unlike the C function of the same name, [asctime()](https://docs.python.org/3/library/time.html" \l "time.asctime" \o "time.asctime) does not add a trailing newline.

time.**pthread\_getcpuclockid**(*thread\_id*)

Return the *clk\_id* of the thread-specific CPU-time clock for the specified *thread\_id*.

Use [threading.get\_ident()](https://docs.python.org/3/library/threading.html" \l "threading.get_ident" \o "threading.get_ident) or the [ident](https://docs.python.org/3/library/threading.html#threading.Thread.ident) attribute of [threading.Thread](https://docs.python.org/3/library/threading.html" \l "threading.Thread" \o "threading.Thread) objects to get a suitable value for *thread\_id*.

**Warning**

Passing an invalid or expired *thread\_id* may result in undefined behavior, such as segmentation fault.

[Availability](https://docs.python.org/3/library/intro.html#availability): Unix

See the man page for *[pthread\_getcpuclockid(3)](https://manpages.debian.org/pthread_getcpuclockid(3))* for further information.

*New in version 3.7.*

time.**clock\_getres**(*clk\_id*)

Return the resolution (precision) of the specified clock *clk\_id*. Refer to [Clock ID Constants](https://docs.python.org/3/library/time.html#time-clock-id-constants) for a list of accepted values for *clk\_id*.

[Availability](https://docs.python.org/3/library/intro.html#availability): Unix.

*New in version 3.3.*

time.**clock\_gettime**(*clk\_id*) → [float](https://docs.python.org/3/library/functions.html#float)

Return the time of the specified clock *clk\_id*. Refer to [Clock ID Constants](https://docs.python.org/3/library/time.html#time-clock-id-constants) for a list of accepted values for *clk\_id*.

Use [clock\_gettime\_ns()](https://docs.python.org/3/library/time.html" \l "time.clock_gettime_ns" \o "time.clock_gettime_ns) to avoid the precision loss caused by the [float](https://docs.python.org/3/library/functions.html#float) type.

[Availability](https://docs.python.org/3/library/intro.html#availability): Unix.

*New in version 3.3.*

time.**clock\_gettime\_ns**(*clk\_id*) → [int](https://docs.python.org/3/library/functions.html#int)

Similar to [clock\_gettime()](https://docs.python.org/3/library/time.html" \l "time.clock_gettime" \o "time.clock_gettime) but return time as nanoseconds.

[Availability](https://docs.python.org/3/library/intro.html#availability): Unix.

*New in version 3.7.*

time.**clock\_settime**(*clk\_id*, *time:*[*float*](https://docs.python.org/3/library/functions.html#float))

Set the time of the specified clock *clk\_id*. Currently, [CLOCK\_REALTIME](https://docs.python.org/3/library/time.html#time.CLOCK_REALTIME) is the only accepted value for *clk\_id*.

Use [clock\_settime\_ns()](https://docs.python.org/3/library/time.html" \l "time.clock_settime_ns" \o "time.clock_settime_ns) to avoid the precision loss caused by the [float](https://docs.python.org/3/library/functions.html#float) type.

[Availability](https://docs.python.org/3/library/intro.html#availability): Unix.

*New in version 3.3.*

time.**clock\_settime\_ns**(*clk\_id*, *time:*[*int*](https://docs.python.org/3/library/functions.html#int))

Similar to [clock\_settime()](https://docs.python.org/3/library/time.html" \l "time.clock_settime" \o "time.clock_settime) but set time with nanoseconds.

[Availability](https://docs.python.org/3/library/intro.html#availability): Unix.

*New in version 3.7.*

time.**ctime**([*secs*])

Convert a time expressed in seconds since the [epoch](https://docs.python.org/3/library/time.html#epoch) to a string of a form: 'Sun Jun 20 23:21:05 1993' representing local time. The day field is two characters long and is space padded if the day is a single digit, e.g.: 'Wed Jun  9 04:26:40 1993'.

If *secs* is not provided or [None](https://docs.python.org/3/library/constants.html#None), the current time as returned by [time()](https://docs.python.org/3/library/time.html#time.time) is used. ctime(secs) is equivalent to asctime(localtime(secs)). Locale information is not used by [ctime()](https://docs.python.org/3/library/time.html" \l "time.ctime" \o "time.ctime).

time.**get\_clock\_info**(*name*)

Get information on the specified clock as a namespace object. Supported clock names and the corresponding functions to read their value are:

* 'monotonic': [time.monotonic()](https://docs.python.org/3/library/time.html" \l "time.monotonic" \o "time.monotonic)
* 'perf\_counter': [time.perf\_counter()](https://docs.python.org/3/library/time.html" \l "time.perf_counter" \o "time.perf_counter)
* 'process\_time': [time.process\_time()](https://docs.python.org/3/library/time.html" \l "time.process_time" \o "time.process_time)
* 'thread\_time': [time.thread\_time()](https://docs.python.org/3/library/time.html" \l "time.thread_time" \o "time.thread_time)
* 'time': [time.time()](https://docs.python.org/3/library/time.html" \l "time.time" \o "time.time)

The result has the following attributes:

* *adjustable*: True if the clock can be changed automatically (e.g. by a NTP daemon) or manually by the system administrator, False otherwise
* *implementation*: The name of the underlying C function used to get the clock value. Refer to [Clock ID Constants](https://docs.python.org/3/library/time.html#time-clock-id-constants) for possible values.
* *monotonic*: True if the clock cannot go backward, False otherwise
* *resolution*: The resolution of the clock in seconds ([float](https://docs.python.org/3/library/functions.html#float))

*New in version 3.3.*

time.**gmtime**([*secs*])

Convert a time expressed in seconds since the [epoch](https://docs.python.org/3/library/time.html#epoch) to a [struct\_time](https://docs.python.org/3/library/time.html" \l "time.struct_time" \o "time.struct_time) in UTC in which the dst flag is always zero. If *secs* is not provided or [None](https://docs.python.org/3/library/constants.html#None), the current time as returned by [time()](https://docs.python.org/3/library/time.html#time.time) is used. Fractions of a second are ignored. See above for a description of the [struct\_time](https://docs.python.org/3/library/time.html" \l "time.struct_time" \o "time.struct_time) object. See [calendar.timegm()](https://docs.python.org/3/library/calendar.html" \l "calendar.timegm" \o "calendar.timegm) for the inverse of this function.

time.**localtime**([*secs*])

Like [gmtime()](https://docs.python.org/3/library/time.html" \l "time.gmtime" \o "time.gmtime) but converts to local time. If *secs* is not provided or [None](https://docs.python.org/3/library/constants.html#None), the current time as returned by [time()](https://docs.python.org/3/library/time.html#time.time) is used. The dst flag is set to 1 when DST applies to the given time.

[localtime()](https://docs.python.org/3/library/time.html#time.localtime) may raise [OverflowError](https://docs.python.org/3/library/exceptions.html" \l "OverflowError" \o "OverflowError), if the timestamp is outside the range of values supported by the platform C localtime() or gmtime() functions, and [OSError](https://docs.python.org/3/library/exceptions.html" \l "OSError" \o "OSError) on localtime() or gmtime() failure. It’s common for this to be restricted to years between 1970 and 2038.

time.**mktime**(*t*)

This is the inverse function of [localtime()](https://docs.python.org/3/library/time.html" \l "time.localtime" \o "time.localtime). Its argument is the [struct\_time](https://docs.python.org/3/library/time.html" \l "time.struct_time" \o "time.struct_time) or full 9-tuple (since the dst flag is needed; use -1 as the dst flag if it is unknown) which expresses the time in *local* time, not UTC. It returns a floating point number, for compatibility with [time()](https://docs.python.org/3/library/time.html#time.time). If the input value cannot be represented as a valid time, either [OverflowError](https://docs.python.org/3/library/exceptions.html" \l "OverflowError" \o "OverflowError) or [ValueError](https://docs.python.org/3/library/exceptions.html" \l "ValueError" \o "ValueError) will be raised (which depends on whether the invalid value is caught by Python or the underlying C libraries). The earliest date for which it can generate a time is platform-dependent.

time.**monotonic**() → [float](https://docs.python.org/3/library/functions.html#float)

Return the value (in fractional seconds) of a monotonic clock, i.e. a clock that cannot go backwards. The clock is not affected by system clock updates. The reference point of the returned value is undefined, so that only the difference between the results of two calls is valid.

Use [monotonic\_ns()](https://docs.python.org/3/library/time.html" \l "time.monotonic_ns" \o "time.monotonic_ns) to avoid the precision loss caused by the [float](https://docs.python.org/3/library/functions.html#float) type.

*New in version 3.3.*

*Changed in version 3.5:*The function is now always available and always system-wide.

*Changed in version 3.10:*On macOS, the function is now system-wide.

time.**monotonic\_ns**() → [int](https://docs.python.org/3/library/functions.html#int)

Similar to [monotonic()](https://docs.python.org/3/library/time.html#time.monotonic), but return time as nanoseconds.

*New in version 3.7.*

time.**perf\_counter**() → [float](https://docs.python.org/3/library/functions.html#float)

Return the value (in fractional seconds) of a performance counter, i.e. a clock with the highest available resolution to measure a short duration. It does include time elapsed during sleep and is system-wide. The reference point of the returned value is undefined, so that only the difference between the results of two calls is valid.

Use [perf\_counter\_ns()](https://docs.python.org/3/library/time.html" \l "time.perf_counter_ns" \o "time.perf_counter_ns) to avoid the precision loss caused by the [float](https://docs.python.org/3/library/functions.html#float) type.

*New in version 3.3.*

*Changed in version 3.10:*On Windows, the function is now system-wide.

time.**perf\_counter\_ns**() → [int](https://docs.python.org/3/library/functions.html#int)

Similar to [perf\_counter()](https://docs.python.org/3/library/time.html" \l "time.perf_counter" \o "time.perf_counter), but return time as nanoseconds.

*New in version 3.7.*

time.**process\_time**() → [float](https://docs.python.org/3/library/functions.html#float)

Return the value (in fractional seconds) of the sum of the system and user CPU time of the current process. It does not include time elapsed during sleep. It is process-wide by definition. The reference point of the returned value is undefined, so that only the difference between the results of two calls is valid.

Use [process\_time\_ns()](https://docs.python.org/3/library/time.html" \l "time.process_time_ns" \o "time.process_time_ns) to avoid the precision loss caused by the [float](https://docs.python.org/3/library/functions.html#float) type.

*New in version 3.3.*

time.**process\_time\_ns**() → [int](https://docs.python.org/3/library/functions.html#int)

Similar to [process\_time()](https://docs.python.org/3/library/time.html" \l "time.process_time" \o "time.process_time) but return time as nanoseconds.

*New in version 3.7.*

time.**sleep**(*secs*)

Suspend execution of the calling thread for the given number of seconds. The argument may be a floating point number to indicate a more precise sleep time.

If the sleep is interrupted by a signal and no exception is raised by the signal handler, the sleep is restarted with a recomputed timeout.

The suspension time may be longer than requested by an arbitrary amount, because of the scheduling of other activity in the system.

On Windows, if *secs* is zero, the thread relinquishes the remainder of its time slice to any other thread that is ready to run. If there are no other threads ready to run, the function returns immediately, and the thread continues execution. On Windows 8.1 and newer the implementation uses a [high-resolution timer](https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/high-resolution-timers) which provides resolution of 100 nanoseconds. If *secs* is zero, Sleep(0) is used.

Unix implementation:

* Use clock\_nanosleep() if available (resolution: 1 nanosecond);
* Or use nanosleep() if available (resolution: 1 nanosecond);
* Or use select() (resolution: 1 microsecond).

*Changed in version 3.11:*On Unix, the clock\_nanosleep() and nanosleep() functions are now used if available. On Windows, a waitable timer is now used.

*Changed in version 3.5:*The function now sleeps at least *secs* even if the sleep is interrupted by a signal, except if the signal handler raises an exception (see [**PEP 475**](https://peps.python.org/pep-0475/) for the rationale).

time.**strftime**(*format*[, *t*])

Convert a tuple or [struct\_time](https://docs.python.org/3/library/time.html" \l "time.struct_time" \o "time.struct_time) representing a time as returned by [gmtime()](https://docs.python.org/3/library/time.html" \l "time.gmtime" \o "time.gmtime) or [localtime()](https://docs.python.org/3/library/time.html" \l "time.localtime" \o "time.localtime) to a string as specified by the *format* argument. If *t* is not provided, the current time as returned by [localtime()](https://docs.python.org/3/library/time.html" \l "time.localtime" \o "time.localtime) is used. *format* must be a string. [ValueError](https://docs.python.org/3/library/exceptions.html" \l "ValueError" \o "ValueError) is raised if any field in *t* is outside of the allowed range.

0 is a legal argument for any position in the time tuple; if it is normally illegal the value is forced to a correct one.

The following directives can be embedded in the *format* string. They are shown without the optional field width and precision specification, and are replaced by the indicated characters in the [strftime()](https://docs.python.org/3/library/time.html" \l "time.strftime" \o "time.strftime) result:

| **Directive** | **Meaning** | **Notes** |
| --- | --- | --- |
| %a | Locale’s abbreviated weekday name. |  |
| %A | Locale’s full weekday name. |  |
| %b | Locale’s abbreviated month name. |  |
| %B | Locale’s full month name. |  |
| %c | Locale’s appropriate date and time representation. |  |
| %d | Day of the month as a decimal number [01,31]. |  |
| %H | Hour (24-hour clock) as a decimal number [00,23]. |  |
| %I | Hour (12-hour clock) as a decimal number [01,12]. |  |
| %j | Day of the year as a decimal number [001,366]. |  |
| %m | Month as a decimal number [01,12]. |  |
| %M | Minute as a decimal number [00,59]. |  |
| %p | Locale’s equivalent of either AM or PM. | (1) |
| %S | Second as a decimal number [00,61]. | (2) |
| %U | Week number of the year (Sunday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Sunday are considered to be in week 0. | (3) |
| %w | Weekday as a decimal number [0(Sunday),6]. |  |
| %W | Week number of the year (Monday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Monday are considered to be in week 0. | (3) |
| %x | Locale’s appropriate date representation. |  |
| %X | Locale’s appropriate time representation. |  |
| %y | Year without century as a decimal number [00,99]. |  |
| %Y | Year with century as a decimal number. |  |
| %z | Time zone offset indicating a positive or negative time difference from UTC/GMT of the form +HHMM or -HHMM, where H represents decimal hour digits and M represents decimal minute digits [-23:59, +23:59]. [1](https://docs.python.org/3/library/time.html#id4) |  |
| %Z | Time zone name (no characters if no time zone exists). Deprecated. [1](https://docs.python.org/3/library/time.html#id4) |  |
| %% | A literal '%' character. |  |

Notes:

1. When used with the [strptime()](https://docs.python.org/3/library/time.html" \l "time.strptime" \o "time.strptime) function, the %p directive only affects the output hour field if the %I directive is used to parse the hour.
2. The range really is 0 to 61; value 60 is valid in timestamps representing [leap seconds](https://en.wikipedia.org/wiki/Leap_second) and value 61 is supported for historical reasons.
3. When used with the [strptime()](https://docs.python.org/3/library/time.html" \l "time.strptime" \o "time.strptime) function, %U and %W are only used in calculations when the day of the week and the year are specified.

Here is an example, a format for dates compatible with that specified in the [**RFC 2822**](https://datatracker.ietf.org/doc/html/rfc2822.html) Internet email standard. [1](https://docs.python.org/3/library/time.html#id4)

>>>

**>>> from** **time** **import** gmtime, strftime

**>>>** strftime("**%a**, **%d** %b %Y %H:%M:%S +0000", gmtime())

'Thu, 28 Jun 2001 14:17:15 +0000'

Additional directives may be supported on certain platforms, but only the ones listed here have a meaning standardized by ANSI C. To see the full set of format codes supported on your platform, consult the *[strftime(3)](https://manpages.debian.org/strftime(3))* documentation.

On some platforms, an optional field width and precision specification can immediately follow the initial '%' of a directive in the following order; this is also not portable. The field width is normally 2 except for %j where it is 3.

time.**strptime**(*string*[, *format*])

Parse a string representing a time according to a format. The return value is a [struct\_time](https://docs.python.org/3/library/time.html" \l "time.struct_time" \o "time.struct_time) as returned by [gmtime()](https://docs.python.org/3/library/time.html" \l "time.gmtime" \o "time.gmtime) or [localtime()](https://docs.python.org/3/library/time.html" \l "time.localtime" \o "time.localtime).

The *format* parameter uses the same directives as those used by [strftime()](https://docs.python.org/3/library/time.html" \l "time.strftime" \o "time.strftime); it defaults to "%a %b %d %H:%M:%S %Y" which matches the formatting returned by [ctime()](https://docs.python.org/3/library/time.html" \l "time.ctime" \o "time.ctime). If *string* cannot be parsed according to *format*, or if it has excess data after parsing, [ValueError](https://docs.python.org/3/library/exceptions.html" \l "ValueError" \o "ValueError) is raised. The default values used to fill in any missing data when more accurate values cannot be inferred are (1900, 1, 1, 0, 0, 0, 0, 1, -1). Both *string* and *format* must be strings.

For example:

>>>

**>>> import** **time**

**>>>** time.strptime("30 Nov 00", "**%d** %b %y")

time.struct\_time(tm\_year=2000, tm\_mon=11, tm\_mday=30, tm\_hour=0, tm\_min=0,

tm\_sec=0, tm\_wday=3, tm\_yday=335, tm\_isdst=-1)

Support for the %Z directive is based on the values contained in tzname and whether daylight is true. Because of this, it is platform-specific except for recognizing UTC and GMT which are always known (and are considered to be non-daylight savings timezones).

Only the directives specified in the documentation are supported. Because strftime() is implemented per platform it can sometimes offer more directives than those listed. But strptime() is independent of any platform and thus does not necessarily support all directives available that are not documented as supported.

*class*time.**struct\_time**

The type of the time value sequence returned by [gmtime()](https://docs.python.org/3/library/time.html" \l "time.gmtime" \o "time.gmtime), [localtime()](https://docs.python.org/3/library/time.html" \l "time.localtime" \o "time.localtime), and [strptime()](https://docs.python.org/3/library/time.html" \l "time.strptime" \o "time.strptime). It is an object with a [named tuple](https://docs.python.org/3/glossary.html#term-named-tuple) interface: values can be accessed by index and by attribute name. The following values are present:

| **Index** | **Attribute** | **Values** |
| --- | --- | --- |
| 0 | tm\_year | (for example, 1993) |
| 1 | tm\_mon | range [1, 12] |
| 2 | tm\_mday | range [1, 31] |
| 3 | tm\_hour | range [0, 23] |
| 4 | tm\_min | range [0, 59] |
| 5 | tm\_sec | range [0, 61]; see **(2)** in [strftime()](https://docs.python.org/3/library/time.html" \l "time.strftime" \o "time.strftime) description |
| 6 | tm\_wday | range [0, 6], Monday is 0 |
| 7 | tm\_yday | range [1, 366] |
| 8 | tm\_isdst | 0, 1 or -1; see below |
| N/A | tm\_zone | abbreviation of timezone name |
| N/A | tm\_gmtoff | offset east of UTC in seconds |

Note that unlike the C structure, the month value is a range of [1, 12], not [0, 11].

In calls to [mktime()](https://docs.python.org/3/library/time.html" \l "time.mktime" \o "time.mktime), tm\_isdst may be set to 1 when daylight savings time is in effect, and 0 when it is not. A value of -1 indicates that this is not known, and will usually result in the correct state being filled in.

When a tuple with an incorrect length is passed to a function expecting a [struct\_time](https://docs.python.org/3/library/time.html" \l "time.struct_time" \o "time.struct_time), or having elements of the wrong type, a [TypeError](https://docs.python.org/3/library/exceptions.html" \l "TypeError" \o "TypeError) is raised.

time.**time**() → [float](https://docs.python.org/3/library/functions.html#float)

Return the time in seconds since the [epoch](https://docs.python.org/3/library/time.html#epoch) as a floating point number. The handling of [leap seconds](https://en.wikipedia.org/wiki/Leap_second) is platform dependent. On Windows and most Unix systems, the leap seconds are not counted towards the time in seconds since the [epoch](https://docs.python.org/3/library/time.html#epoch). This is commonly referred to as [Unix time](https://en.wikipedia.org/wiki/Unix_time).

Note that even though the time is always returned as a floating point number, not all systems provide time with a better precision than 1 second. While this function normally returns non-decreasing values, it can return a lower value than a previous call if the system clock has been set back between the two calls.

The number returned by [time()](https://docs.python.org/3/library/time.html#time.time) may be converted into a more common time format (i.e. year, month, day, hour, etc…) in UTC by passing it to [gmtime()](https://docs.python.org/3/library/time.html" \l "time.gmtime" \o "time.gmtime) function or in local time by passing it to the [localtime()](https://docs.python.org/3/library/time.html" \l "time.localtime" \o "time.localtime) function. In both cases a [struct\_time](https://docs.python.org/3/library/time.html" \l "time.struct_time" \o "time.struct_time) object is returned, from which the components of the calendar date may be accessed as attributes.

Use [time\_ns()](https://docs.python.org/3/library/time.html" \l "time.time_ns" \o "time.time_ns) to avoid the precision loss caused by the [float](https://docs.python.org/3/library/functions.html#float) type.

time.**time\_ns**() → [int](https://docs.python.org/3/library/functions.html#int)

Similar to [time()](https://docs.python.org/3/library/time.html#time.time) but returns time as an integer number of nanoseconds since the [epoch](https://docs.python.org/3/library/time.html#epoch).

The TZ environment variable should contain no whitespace.

The standard format of the TZ environment variable is (whitespace added for clarity):

std offset [dst [offset [,start[/time], end[/time]]]]

Where the components are:

std and dst

Three or more alphanumerics giving the timezone abbreviations. These will be propagated into time.tzname

offset

The offset has the form: ± hh[:mm[:ss]]. This indicates the value added the local time to arrive at UTC. If preceded by a ‘-’, the timezone is east of the Prime Meridian; otherwise, it is west. If no offset follows dst, summer time is assumed to be one hour ahead of standard time.

start[/time], end[/time]

Indicates when to change to and back from DST. The format of the start and end dates are one of the following:

J*n*

The Julian day *n* (1 <= *n* <= 365). Leap days are not counted, so in all years February 28 is day 59 and March 1 is day 60.

*n*

The zero-based Julian day (0 <= *n* <= 365). Leap days are counted, and it is possible to refer to February 29.

M*m*.*n*.*d*

The *d*’th day (0 <= *d* <= 6) of week *n* of month *m* of the year (1 <= *n* <= 5, 1 <= *m* <= 12, where week 5 means “the last *d* day in month *m*” which may occur in either the fourth or the fifth week). Week 1 is the first week in which the *d*’th day occurs. Day zero is a Sunday.

time has the same format as offset except that no leading sign (‘-’ or ‘+’) is allowed. The default, if time is not given, is 02:00:00.

>>>

**>>>** os.environ['TZ'] = 'EST+05EDT,M4.1.0,M10.5.0'

**>>>** time.tzset()

**>>>** time.strftime('**%X** **%x** %Z')

'02:07:36 05/08/03 EDT'

**>>>** os.environ['TZ'] = 'AEST-10AEDT-11,M10.5.0,M3.5.0'

**>>>** time.tzset()

**>>>** time.strftime('**%X** **%x** %Z')

'16:08:12 05/08/03 AEST'

On many Unix systems (including \*BSD, Linux, Solaris, and Darwin), it is more convenient to use the system’s zoneinfo (*[tzfile(5)](https://manpages.debian.org/tzfile(5))*) database to specify the timezone rules. To do this, set the TZ environment variable to the path of the required timezone datafile, relative to the root of the systems ‘zoneinfo’ timezone database, usually located at /usr/share/zoneinfo. For example, 'US/Eastern', 'Australia/Melbourne', 'Egypt' or 'Europe/Amsterdam'

>>>

**>>>** os.environ['TZ'] = 'US/Eastern'

**>>>** time.tzset()

**>>>** time.tzname

('EST', 'EDT')

**>>>** os.environ['TZ'] = 'Egypt'

**>>>** time.tzset()

**>>>** time.tzname

('EET', 'EEST')

## Astropy

[Astropy Documentation — Astropy v5.1.1](https://docs.astropy.org/en/stable/index.html)

1. References: Hankin, Robin. *R for Octave users* (**), available from http://cran.r-project.org/doc/contrib/R-and-octave-**.txt (accessed **.**.**); Martelli, Alex. *Python in a Nutshell* (O’Reilly, **);

   Oliphant, Travis. *Guide to NumPy* (Trelgol, **); Hunter, John. *The Matplotlib User’s Guide* (**), available from http://matplotlib.sf.net/ (accessed **.**.**); Langtangen, Hans Petter. *Python*

   *Scripting for Computational Science* (Springer, **); Ascher et al.: *Numeric Python manual* (**), available from http://numeric.scipy.org/numpy.pdf (accessed **.**.**); Moler, Cleve. *Numerical Computing with MATLAB* (MathWorks, **), available from http://www.mathworks.com/moler/ (accessed **.**.**); Eaton, John W. *Octave Quick Reference* (**); Merrit, Ethan. *Demo scripts for gnuplot version 4.0* (**), available from http://gnuplot.sourceforge.net/demo/ (accessed **.**.**); Woo, Alex. *Gnuplot Quick Reference* (**), available from http://www.gnuplot.info/docs/gpcard.pdf (accessed **.**.**); Venables & Smith: *An Introduction to R* (**), available from http://cran.r-project.org/doc/manuals/R-intro.pdf (accessed **.**.**); Short, Tom. *R reference card* (**), available from http://www.rpad.org/Rpad/R-refcard.pdf (accessed **.**.**). [↑](#footnote-ref-1)