COMP1730/COMP6730
Programming for Scientists

Sequence types

Lecture outline

* Sequence data types
* NumPy arrays
* Indexing, length & slicing

Sequences

- A *sequence* contains zero or more values.
- Each value in a sequence has a *position*, or *index*, ranging from 0 to $n - 1$.
- The *indexing operator* can be applied to all sequence types, and returns the value at a specified position in the sequence.
  - Indexing is done by writing the index in square brackets after the sequence value, like so:

    `sequence[pos]`

Sequence data types

* python has three built-in sequence types:
  - strings (`str`) contain only text;
  - lists (`list`) can contain a mix of value types;
  - tuples (`tuple`) are like lists, but immutable.
* Sequence types provided by other modules:
  - NumPy arrays (`numpy.ndarray`).

NumPy arrays

NumPy and SciPy

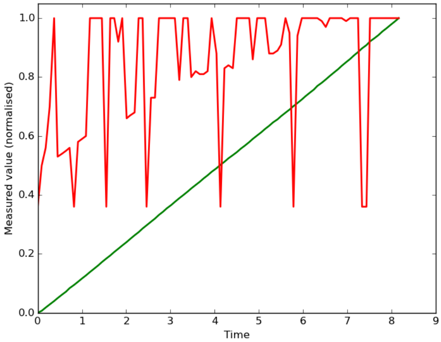
* The NumPy and SciPy libraries are not part of the python standard library, but often considered essential for scientific / engineering applications.
* The NumPy and SciPy libraries provide
  – an *n*-dimensional array data type (`ndarray`);
  – fast math operations on arrays/matrices;
  – linear algebra, Fourier transform, random number generation, signal processing, optimisation, and statistics functions;
  – plotting (via `matplotlib`).
* Documentation: `numpy.org` and `scipy.org`.

Problem: Sensor modelling

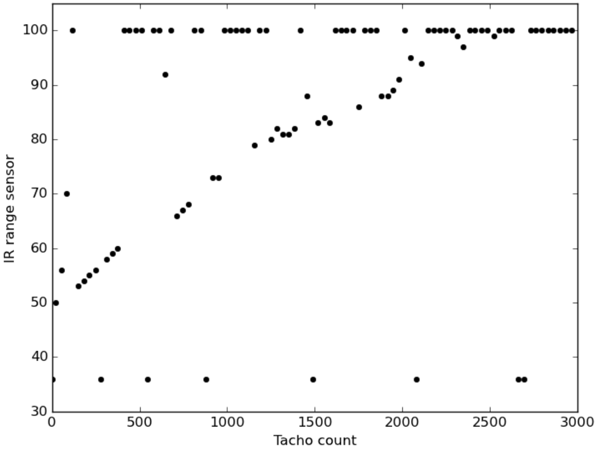
* Time series of two measurements:

* IR sensor (% of range)

* Tachometer ($^1/_{360}$th rev.)



* Is there a linear relation between *x* and *y*?



- Fit a straight line ($y = ax + b$) as close to all of the points as possible.
  - This can be done by solving a least-squares optimisation problem.
  - Simpler idea: Calculate the average slope between pairs of (adjacent) points.
- Need to remove or ignore “outliers”.
- Calculate residuals ($r_i = y_i - (ax_i + b)$) and check if they are normally distributed.

The NumPy `ndarray` type

* `ndarray` is a sequence type.
* All values in an array must be of the same type.
* Typically numbers (integers, floating point or complex) or Booleans, but can be any type.

```
>>> import numpy as np
>>> x = np.array([1.0, 2, 3])
>>> x
array([ 1., 2., 3.])
>>> type(x)
<class 'numpy.ndarray'>
>>> x.dtype
dtype('float64')
```

Indexing & length

array:	`3.0`	`1.5`	`0.0`	`-1.5`	`-3.0`
index:	0	1	2	3	4
	-5	-4	-3	-2	-1

- In python, all sequences are indexed from 0.
- The index must be an integer.
- python also allows indexing from the sequence end using negative indices, starting with -1.
- The length of a sequence is the number of elements, *not* the index of the last element.

- `len(`*sequence*`)` returns sequence length.
- Sequence elements are accessed by writing the index in square brackets, `[]`.

```
>>> x = np.array([3, 1.5, 0, -1.5, -3])
>>> x[1]
1.5
>> x[-1]
-3.0
>>> len(x)
5
>>> x[5]
IndexError: index 5 is out of bounds
            for axis 0 with size 5
```

More operations on NumPy arrays

Creating 1-dimensional arrays

```
>>> np.zeros(5)
array([ 0., 0., 0., 0., 0.])
>>> np.ones(3) * 5
array([ 5., 5., 5.])
>>> np.linspace(3, -3, 5)
array([3. , 1.5, 0. , -1.5, -3. ])
>>> import numpy.random as rnd
>>> rnd.normal(0, 2, 10)
array([0.11224282, -1.84772958, ...
```

Element-wise operators

- Arithmetic (`+,-,*,/,**,//,%`), comparison (`==,!=,<,>,<=,>=`) and logical (`&,|`) operators can be applied to
  - an `ndarray` and a single value: the operation is done between each element of the array and the value; or
  - two `ndarray`s of the same size: the operation is done between pairs of elements in equal positions.

```
>>> x = np.array([-2.,-1.,0.,1.,2.])
>>> -(x ** 2) + 2
array([-2., 1., 2., 1., -2.])
>>> y = 2 ** x
>>> y
array([ 0.25, 0.5, 1., 2., 4.])
>>> x + y
array([ -1.75, -0.5, 1., 3., 6.])
>>> x + array([1., -1.])
ValueError: operands could not be
broadcast with shapes (5,) (2,)
```

* NumPy provides most math functions (e.g., `cos`, `exp`, `log`, `sqrt`, etc) that also work element-wise on arrays.

```
>>> x = np.linspace(-np.pi, np.pi, 9)
>>> np.cos(x)
array([-1.00e+00, -7.07e-01,  6.12e-17,
        7.07e-01,  1.00e+00,  7.07e-01,
        6.12e-17, -7.07e-01, -1.00e+00])
>>> np.sqrt(x)
RuntimeWarning:  invalid value ...
array([ nan, nan, 0., 1., 1.41421356])
```

Functions of arrays

```
>>> x = np.linspace(-1, 3, 5)
>>> np.min(x ** 2)
0.0
>>> np.max(x)
3.0
>>> np.sum(x)
5.0
>>> np.mean(x)
1.0
>>> np.std(x)
1.4142135623730951
```

Generalised indexing

- Most python sequence types support *slicing* – accessing a subsequence by indexing a range of positions:

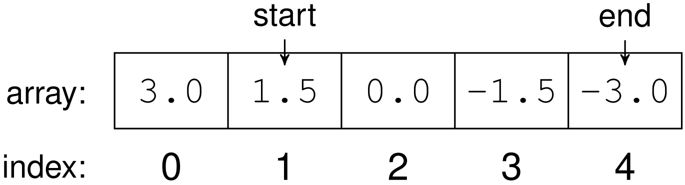
  `sequence[start:end]`

- Unique to NumPy array:
  - Indexing with an *array of integers* selects elements from the positions in the index array.
  - Indexing with an *array of Booleans* selects elements from the positions where the index array contains `True`.

Slicing

- The slice range is “half-open”: start index is included, end index is one after last included element.

```
>>> x = np.array([3, 1.5, 0, -1.5, -3])
>>> x[1:4]
array([ 1.5, 0, -1.5])
```



Indexing with arrays

```
>>> x = np.array([3,1.5,0,-1.5,-3])
>>> i = np.array([0,1,4])
>>> x[i]
array([ 3., 1.5., -3.])
>>> j = (x == np.floor(x))
>>> j
array([True, False, True, False, True])
>>> x[j]
array([ 3., 0., -3.])
```

```
# select "good" sample points:
ok = (y > np.min(y)) & (y < np.max(y))
# compute y and x difference:
dy = y[ok][1:] - y[ok][0:-1]
dx = x[ok][1:] - x[ok][0:-1]
# average slope:
a = np.mean(dy / dx)
# find average intercept:
b = np.mean(y[ok] - a * x[ok])
# compute residuals:
r = y[ok] - (a * x[ok] + b)
```

...or...

```
import scipy
ok = (y > np.min(y)) & (y < np.max(y))
# fit a 1st degree polynomial:
p = scipy.polyfit(x[ok], y[ok], 1)
# calculate r = y - p(x)
r = y[ok] - scipy.polyval(p, x[ok])
```