1 - INTRODUCTION TO IMAGE PROCESSING

- Image Processing is a technique to enhance raw images received from cameras / sensors placed on satellites, space probes and aircrafts or pictures taken in normal day-to-day life for various applications.
- **Methods of Image Processing**

  There are two methods available in Image Processing, they are:

  a) Analog Image processing

  b) Digital Image processing

- The principle advantage of Digital Image Processing methods is its versatility, repeatability and the preservation of original data precision.
- The various Image Processing techniques are:

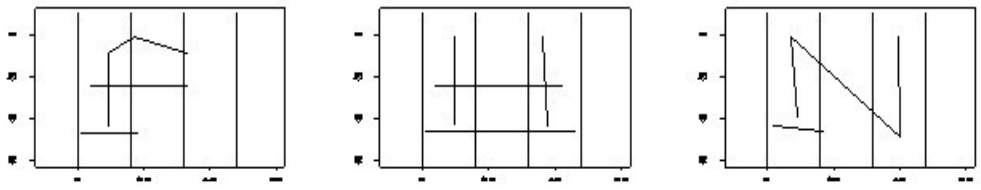
1. Image representation
2. Image pre-processing
3. Image enhancement
4. Image restoration
5. Image analysis
6. Image reconstruction
7. Image data compression

Image Representation :

An image defined in the "real world" is considered to be a function of two real variables, for example, $f(x, y)$ with f as the amplitude (e.g. brightness) of the image at the *real* coordinate position (x, y).

Image Enhancement Techniques:

Sometimes images obtained from satellites and conventional and digital cameras lack in contrast and brightness because of the limitations of imaging sub systems and illumination conditions while capturing image. Images may have different types of noise

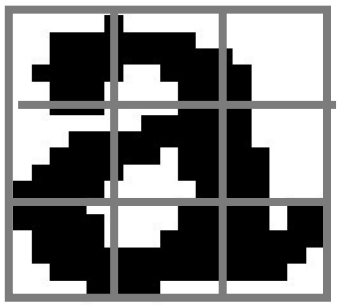


Strokes extracted from the capital letters F, H and N.

Image Segmentation:

Image segmentation is the process that subdivides an image into its constituent parts or objects. The level to which this subdivision is carried out depends on the problem being solved, i.e., the segmentation should stop when the objects of interest in an application have been isolated e.g., in autonomous air-to ground target acquisition, suppose our interest lies in identifying vehicles on a road, the first step is to segment the road from the image and then to segment the contents of the road down to potential vehicles. Image thresholding techniques are used for image segmentation.

es



Zoning

Image Restoration:

Image restoration refers to removal or minimization of degradations in an image. This includes de-blurring of images degraded by the limitations of a sensor or its environment, noise filtering, and correction of geometric distortion or non-linearity due to sensors

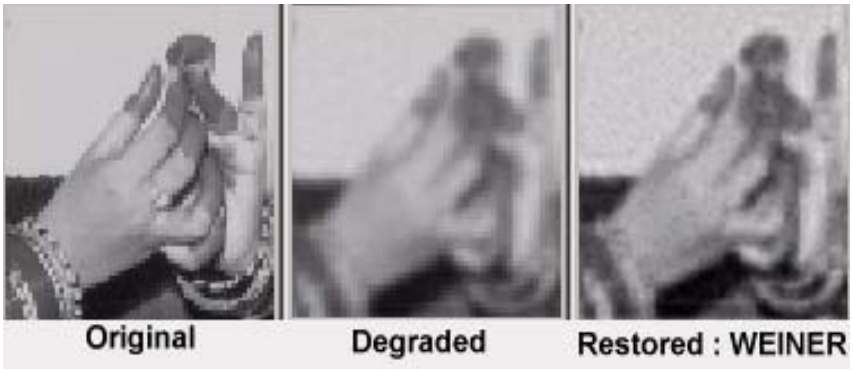
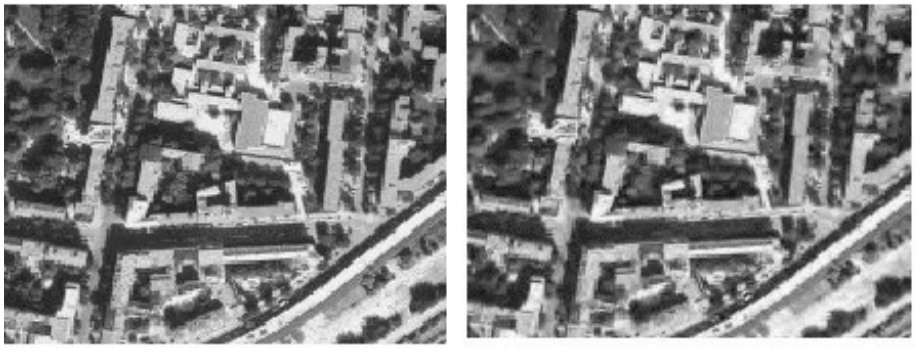


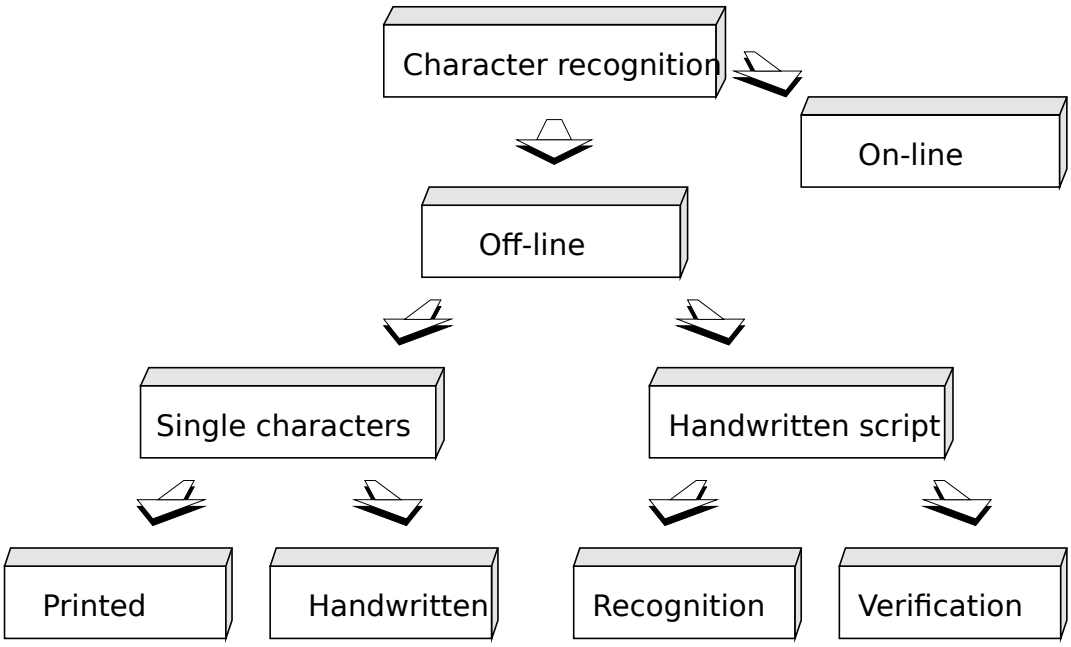
Image Compression :

- Compression is a very essential tool for archiving image data, image data transfer on the network etc.
- They are various techniques available for lossy and lossless compressions.
- One of most popular compression techniques, JPEG (Joint Photographic Experts Group) uses Discrete
- Cosine Transformation (DCT) based compression technique.
- Currently wavelet based compression techniques are used for higher compression ratios with minimal loss of data.



2 - INTRODUCTION TO OCR

- OCR refers to Optical Character Recognition, it belongs to the family of techniques performing automatic identification.
- Speech recognition,Radio frequency,Vision systems,Magnetic stripe,Bar code,Magnetic ink,Optical Mark Reading, Optical Character Recognition are enmormously using this.
- Optical Character Recognition deals with the problem of recognizing optically processed characters.
- Optical recognition is performed offline after the writing or printing has been completed, as opposed to online recognition where the computer recognizes the characters as they are drawn.
- Both hand printed and printed characters may be recognized, but the performance is directly dependent upon the quality of the input documents.



History of OCR:

In 1870,the C.R.Carey of Boston Massachusetts invented the retina scanner, Polish P. Nipkow invented the sequential scanner which led to development of digital computer.

First Generation OCR:

- The commercial OCR systems appearing in the period from 1960 to 1965 may be called the first generation of OCR
- This generation of OCR machines were mainly characterized by the constrained letter shapes read.
- The symbols were specially designed for machine reading, and the first ones did not even look very natural.

Second Generation OCR:

- The reading machines of the second generation appeared in the middle of the 1960's and early 1970's.
- These systems were able to recognize regular machine printed characters and also had hand-printed character recognition capabilities.
- When hand-printed characters were considered, the character set was constrained to numerals and a few letters and symbols.
- In 1966, a thorough study of OCR requirements was completed and an American standard OCR character set was defined; OCR-A.
- This font was highly stylized and designed to facilitate optical recognition, although still readable to humans.
- A European font was also designed, OCR-B, which had more natural fonts than the American standard.

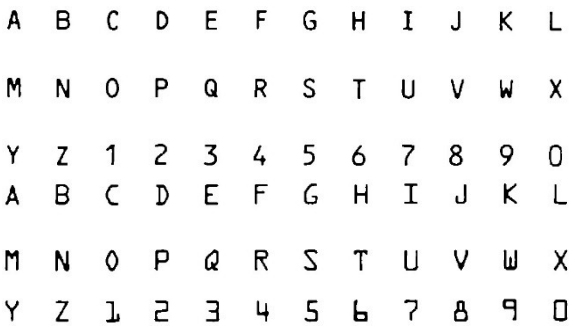


Figure 2 : OCR-A (top), OCR-B (bottom)

Third Generation OCR:

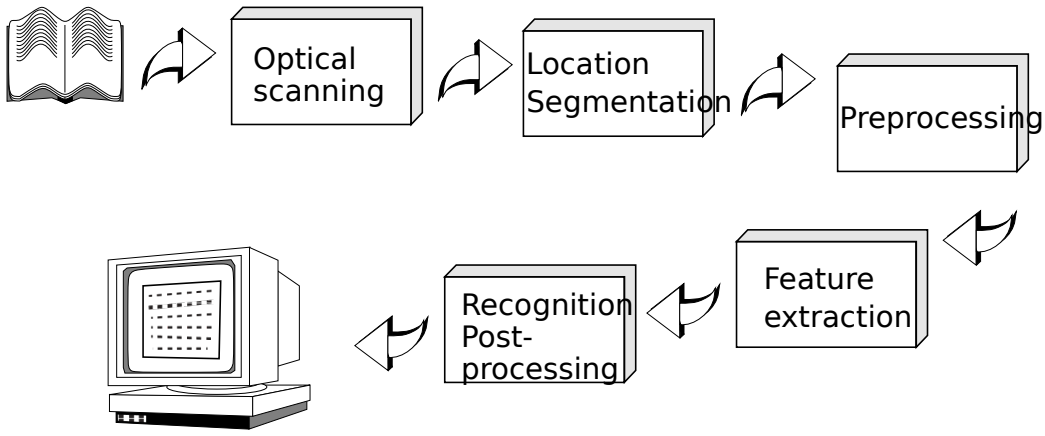
- The uniform print spacing and small number of fonts made simply designed OCR devices very useful.
- Rough drafts could be created on ordinary typewriters and fed into the computer through an OCR device for final editing.
- In this way word processors, which were an expensive resource at this time, could support several people and the costs for equipment could be cut.

1870	The very first attempts
1940	The modern version of OCR.
1950	The first OCR machines appear
1960 - 1965	First generation OCR
1965 – 1975	Second generation OCR
1975 - 1985	Third generation OCR
1986 ->	OCR to the people

A short OCR chronology

Components of OCR:

- The first step in the process is to digitize the analog document using an optical scanner.
- When the regions containing text are located, each symbol is extracted through a segmentation process.
- The extracted symbols may then be preprocessed, eliminating noise, to facilitate the extraction of features in the next step.



Components of an OCR-system

- The identity of each symbol is found by comparing the extracted features with descriptions of the symbol classes obtained through a previous learning phase.
- Finally contextual information is used to reconstruct the words and numbers of the original text.

APPLICATIONS OF OCR:

1. Data entry
2. Text entry
3. Process automation
4. Automatic number-plate readers
5. Automatic cartography
6. Form readers
7. Signature verification and identification

The Future Of OCR:

- New methods for character recognition are still expected to appear, as the computer technology develops and decreasing computational restrictions open up for new approaches.
- The greatest potential seems to lie within the exploitation of existing methods, by mixing methodologies and making more use of context.
- Integration of segmentation and contextual analysis can improve recognition of joined and split characters.
- Higher level contextual analysis which look at the semantics of entire sentences may be useful.
- The frontiers of research within character recognition have now moved towards the recognition of cursive script, that is handwritten connected or calligraphic characters.

OCR COMPLETION GUIDANCE

RULES	EXAMPLES	
	Correct	Incorrect
1. Use black pen whenever possible.		
2. Form large characters, but within the box edges.	32460	32460
3. Use simple shapes, avoid loops or curls or flourishes.	23705	23705
4. Close loops.	0689	6689
5. Connect lines.	45	45
6. Do not use alternative shape four continental seven continental one.	471	471
7. Do not link characters.	5621	5621
8. Do not overlap characters.	4762	47 62

Alpha Character Set

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Numeric Character Set 1 2 3 4 5 6 7 8 9 0

Instructions for OCR handwriting

NumPY and SciPY

- NumPy and SciPy are open-source add-on modules to Python that provide common mathematical and numerical routines in pre-compiled, fast functions.
- These are growing into highly mature packages that provide functionality that meets, or perhaps exceeds, that associated with common commercial software like MatLab.
- The NumPy (Numeric Python) package provides basic routines for manipulating large arrays and matrices of numeric data.
- The SciPy (Scientific Python) package extends the functionality of NumPy with a substantial collection of useful algorithms, like minimization, Fourier transformation, regression, and other applied mathematical techniques.

Importing the NumPy module

There are several ways to import NumPy. The standard approach is to use a simple import statement in python :

```
>>> import numpy
```

However, for large amounts of calls to NumPy functions, it can become tedious to write numpy.X over and over again. Instead, it is common to import under the briefer name np:

```
>>> import numpy as np
```

This statement will allow us to access NumPy objects using np.X instead of numpy.X. It is also possible to import NumPy directly into the current namespace so that we don't have to use dot notation at all, but rather simply call the functions as if they were built-in:

```
>>> from numpy import *
```

Arrays

The central feature of NumPy is the *array* object class. Arrays are similar to lists in Python, except that every element of an array must be of the same type, typically a numeric type like float or int. Arrays make operations with

large amounts of numeric data very fast and are generally much more efficient than lists.

An array can be created from a list:

```
>>> a = np.array([1, 4, 5, 8], float)
>>> a
array([ 1.,  4.,  5.,  8.])
>>> type(a)
<type 'numpy.ndarray'>
```

Array slicing works with multiple dimensions in the same way as usual, applying each slice specification as a filter to a specified dimension. Use of a single ":" in a dimension indicates the use of everything along that dimension:

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> a[1,:]
array([ 4.,  5.,  6.])
>>>              a[:,2]
array([   3.,      6.])
>>>         a[-1:,-2:]
array([[ 5.,  6.]])
```

Polynomial mathematics

NumPy supplies methods for working with polynomials. Given a set of roots, it is possible to show the polynomial coefficients:

```
>>>   np.poly([-1,   1,   1,   10])
array([  1, -11,   9,  11, -10])
```

Here, the return array gives the coefficients corresponding to $x^4 - 11x^3 + 9x^2 + 11x - 10$.

The opposite operation can be performed: given a set of coefficients, the root function returns all of the polynomial roots:

```
>>> np.roots([1, 4, -2, 3])
array([-4.57974010+0.j        ,  0.28987005+0.75566815j,
       0.28987005-0.75566815j])
```

Statistics

In addition to the mean, var, and std functions, NumPy supplies several other methods for returning statistical features of arrays.

The median can be found:

```
>>> a = np.array([1, 4, 3, 8, 9, 2, 3], float)
>>> np.median(a)
3.0
```

The correlation coefficient for multiple variables observed at multiple instances can be found for arrays of the form [[x1, x2, …], [y1, y2, …], [z1, z2, …], …] where x, y, z are different observables and the numbers indicate the observation times:

```
>>> a = np.array([[1, 2, 1, 3], [5, 3, 1, 8]], float)
>>> c = np.corrcoef(a)
>>> c
array([[ 1.        ,  0.72870505],
[ 0.72870505,  1.        ]])
```

Here the return array c[i,j] gives the correlation coefficient for the ith and jth observables. Similarly, the covariance for data can be found:

```
>>> np.cov(a)
```

```
array([[ 0.91666667,  2.08333333],
[ 2.08333333,  8.91666667]])
```

Modules available in SciPy

SciPy greatly extends the functionality of the NumPy routines. We will not cover this module in detail but rather mention some of its capabilities. Many SciPy routines can be accessed by simply importing the module:

```
>>> import scipy
```

information on the packages that SciPy offers:

```
>>> help(scipy)
Help on package scipy:
  NAME
scipy
  FI
LE
```

c:\python25\lib\site-packages\scipy__init__.py

DESCRIPTION

SciPy --- A scientific computing package for Python

==========

=====

Documentation is available in the docstrings and online at http://docs.scipy.org.

Contents

SciPy imports all the functions from the NumPy namespace, and in addition provides:

Available subpackages ----------

odr --- Orthogonal Distance Regression [*] misc --- Various utilities that don't have another home. sparse.linalg.eigen.arpack --- Eigenvalue solver using iterative methods. [*]

fftpack --- Discrete Fourier Transform algorithms [*]

io --- Data input and output [*]

sparse.linalg.eigen.lobpcg --- Locally Optimal Block Preconditioned Conjugate Gradient Method (LOBPCG) [*] special --- Airy Functions [*] lib.blas --- Wrappers to BLAS library [*] sparse.linalg.eigen --- Sparse Eigenvalue Solvers [*] stats --- Statistical Functions [*]

lib --- Python wrappers to external libraries [*]

lib.lapack --- Wrappers to LAPACK library [*] maxentropy --- Routines for fitting maximum entropy models [*]

integrate --- Integration routines [*] ndimage --- n-dimensional image package [*] linalg --- Linear algebra routines [*]

spatial --- Spatial data structures and algorithms [*]

interpolate --- Interpolation Tools [*] sparse.linalg --- Sparse Linear Algebra [*]

sparse.linalg.dsolve.umfpack --- :Interface to the UMFPACK library: [*] sparse.linalg.dsolve --- Linear Solvers [*] optimize --- Optimization Tools [*] cluster --- Vector Quantization / Kmeans [*] signal --- Signal Processing Tools [*] sparse --- Sparse Matrices [*]

[*] - using a package requires explicit import (see pkgload) .

The table below we mention a subset of its capabilities:

module	**code for…**
scipy.constants	Many mathematical and physical constants.
scipy.special	Special functions for mathematical physics, such as iry, elliptic, bessel, gamma, beta, hypergeometric, parabolic cylinder, mathieu, spheroidal wave, struve, and kelvin functions.
scipy.integrate	Functions for performing numerical integration using trapezoidal, Simpson's, Romberg, and other methods. Also provides methods for integration of ordinary differential equations.
scipy.optimize	Standard minimization / maximization routines that operate on generic user-defined objective functions. Algorithms include: Nelder-Mead Simplex, Powell's, conjugate gradient, BFGS, least-squares, constrained optimizers, simulated annealing, brute force, Brent's method, Newton's method, bisection method, Broyden, Anderson, and line search.
scipy.linalg	Much broader base of linear algebra routines than NumPy. Offers more control for using special, faster routines for specific cases (e.g., tridiagonal matrices). Methods include: inverse, determinant, solving a linear system of equations, computing norms and pseudo/generalized inverses, eigenvalue/eigenvector decomposition, singular value decomposition, LU decomposition, Cholesky decomposition, QR decomposition, Schur decomposition, and various other mathematical operations on matrices.
scipy.sparse	Routines for working with large, sparse matrices.

scipy.interpolate	Routines and classes for interpolation objects that can be used with discrete numeric data. Linear and spline interpolation available for one- and two-dimensional data sets.
scipy.fftpack	Fast Fourier transform routines and processing.

OPENCV:

- OpenCV (Open Source Computer Vision Library is an open-source BSD-licensed library that includes several hundreds of computer vision algorithms. The document describes the so-called OpenCV 2.x API, which is essentially a C++ API, as opposite to the C-based OpenCV 1.x API. The latter is described in opencv1x.pdf.

  OpenCV has a modular structure, which means that the package includes several shared or static libraries. The following modules are available:

  - core - a compact module defining basic data structures, including the dense multi-dimensional array Mat and basic functions used by all other modules.
  - imgproc - an image processing module that includes linear and non-linear image filtering, geometrical image transformations (resize, affine and perspective warping, generic table-based remapping), color space conversion, histograms, and so on.
  - video - a video analysis module that includes motion estimation, background subtraction, and object tracking algorithms.
  - calib3d - basic multiple-view geometry algorithms, single and stereo camera calibration, object pose estimation, stereo correspondence algorithms, and elements of 3D reconstruction.
  - features2d - salient feature detectors, descriptors, and descriptor matchers.
  - objdetect - detection of objects and instances of the predefined classes (for example, faces, eyes, mugs, people, cars, and so on).
  - highgui - an easy-to-use interface to video capturing, image and video codecs, as well as simple UI capabilities.

- gpu - GPU-accelerated algorithms from different OpenCV modules

some other helper modules , such as FLANN and Google test wrappers, Python bindings, and others.

API concepts

- if any (this part involves decrementing the reference counter and comparing it with zero), and then allocates a new buffer of the required size.
- Most functions call the Mat::create method for each output array, and so the automatic output data allocation is implemented.
- As a computer vision library, OpenCV deals a lot with image pixels that are often encoded in a compact, 8- or 16-bit per channel, form and thus have a limited value range.

Error Handling

OpenCV uses exceptions to signal critical errors. When the input data has a correct format and belongs to the specified value range, but the algorithm cannot succeed for some reason (for example, the optimization algorithm did not converge), it returns a special error code (typically, just a boolean variable).

The exceptions can be instances of the cv::Exception class or its derivatives. In its turn, cv::Exception is a derivative of std::exception. So it can be gracefully handled in the code using other standard C++ library components.

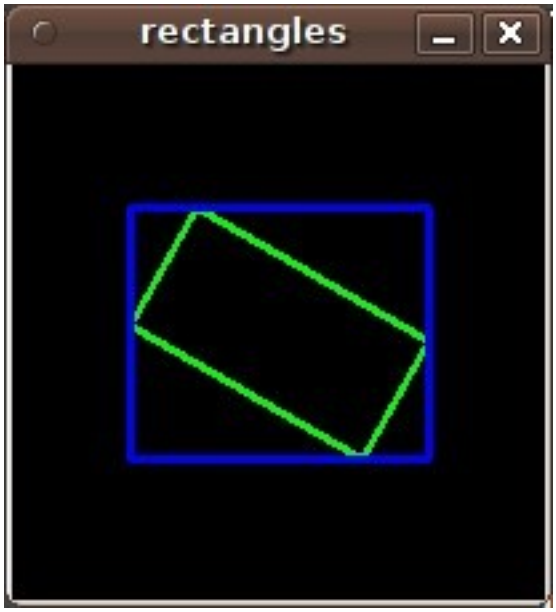
The exception is typically thrown either using the CV_Error(errcode, description) macro, or its printf-like CV_Error_(errcode, printf-spec, (printf-args)) variant, or using the CV_Assert(condition) macro that checks the condition and throws an exception when it is not satisfied. For performance-critical code, there is CV_DbgAssert(condition) that is only retained in the Debug configuration. Due to the automatic memory management, all the intermediate buffers are automatically deallocated in case of a sudden error. You only need to add a try statement to catch exceptions, if needed:

```
{
    ... // call OpenCV
```

```
}
catch( cv::Excep
tion& e )
{ const char* err_msg = e.what(); std::cout <<
    "exception caught: " << err_msg <<
    std::endl;
}
```

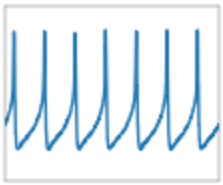
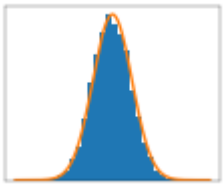
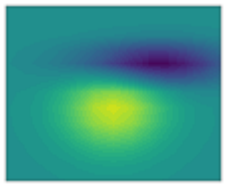
The sample below demonstrates how to use RotatedRect:

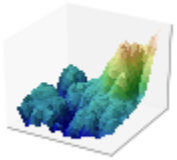
```
Mat image(200, 200, CV_8UC3, Scalar(0));
RotatedRect rRect = RotatedRect(Point2f(100,100), Size2f(100,50), 30);

Point2f vertices[4];
rRect.points(vertice
s);
for (int i = 0; i < 4; i++)
    line(image, vertices[i], vertices[(i+1)%4], Scalar(0,255,0));

Rect brect =
rRect.boundingRect();
rectangle(image, brect,
Scalar(255,0,0));

imshow("rectan
gles", image);
waitKey(0);
```



MATPLOTLIB:

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shell, the jupyter notebook, web application servers, and four graphical user interface toolkits.

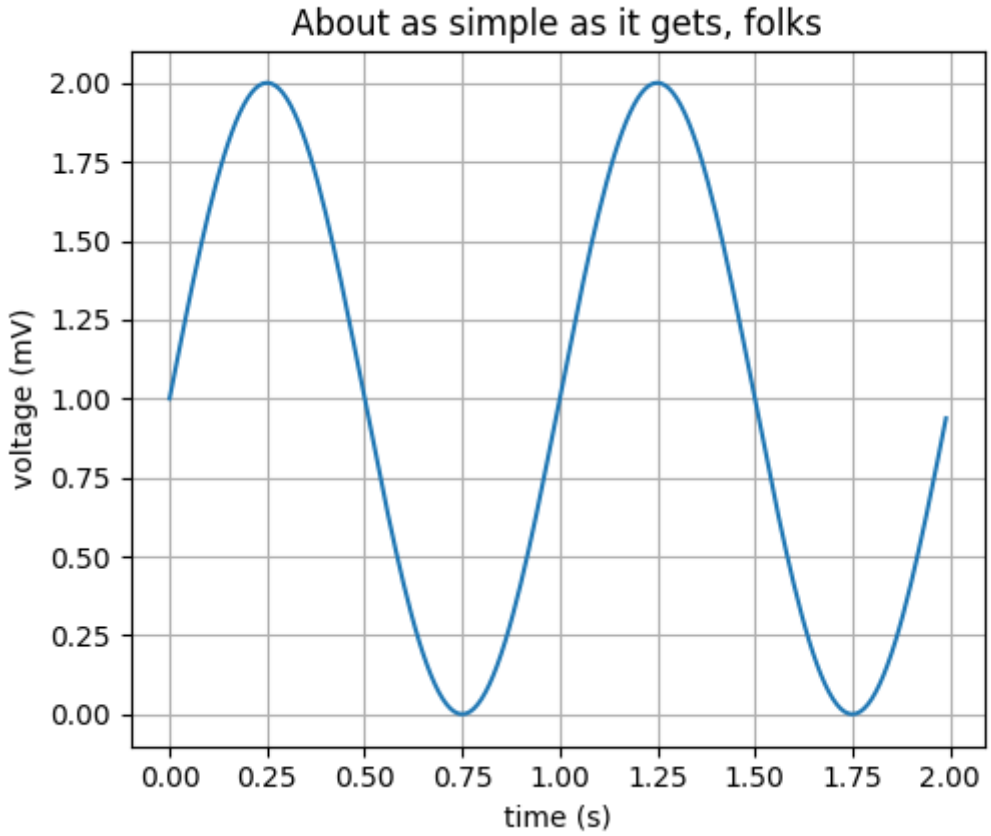






- Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc., with just a few lines of code.

- Simple Plot

Here's a very basic plot() with text labels:

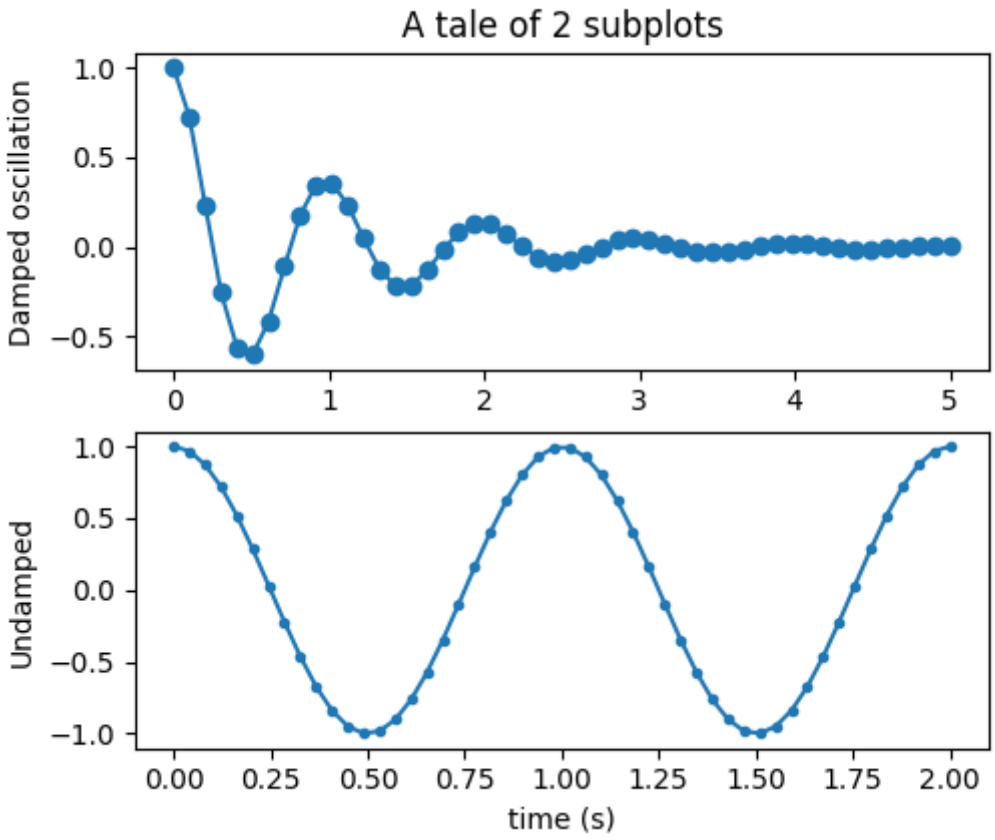


- For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

_

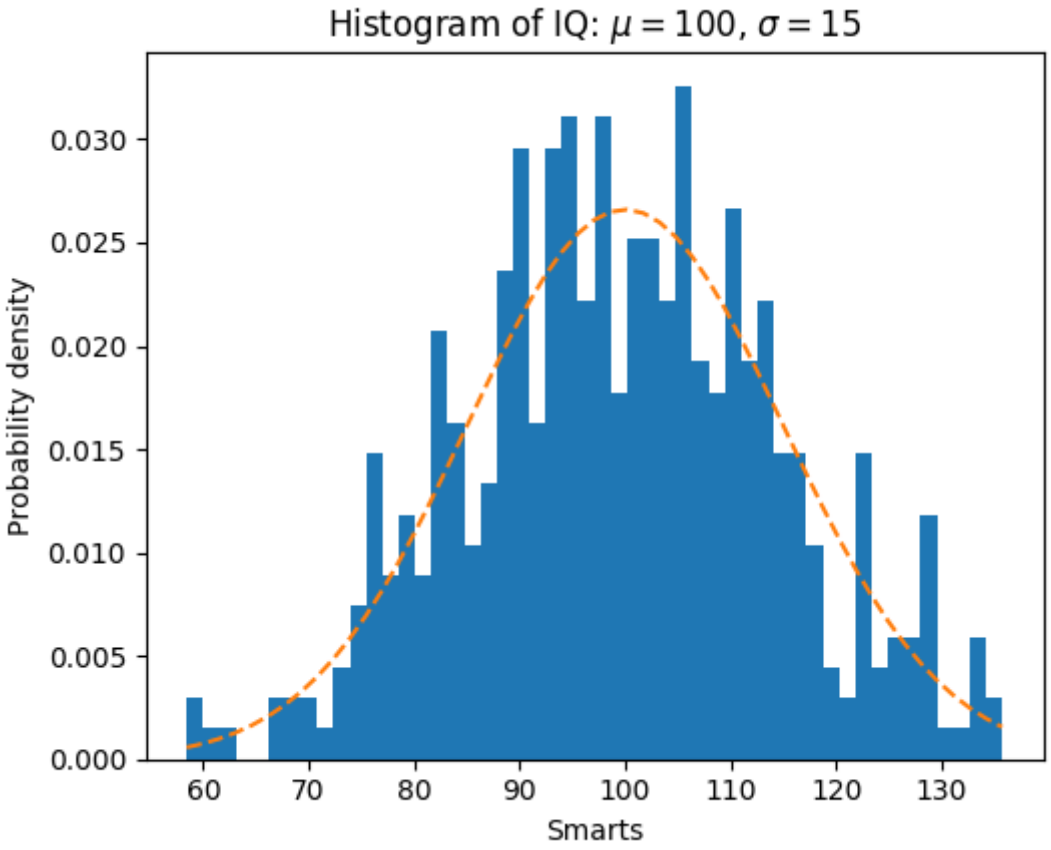
Subplot demo

Multiple axes (i.e. subplots) are created with the subplot() command:



Histograms

The hist() command automatically generates histograms and returns the bin counts or probabilities:



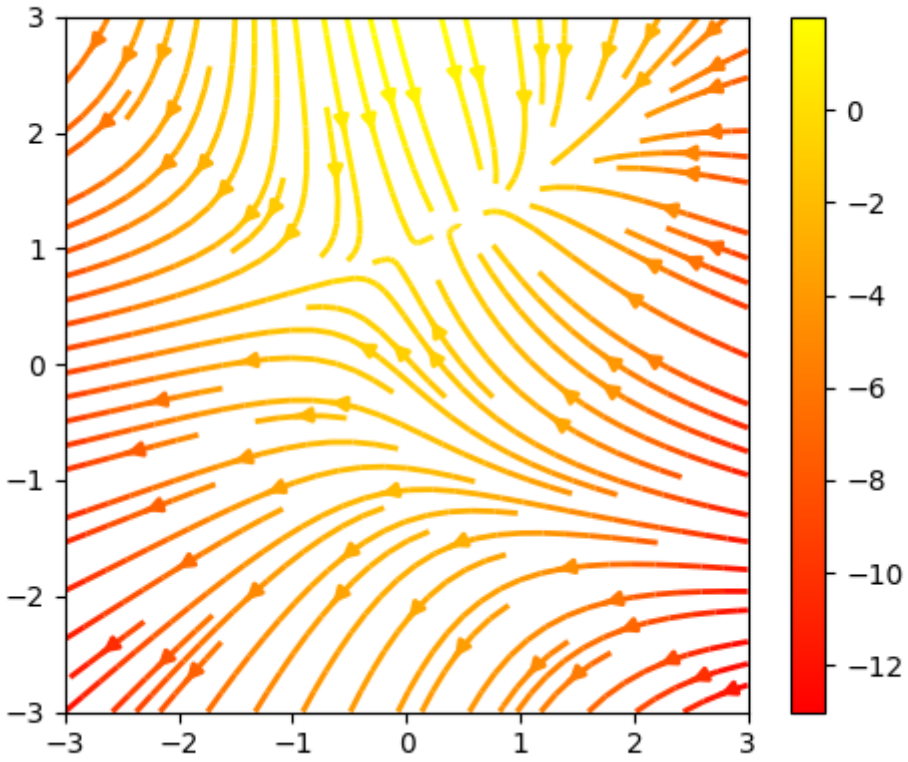
mplot3d

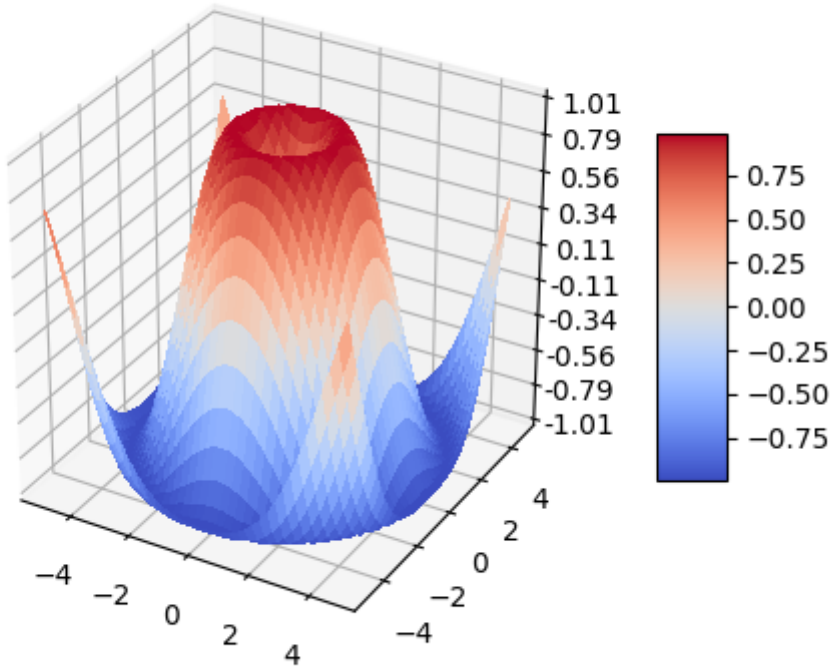
The mplot3d toolkit (see mplot3d tutorial and mplot3d Examples) has support for simple 3d graphs including surface, wireframe, scatter, and bar charts.

Streamplot

The streamplot() function plots the streamlines of a vector field. In addition to simply plotting the streamlines, it allows you to map the colors and/or line widths of streamlines to a separate parameter, such as the speed or local intensity of the vector field.

(Source code)


3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
0
−2
−4
−6
−8
−10
−12

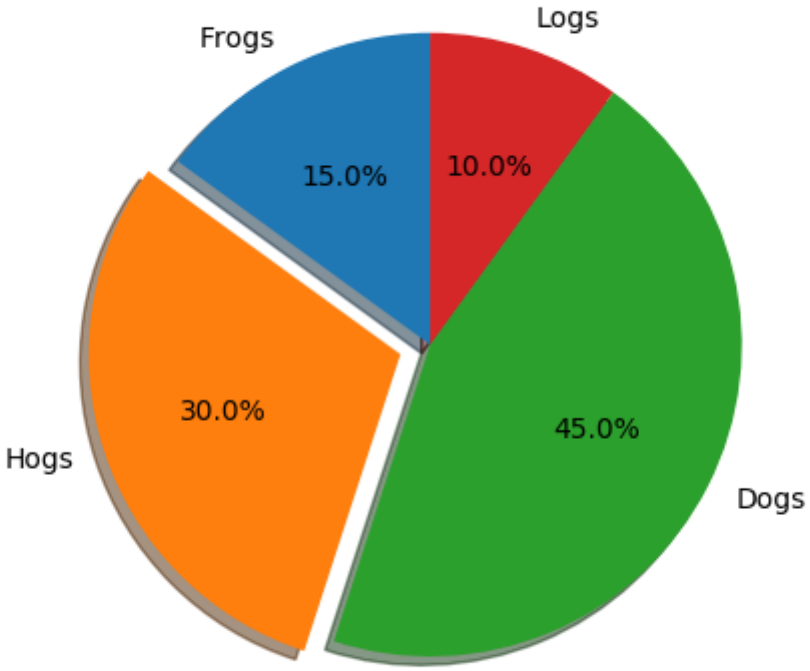

1.01
0.79
0.56
0.34
0.11
-0.11
-0.34
-0.56
-0.79
-1.01
−4
−2
0
2
4
4
2
0
−2
−4
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

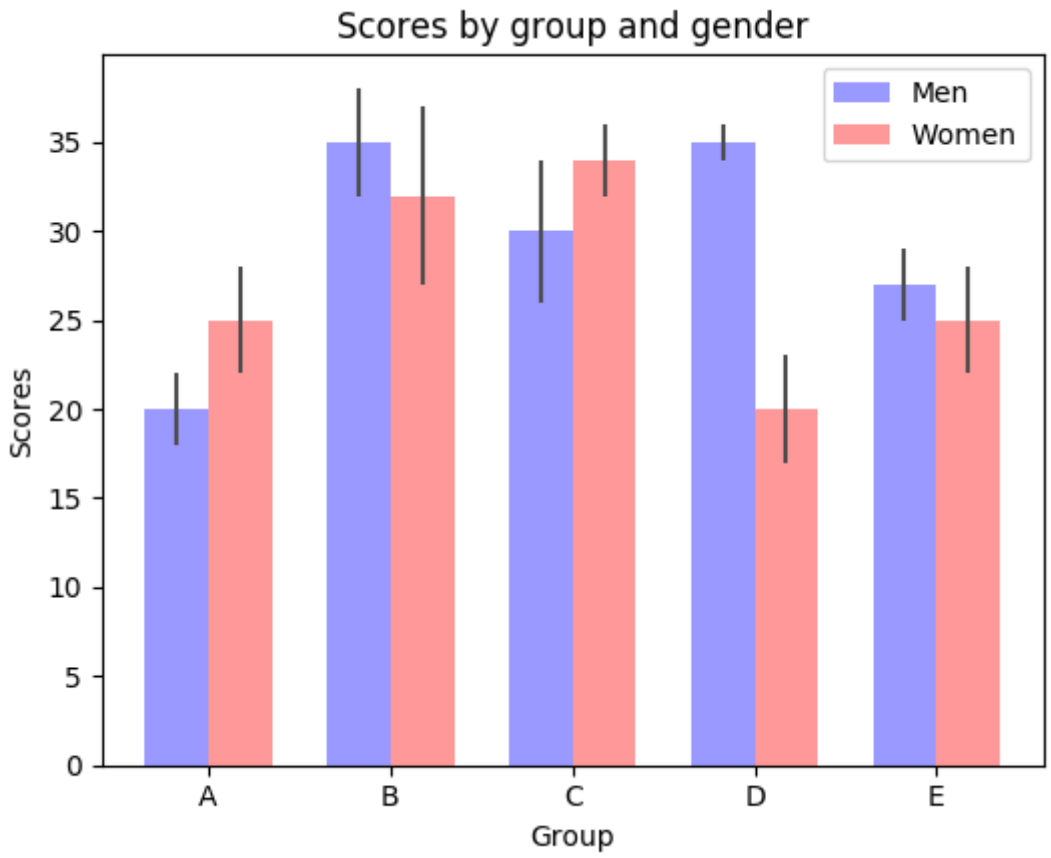
Bar charts

Bar charts are simple to create using the bar() command, which includes customizations such as error bars:

Pie charts

The pie() command allows you to easily create pie charts. Optional features include auto-labeling the percentage of area, exploding one or more wedges from the center of the pie, and a shadow effect. Take a close look at the attached code, which generates this figure in just a few lines of code.





Line plots

- The basic syntax for creating line plots is plt.plot(x,y), where x and y are arrays of the same length that specify the (*x*,*y*) pairs that form the line. For example, let's plot the cosine function from −2 to . To do so, we need to provide a discretization (grid) of the values along the *x*-axis, and evaluate the function on each *x* value. This can typically be done with numpy.arange or numpy.linspace.

  xvals = np.arange(-2, 1, 0.01) # Grid of 0.01 spacing from -2 to 10 yvals = np.cos(xvals) # Evaluate function on xvals plt.plot(xvals, yvals) # Create line plot with yvals against xvals plt.show()

- The plt.show command last after you have made all relevant changes to the plot. You can create multiple figures by creating new figure windows with plt.figure().

- To output all these figures at once, you should only have one plt.show command at the very end. Also, unless you turned the interactive mode on, the code will be paused until you close the figure window.

  newyvals = 1 - 0.5 * xvals**2 # Evaluate quadratic approximation on xvals plt.plot(xvals, newyvals, 'r--') # Create line plot with red dashed line plt.title('Example plots') plt.xlabel('Input') plt.ylabel('Function values')