Lecture 8 Matplotlib and Image Processing

Matplotlib

Matplotlib (https://matplotlib.org/) serves as the package to produce publication-quality figures in Python, and provides interface closely resembling to matlab (https://matplotlib.org/tutorials/introductory/pyplot.html).

In [16]:

```
import matplotlib as mpl # import whole package
import matplotlib.pyplot as plt # or just import submodule pylot, providing matlab-li
ke functions
# these are "standard shorthands", though some poeple use other nicknames
```

In [19]:
```
dir(mpl)
```

```
Out[19]: ['ExecutableNotFoundError',
          'LooseVersion',
          'MatplotlibDeprecationWarning',
          'MutableMapping',
          'Parameter',
          'Path',
          'RcParams',
          'URL_REGEX',
          '_DATA_DOC_APPENDIX',
          '_DATA_DOC_TITLE',
          '_ExecInfo',
          '__bibtex__',
          '__builtins__',
          '__cached__',
          '__doc__',
          '__file__',
          '__loader__',
          '__name__',
          '__package__',
          '__path__',
          '__spec__',
          '__version__',
          '_add_data_doc',
          '_all_deprecated',
          '_animation_data',
          '_check_versions',
          '_cm',
          '_cm_listed',
          '_color_data',
          '_constrained_layout',
          '_deprecated_ignore_map',
          '_deprecated_map',
          '_deprecated_remain_as_none',
          '_ensure_handler',
          '_get_config_or_cache_dir',
          '_get_data_path',
          '_get_executable_info',
          '_get_ssl_context',
          '_get_xdg_cache_dir',
          '_get_xdg_config_dir',
          '_image',
          '_init_tests',
          '_label_from_arg',
          '_layoutbox',
          '_log',
          '_logged_cached',
          '_mathtext_data',
          '_open_file_or_url',
          '_path',
          '_preprocess_data',
          '_pylab_helpers',
          '_rc_params_in_file',
          '_replacer',
          '_text_layout',
          '_version',
          'afm',
          'animation',
          'artist',
          'atexit',
          'axes',
          'axis',
          'backend_bases',
          'backend_managers',
          'backend_tools',
          'backends',
          'bezier',
          'blocking_input',
          'category',
          'cbook',
          'checkdep_ps_distiller',
```

```
 'checkdep_usetex',
 'cm',
 'collections',
 'colorbar',
 'colors',
 'compare_versions',
 'container',
 'contextlib',
 'contour',
 'cycler',
 'dates',
 'defaultParams',
 'default_test_modules',
 'docstring',
 'dviread',
 'figure',
 'font_manager',
 'fontconfig_pattern',
 'ft2font',
 'functools',
 'get_backend',
 'get_cachedir',
 'get_configdir',
 'get_data_path',
 'get_home',
 'gridspec',
 'image',
 'importlib',
 'inspect',
 'interactive',
 'is_interactive',
 'is_url',
 'legend',
 'legend_handler',
 'lines',
 'locale',
 'logging',
 'markers',
 'mathtext',
 'matplotlib_fname',
 'mlab',
 'mplDeprecation',
 'namedtuple',
 'numpy',
 'offsetbox',
 'os',
 'patches',
 'path',
 'pprint',
 'projections',
 'pyplot',
 'quiver',
 'rc',
 'rcParams',
 'rcParamsDefault',
 'rcParamsOrig',
 'rc_context',
 'rc_file',
 'rc_file_defaults',
 'rc_params',
 'rc_params_from_file',
 'rcdefaults',
 'rcsetup',
 're',
 'sanitize_sequence',
 'scale',
 'set_loglevel',
 'shutil',
 'spines',
 'stackplot',
```

```
 'streamplot',
 'style',
 'subprocess',
 'sys',
 'table',
 'tempfile',
 'test',
 'texmanager',
 'text',
 'textpath',
 'ticker',
 'tight_bbox',
 'tight_layout',
 'transforms',
 'tri',
 'units',
 'use',
 'validate_backend',
 'warnings',
 'widgets']
```

In [20]:

```
dir(plt)
```

```
Out[20]: ['Annotation',
 'Arrow',
 'Artist',
 'AutoLocator',
 'Axes',
 'Button',
 'Circle',
 'Figure',
 'FigureCanvasBase',
 'FixedFormatter',
 'FixedLocator',
 'FormatStrFormatter',
 'Formatter',
 'FuncFormatter',
 'GridSpec',
 'IndexLocator',
 'Line2D',
 'LinearLocator',
 'Locator',
 'LogFormatter',
 'LogFormatterExponent',
 'LogFormatterMathtext',
 'LogLocator',
 'MaxNLocator',
 'MouseButton',
 'MultipleLocator',
 'Normalize',
 'NullFormatter',
 'NullLocator',
 'Number',
 'PolarAxes',
 'Polygon',
 'Rectangle',
 'ScalarFormatter',
 'Slider',
 'Subplot',
 'SubplotTool',
 'Text',
 'TickHelper',
 'Widget',
 '_INSTALL_FIG_OBSERVER',
 '_IP_REGISTERED',
 '__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 '_auto_draw_if_interactive',
 '_backend_mod',
 '_code_objs',
 '_copy_docstring_and_deprecators',
 '_get_required_interactive_framework',
 '_interactive_bk',
 '_log',
 '_pylab_helpers',
 '_setup_pyplot_info_docstrings',
 '_warn_if_gui_out_of_main_thread',
 '_xkcd',
 'acorr',
 'angle_spectrum',
 'annotate',
 'arrow',
 'autoscale',
 'autumn',
 'axes',
 'axhline',
 'axhspan',
```

```
'axis',
'axline',
'axvline',
'axvspan',
'bar',
'barbs',
'barh',
'bone',
'box',
'boxplot',
'broken_barh',
'cbook',
'cla',
'clabel',
'clf',
'clim',
'close',
'cm',
'cohere',
'colorbar',
'colormaps',
'connect',
'contour',
'contourf',
'cool',
'copper',
'csd',
'cycler',
'delaxes',
'disconnect',
'docstring',
'draw',
'draw_all',
'draw_if_interactive',
'errorbar',
'eventplot',
'figaspect',
'figimage',
'figlegend',
'fignum_exists',
'figtext',
'figure',
'fill',
'fill_between',
'fill_betweenx',
'findobj',
'flag',
'functools',
'gca',
'gcf',
'gci',
'get',
'get_backend',
'get_cmap',
'get_current_fig_manager',
'get_figlabels',
'get_fignums',
'get_plot_commands',
'get_scale_names',
'getp',
'ginput',
'gray',
'grid',
'hexbin',
'hist',
'hist2d',
'hlines',
'hot',
'hsv',
'importlib',
```

```
 'imread',
 'imsave',
 'imshow',
 'inferno',
 'inspect',
 'install_repl_displayhook',
 'interactive',
 'ioff',
 'ion',
 'isinteractive',
 'jet',
 'legend',
 'locator_params',
 'logging',
 'loglog',
 'magma',
 'magnitude_spectrum',
 'margins',
 'matplotlib',
 'matshow',
 'minorticks_off',
 'minorticks_on',
 'mlab',
 'new_figure_manager',
 'nipy_spectral',
 'np',
 'pause',
 'pcolor',
 'pcolormesh',
 'phase_spectrum',
 'pie',
 'pink',
 'plasma',
 'plot',
 'plot_date',
 'plotting',
 'polar',
 'prism',
 'psd',
 'quiver',
 'quiverkey',
 'rc',
 'rcParams',
 'rcParamsDefault',
 'rcParamsOrig',
 'rc_context',
 'rcdefaults',
 'rcsetup',
 're',
 'register_cmap',
 'rgrids',
 'savefig',
 'sca',
 'scatter',
 'sci',
 'semilogx',
 'semilogy',
 'set_cmap',
 'set_loglevel',
 'setp',
 'show',
 'specgram',
 'spring',
 'spy',
 'stackplot',
 'stem',
 'step',
 'streamplot',
 'style',
 'subplot',
```

```
 'subplot2grid',
 'subplot_mosaic',
 'subplot_tool',
 'subplots',
 'subplots_adjust',
 'summer',
 'suptitle',
 'switch_backend',
 'sys',
 'table',
 'text',
 'thetagrids',
 'threading',
 'tick_params',
 'ticklabel_format',
 'tight_layout',
 'time',
 'title',
 'tricontour',
 'tricontourf',
 'tripcolor',
 'triplot',
 'twinx',
 'twiny',
 'uninstall_repl_displayhook',
 'violinplot',
 'viridis',
 'vlines',
 'waitforbuttonpress',
 'winter',
 'xcorr',
 'xkcd',
 'xlabel',
 'xlim',
 'xscale',
 'xticks',
 'ylabel',
 'ylim',
 'yscale',
 'yticks']
```

Of course you can explore the [Github (https://github.com/matplotlib/matplotlib/tree/master/lib/matplotlib)](https://github.com/matplotlib/matplotlib/tree/master/lib/matplotlib) to see the source codes if you like.

In [21]:
```
help(plt.plot)
```

```
Help on function plot in module matplotlib.pyplot:

plot(*args, scalex=True, scaley=True, data=None, **kwargs)
    Plot y versus x as lines and/or markers.

    Call signatures::

        plot([x], y, [fmt], *, data=None, **kwargs)
        plot([x], y, [fmt], [x2], y2, [fmt2], ..., **kwargs)

    The coordinates of the points or line nodes are given by *x*, *y*.

    The optional parameter *fmt* is a convenient way for defining basic
    formatting like color, marker and linestyle. It's a shortcut string
    notation described in the *Notes* section below.

    >>> plot(x, y)        # plot x and y using default line style and color
    >>> plot(x, y, 'bo')  # plot x and y using blue circle markers
    >>> plot(y)           # plot y using x as index array 0..N-1
    >>> plot(y, 'r+')     # ditto, but with red plusses

    You can use `.Line2D` properties as keyword arguments for more
    control on the appearance. Line properties and *fmt* can be mixed.
    The following two calls yield identical results:

    >>> plot(x, y, 'go--', linewidth=2, markersize=12)
    >>> plot(x, y, color='green', marker='o', linestyle='dashed',
    ...      linewidth=2, markersize=12)

    When conflicting with *fmt*, keyword arguments take precedence.


    **Plotting labelled data**

    There's a convenient way for plotting objects with labelled data (i.e.
    data that can be accessed by index ``obj['y']``). Instead of giving
    the data in *x* and *y*, you can provide the object in the *data*
    parameter and just give the labels for *x* and *y*::

    >>> plot('xlabel', 'ylabel', data=obj)

    All indexable objects are supported. This could e.g. be a `dict`, a
    `pandas.DataFrame` or a structured numpy array.


    **Plotting multiple sets of data**

    There are various ways to plot multiple sets of data.

    - The most straight forward way is just to call `plot` multiple times.
      Example:

      >>> plot(x1, y1, 'bo')
      >>> plot(x2, y2, 'go')

    - Alternatively, if your data is already a 2d array, you can pass it
      directly to *x*, *y*. A separate data set will be drawn for every
      column.

      Example: an array ``a`` where the first column represents the *x*
      values and the other columns are the *y* columns::

      >>> plot(a[0], a[1:])

    - The third way is to specify multiple sets of *[x]*, *y*, *[fmt]*
      groups::

      >>> plot(x1, y1, 'g^', x2, y2, 'g-')

      In this case, any additional keyword argument applies to all
```

```
  datasets. Also this syntax cannot be combined with the *data*
  parameter.

By default, each line is assigned a different style specified by a
'style cycle'. The *fmt* and line property parameters are only
necessary if you want explicit deviations from these defaults.
Alternatively, you can also change the style cycle using
:rc:`axes.prop_cycle`.


Parameters
----------
x, y : array-like or scalar
    The horizontal / vertical coordinates of the data points.
    *x* values are optional and default to ``range(len(y))``.

    Commonly, these parameters are 1D arrays.

    They can also be scalars, or two-dimensional (in that case, the
    columns represent separate data sets).

    These arguments cannot be passed as keywords.

fmt : str, optional
    A format string, e.g. 'ro' for red circles. See the *Notes*
    section for a full description of the format strings.

    Format strings are just an abbreviation for quickly setting
    basic line properties. All of these and more can also be
    controlled by keyword arguments.

    This argument cannot be passed as keyword.

data : indexable object, optional
    An object with labelled data. If given, provide the label names to
    plot in *x* and *y*.

    .. note::
        Technically there's a slight ambiguity in calls where the
        second label is a valid *fmt*. ``plot('n', 'o', data=obj)``
        could be ``plt(x, y)`` or ``plt(y, fmt)``. In such cases,
        the former interpretation is chosen, but a warning is issued.
        You may suppress the warning by adding an empty format string
        ``plot('n', 'o', '', data=obj)``.

Returns
-------
list of `.Line2D`
    A list of lines representing the plotted data.

Other Parameters
----------------
scalex, scaley : bool, default: True
    These parameters determine if the view limits are adapted to the
    data limits. The values are passed on to `autoscale_view`.

**kwargs : `.Line2D` properties, optional
    *kwargs* are used to specify properties like a line label (for
    auto legends), linewidth, antialiasing, marker face color.
    Example::

    >>> plot([1, 2, 3], [1, 2, 3], 'go-', label='line 1', linewidth=2)
    >>> plot([1, 2, 3], [1, 4, 9], 'rs', label='line 2')

    If you make multiple lines with one plot call, the kwargs
    apply to all those lines.

    Here is a list of available `.Line2D` properties:

    Properties:
```

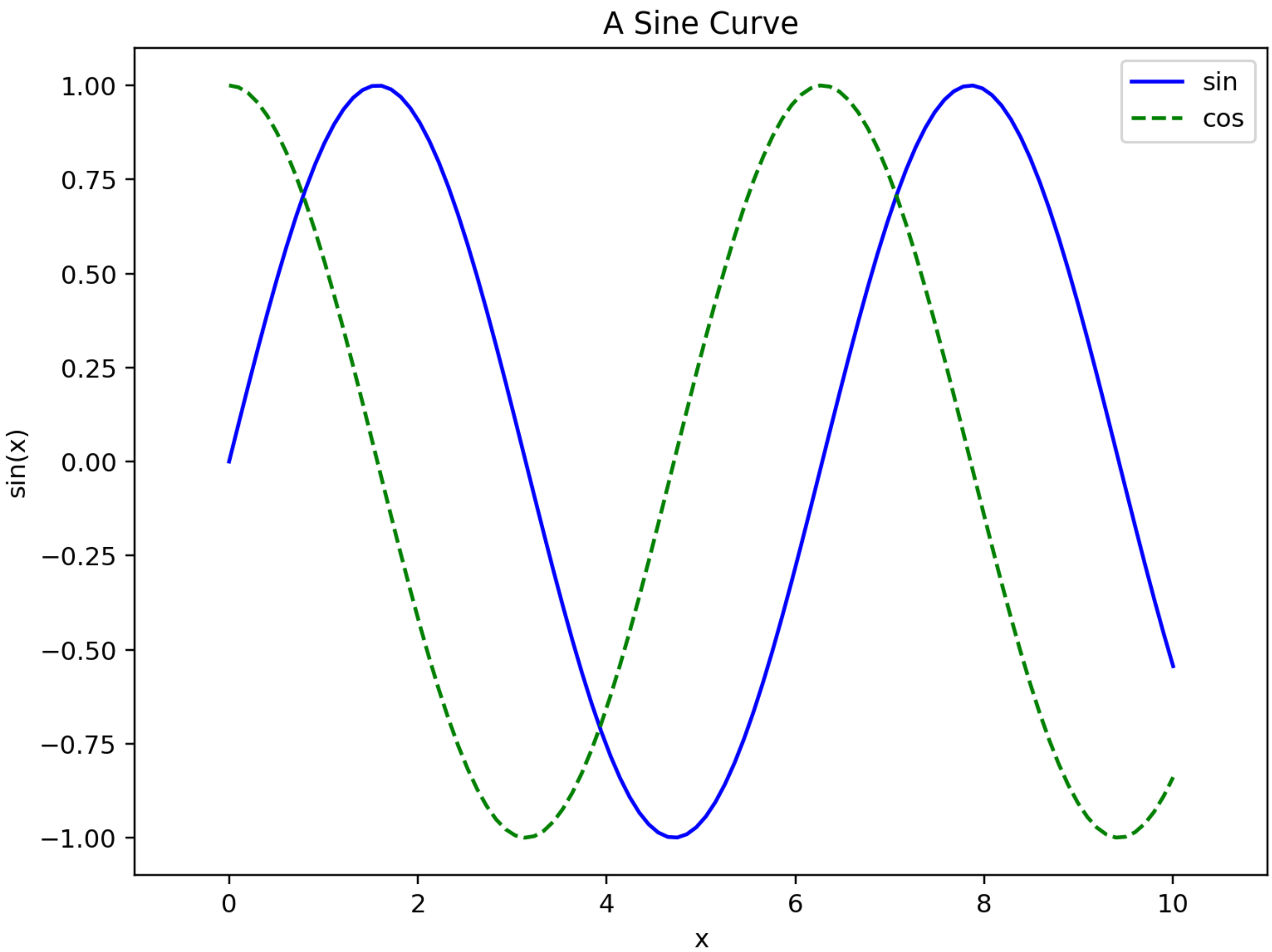
```
        agg_filter: a filter function, which takes a (m, n, 3) float array and a dpi
value, and returns a (m, n, 3) array
        alpha: float or None
        animated: bool
        antialiased or aa: bool
        clip_box: `.Bbox`
        clip_on: bool
        clip_path: Patch or (Path, Transform) or None
        color or c: color
        contains: unknown
        dash_capstyle: {'butt', 'round', 'projecting'}
        dash_joinstyle: {'miter', 'round', 'bevel'}
        dashes: sequence of floats (on/off ink in points) or (None, None)
        data: (2, N) array or two 1D arrays
        drawstyle or ds: {'default', 'steps', 'steps-pre', 'steps-mid', 'steps-pos
t'}, default: 'default'
        figure: `.Figure`
        fillstyle: {'full', 'left', 'right', 'bottom', 'top', 'none'}
        gid: str
        in_layout: bool
        label: object
        linestyle or ls: {'-', '--', '-.', ':', '', (offset, on-off-seq), ...}
        linewidth or lw: float
        marker: marker style string, `~.path.Path` or `~.markers.MarkerStyle`
        markeredgecolor or mec: color
        markeredgewidth or mew: float
        markerfacecolor or mfc: color
        markerfacecoloralt or mfcalt: color
        markersize or ms: float
        markevery: None or int or (int, int) or slice or List[int] or float or (floa
t, float) or List[bool]
        path_effects: `.AbstractPathEffect`
        picker: unknown
        pickradius: float
        rasterized: bool or None
        sketch_params: (scale: float, length: float, randomness: float)
        snap: bool or None
        solid_capstyle: {'butt', 'round', 'projecting'}
        solid_joinstyle: {'miter', 'round', 'bevel'}
        transform: `matplotlib.transforms.Transform`
        url: str
        visible: bool
        xdata: 1D array
        ydata: 1D array
        zorder: float

    See Also
    --------
    scatter : XY scatter plot with markers of varying size and/or color (
        sometimes also called bubble chart).

    Notes
    -----
    **Format Strings**

    A format string consists of a part for color, marker and line::

        fmt = '[marker][line][color]'

    Each of them is optional. If not provided, the value from the style
    cycle is used. Exception: If ``line`` is given, but no ``marker``,
    the data will be a line without markers.

    Other combinations such as ``[color][marker][line]`` are also
    supported, but note that their parsing may be ambiguous.

    **Markers**

    =============    ===============================
    character        description
```

```
=============    ===============================
``'.'``          point marker
``','``          pixel marker
``'o'``          circle marker
``'v'``          triangle_down marker
``'^'``          triangle_up marker
``'<'``          triangle_left marker
``'>'``          triangle_right marker
``'1'``          tri_down marker
``'2'``          tri_up marker
``'3'``          tri_left marker
``'4'``          tri_right marker
``'s'``          square marker
``'p'``          pentagon marker
``'*'``          star marker
``'h'``          hexagon1 marker
``'H'``          hexagon2 marker
``'+'``          plus marker
``'x'``          x marker
``'D'``          diamond marker
``'d'``          thin_diamond marker
``'|'``          vline marker
``'_'``          hline marker
=============    ===============================

**Line Styles**

=============    ===============================
character        description
=============    ===============================
``'-'``          solid line style
``'--'``         dashed line style
``'-.'``         dash-dot line style
``':'``          dotted line style
=============    ===============================

Example format strings::

    'b'    # blue markers with default shape
    'or'   # red circles
    '-g'   # green solid line
    '--'   # dashed line with default color
    '^k:'  # black triangle_up markers connected by a dotted line

**Colors**

The supported color abbreviations are the single letter codes

=============    ===============================
character        color
=============    ===============================
``'b'``          blue
``'g'``          green
``'r'``          red
``'c'``          cyan
``'m'``          magenta
``'y'``          yellow
``'k'``          black
``'w'``          white
=============    ===============================

and the ``'CN'`` colors that index into the default property cycle.

If the color is the only part of the format string, you can
additionally use any  `matplotlib.colors` spec, e.g. full names
(``'green'``) or hex strings (``'#008000'``).
```

Basic usage of pyplot: Very similiar to Matlab

In [29]:
```python
import numpy as np
x = np.linspace(0, 10, 100)
fig = plt.figure(figsize=(8, 6),dpi=220) # create the figure, just like figure() in m
atlab
plt.plot(x, np.sin(x), linestyle = '-',color = 'b',label='sin')  # label is used for
 legend
plt.plot(x, np.cos(x), '--g', label = 'cos')
plt.xlim(-1, 11)
plt.title("A Sine Curve")
plt.xlabel('x')
plt.ylabel("sin(x)")
plt.legend()
```

Out[29]:
```
<matplotlib.legend.Legend at 0x7faec57be290>
```



Of course there is some object-oriented feature.

In [23]:
```python
type(fig)
```

Out[23]:
```
matplotlib.figure.Figure
```

In [24]:
```
dir(fig)
```

```
Out[24]: ['__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__setstate__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 '_add_axes_internal',
 '_agg_filter',
 '_align_xlabel_grp',
 '_align_ylabel_grp',
 '_alpha',
 '_animated',
 '_axobservers',
 '_axstack',
 '_cachedRenderer',
 '_clipon',
 '_clippath',
 '_constrained',
 '_constrained_layout_pads',
 '_contains',
 '_default_contains',
 '_dpi',
 '_gci',
 '_get_clipping_extent_bbox',
 '_get_dpi',
 '_gid',
 '_gridspecs',
 '_in_layout',
 '_label',
 '_layoutbox',
 '_make_key',
 '_mouseover',
 '_normalize_grid_string',
 '_oid',
 '_path_effects',
 '_picker',
 '_process_projection_requirements',
 '_propobservers',
 '_rasterized',
 '_remove_method',
 '_repr_html_',
 '_set_artist_props',
 '_set_dpi',
 '_set_gc_clip',
 '_sketch',
 '_snap',
 '_stale',
 '_sticky_edges',
```

```
'_suptitle',
'_tight',
'_tight_parameters',
'_transform',
'_transformSet',
'_url',
'_visible',
'add_artist',
'add_axes',
'add_axobserver',
'add_callback',
'add_gridspec',
'add_subplot',
'align_labels',
'align_xlabels',
'align_ylabels',
'artists',
'autofmt_xdate',
'axes',
'bbox',
'bbox_inches',
'callbacks',
'canvas',
'clear',
'clf',
'clipbox',
'colorbar',
'contains',
'convert_xunits',
'convert_yunits',
'delaxes',
'dpi',
'dpi_scale_trans',
'draw',
'draw_artist',
'eventson',
'execute_constrained_layout',
'figimage',
'figure',
'findobj',
'format_cursor_data',
'frameon',
'gca',
'get_agg_filter',
'get_alpha',
'get_animated',
'get_axes',
'get_children',
'get_clip_box',
'get_clip_on',
'get_clip_path',
'get_constrained_layout',
'get_constrained_layout_pads',
'get_contains',
'get_cursor_data',
'get_default_bbox_extra_artists',
'get_dpi',
'get_edgecolor',
'get_facecolor',
'get_figheight',
'get_figure',
'get_figwidth',
'get_frameon',
'get_gid',
'get_in_layout',
'get_label',
'get_path_effects',
'get_picker',
'get_rasterized',
'get_size_inches',
```

```
'get_sketch_params',
'get_snap',
'get_tight_layout',
'get_tightbbox',
'get_transform',
'get_transformed_clip_path_and_affine',
'get_url',
'get_visible',
'get_window_extent',
'get_zorder',
'ginput',
'have_units',
'images',
'init_layoutbox',
'is_transform_set',
'legend',
'legends',
'lines',
'mouseover',
'number',
'patch',
'patches',
'pchanged',
'pick',
'pickable',
'properties',
'remove',
'remove_callback',
'savefig',
'sca',
'set',
'set_agg_filter',
'set_alpha',
'set_animated',
'set_canvas',
'set_clip_box',
'set_clip_on',
'set_clip_path',
'set_constrained_layout',
'set_constrained_layout_pads',
'set_contains',
'set_dpi',
'set_edgecolor',
'set_facecolor',
'set_figheight',
'set_figure',
'set_figwidth',
'set_frameon',
'set_gid',
'set_in_layout',
'set_label',
'set_path_effects',
'set_picker',
'set_rasterized',
'set_size_inches',
'set_sketch_params',
'set_snap',
'set_tight_layout',
'set_transform',
'set_url',
'set_visible',
'set_zorder',
'show',
'stale',
'stale_callback',
'sticky_edges',
'subplot_mosaic',
'subplotpars',
'subplots',
'subplots_adjust',
```

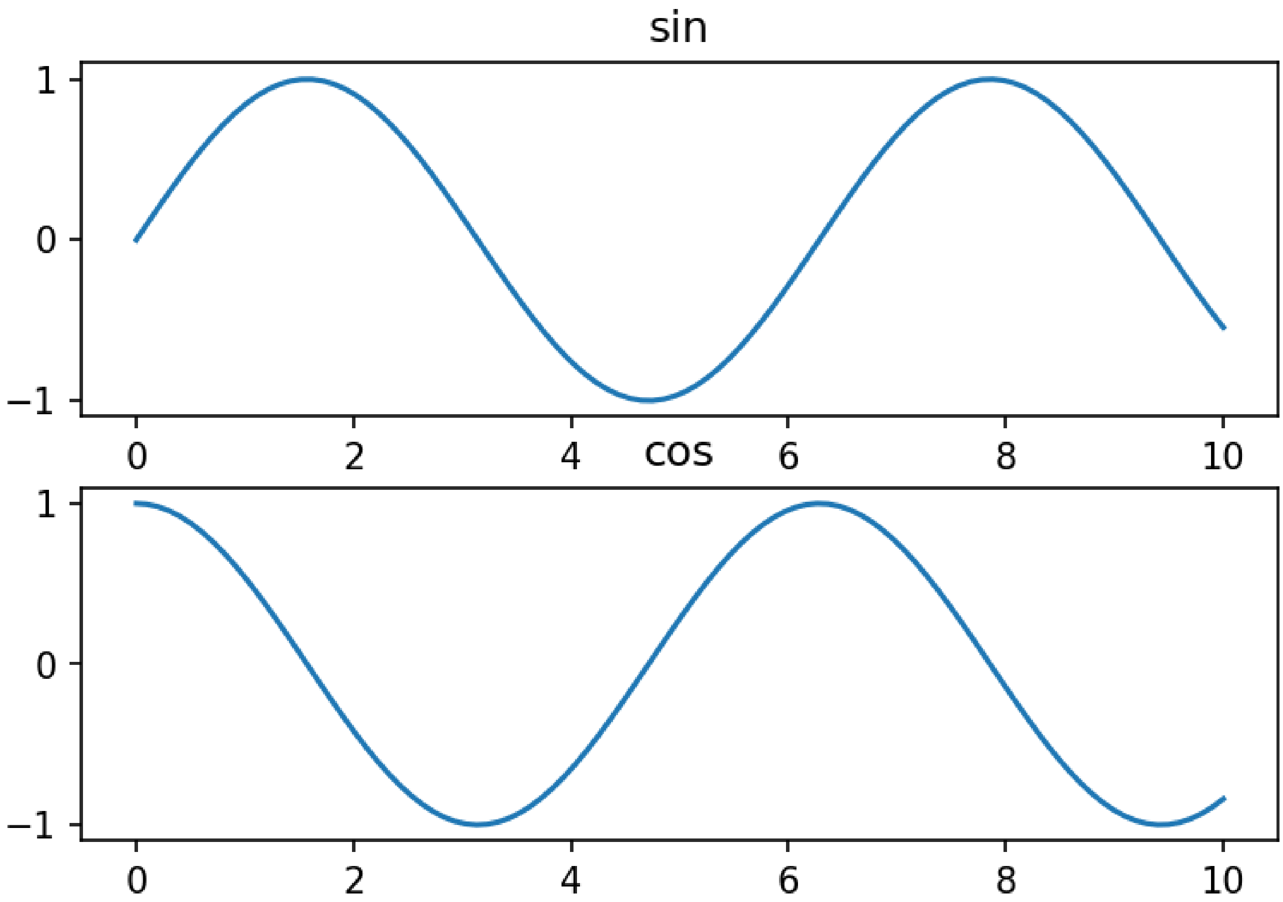
```
 'suppressComposite',
 'suptitle',
 'text',
 'texts',
 'tight_layout',
 'transFigure',
 'update',
 'update_from',
 'waitforbuttonpress',
 'zorder']
```

In [25]:
```
fig.savefig('myfigure.png') # savefig is just a method of instance fig!
```

The object-oriented feature is more evident in making subplots. Explore more usages here (https://matplotlib.org/3.1.0/gallery/subplots_axes_and_figures/subplots_demo.html).

In [30]:
```
# subplots
fig, ax = plt.subplots(2, dpi =150)
ax[0].plot(x, np.sin(x)) # plot and set_title are the methods of ax[0] -axes
ax[0].set_title('sin')
ax[1].plot(x, np.cos(x))
ax[1].set_title('cos')
```

Out[30]:
```
Text(0.5, 1.0, 'cos')
```



Distinguish the concept of axes and axis in Matplotlib (https://matplotlib.org/faq/usage_faq.html)

In [31]:
```
type(ax)
```

Out[31]:
```
numpy.ndarray
```

In [32]:
```
type(ax[0])
```

Out[32]:
```
matplotlib.axes._subplots.AxesSubplot
```

In [33]:
```
fig
```

Out[33]:

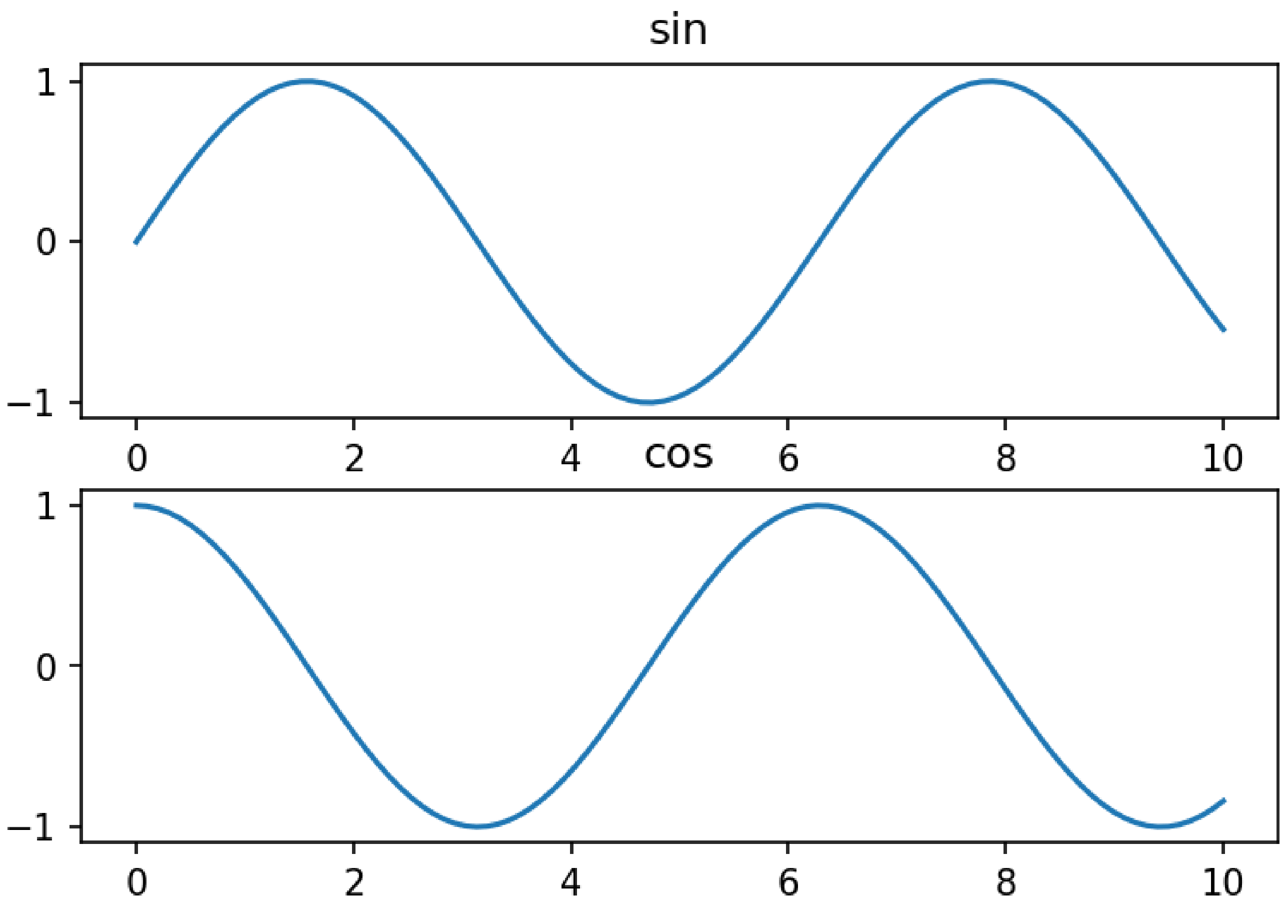


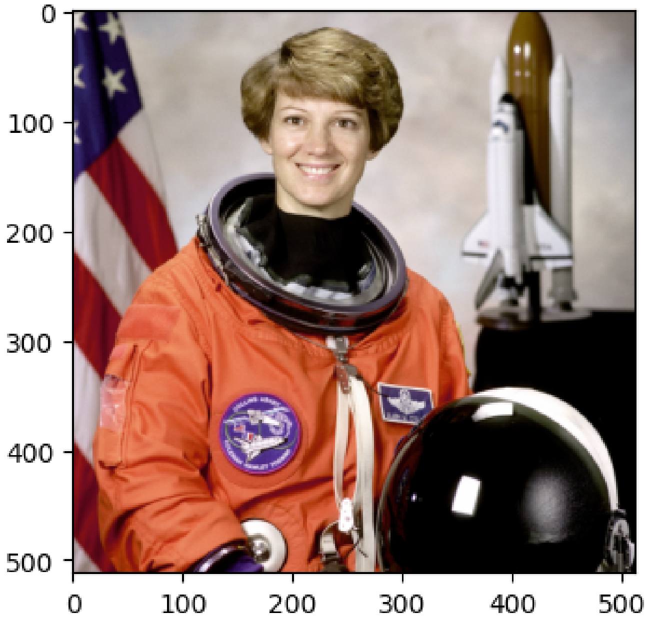
Image Processing

There are many great packages available to handle the image data in Python, such as Pillow (https://pillow.readthedocs.io/en/stable/handbook/tutorial.html#using-the-image-class), Scikit-Image (https://scikit-image.org/) and opencv-python (https://github.com/skvark/opencv-python).

Here we import images from Scikit-Image which is well-compatible with Numpy (https://scikit-image.org/docs/dev/user_guide/numpy_images.html), and use Numpy to manipulate images.

In [34]:
```python
from skimage import data
image_astro = data.astronaut()# read the image as numpy array
image_rock = data.rocket()
fig = plt.figure(dpi=100)
plt.imshow(image_astro)
```

Out[34]:
```
<matplotlib.image.AxesImage at 0x7faec5cab4d0>
```



In [35]:
```python
fig = plt.figure(dpi=100)
plt.imshow(image_rock)
```

Out[35]:
```
<matplotlib.image.AxesImage at 0x7faec5c0fb10>
```

In data science, a common way to store image is through 2D matrix (gray) or 3D tensor (RGB color).

For instance, a gray-scale image with size $m \times n$ can be represented by a matrix $I_1 \in \mathbb{R}^{m \times n}$, whose elements denotes the intensities of pixels.

A color image $m \times n$ can be represented by a tensor (or you can imagine three matrices stacked together) $I_2 \in \mathbb{R}^{m \times n \times 3}$, where the three $m \times n$ matrices denote the intensity in red, green and blue channels respectively (basic assumption is any color can be decomposed in RGB)

In [6]:
```
image_astro.shape # 512-by-512 pixels, with RGB color channels
```
Out[6]: (512, 512, 3)

In [7]:
```
image_rock.shape
```
Out[7]: (427, 640, 3)

In [15]:
```
image_rock[0,0,] # the RGB of first pixel
```
Out[15]: array([17, 33, 58], dtype=uint8)

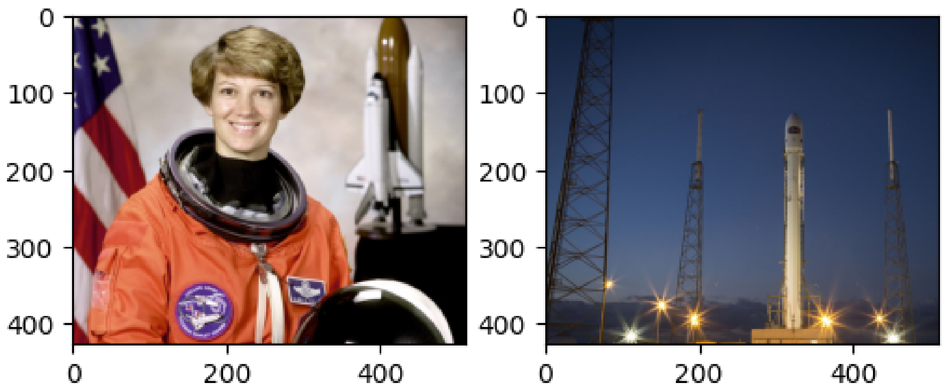
In [36]:
```
[np.max(image_astro),np.min(image_astro)]
```
Out[36]: [255, 0]

Even with simple Numpy expressions, you can do some image processing like in Photoshop!

- Crop the images

In [37]:
```
image_astro_split = image_astro[:427,:,:]
image_rock_split = image_rock[:,:512,:]
```

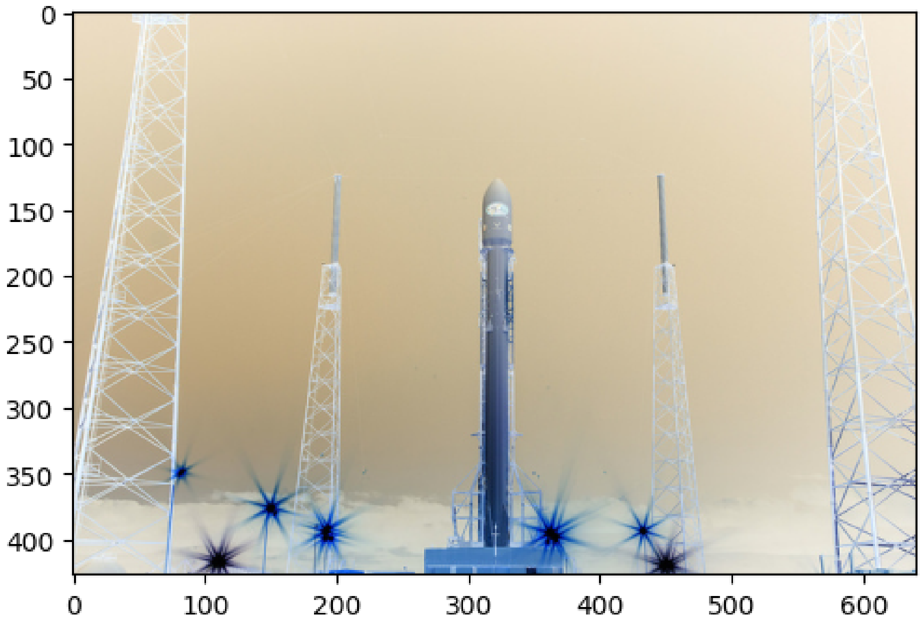
In [38]:
```
fig, ax = plt.subplots(ncols=2, dpi = 100)
ax[0].imshow(image_astro_split) # plot and set_title are the methods of ax[0] -axes
ax[1].imshow(image_rock_split)
```
Out[38]: <matplotlib.image.AxesImage at 0x7faec5bf3090>



- Invert the color intensities

In [39]:
```
fig = plt.figure(dpi=100)
plt.imshow(255-image_rock)
```

Out[39]: <matplotlib.image.AxesImage at 0x7faec6868d50>



- Exchange RGB channels

In [40]:
```
fig = plt.figure(dpi=100)
plt.imshow(image_rock[:,:,[2,1,0]])
```

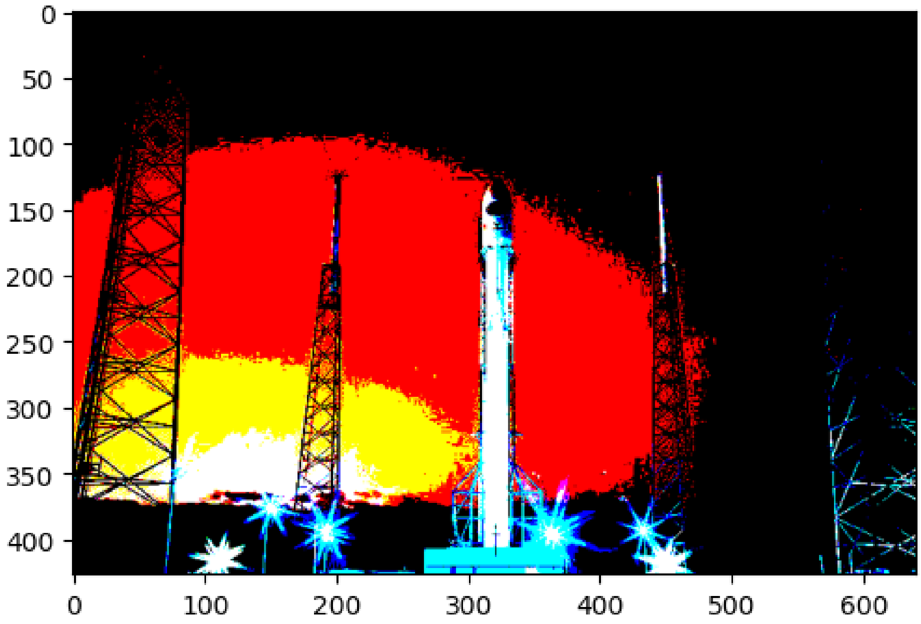
Out[40]: <matplotlib.image.AxesImage at 0x7faec6e03fd0>

- Binarize the image

In [41]:

```python
image = image_rock
image_bi = np.empty_like(image)

thresh = 90
maxval = 255

for i in range(3): #loop over each color channel
    image_bi[:, :, i] = (image[:, :, i] > thresh) * maxval

fig = plt.figure(dpi=100)
plt.imshow(image_bi[:,:,[2,1,0]])
```

Out[41]: <matplotlib.image.AxesImage at 0x7faec34c76d0>



In [28]:
```
image_bi
```

Out[28]:
```
array([[[  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0],
        ...,
        [  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0]],

       [[  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0],
        ...,
        [  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0]],

       [[  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0],
        ...,
        [  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0]],

       ...,

       [[  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0],
        ...,
        [255, 255,   0],
        [255, 255,   0],
        [255, 255,   0]],

       [[  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0],
        ...,
        [255,   0,   0],
        [255,   0,   0],
        [  0,   0,   0]],

       [[  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0],
        ...,
        [  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0]]], dtype=uint8)
```

- Blending

In [42]:
```
image_combine = 0.4*image_astro_split+0.6*image_rock_split
fig = plt.figure(dpi=100)
plt.imshow(image_combine.astype('uint8'))
plt.axis('off')
```

Out[42]:
```
(-0.5, 511.5, 426.5, -0.5)
```