SciPy: Scientific Computing

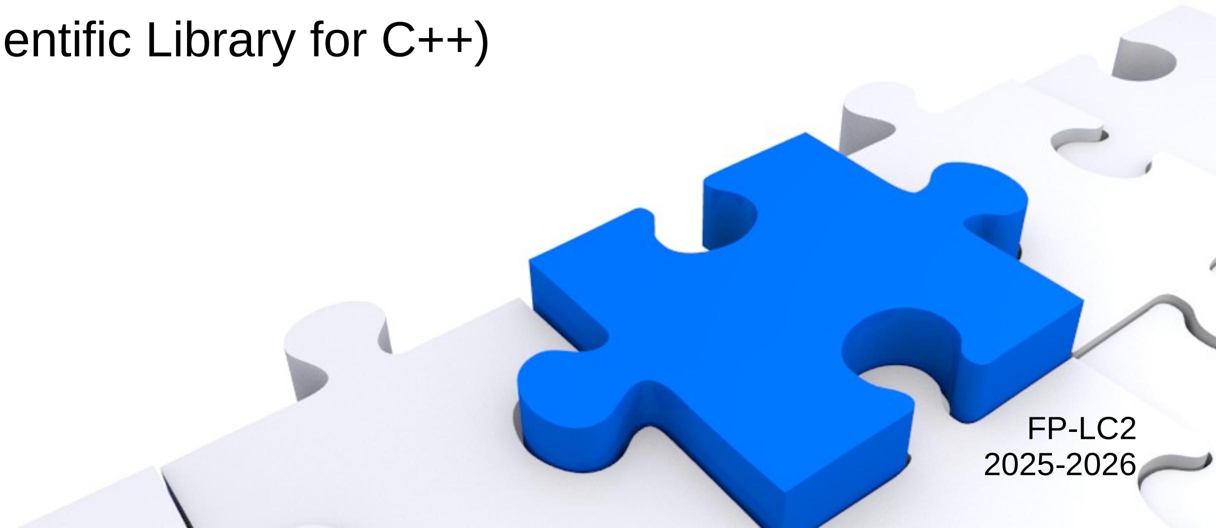
Hendrik Speleers

SciPy: Scientific Computing

- Overview
  - Collection of SciPy toolboxes
  - Interpolation
    - Spline interpolation and smoothing
  - Optimization
    - Local and global optimization
  - Linear algebra
    - Solving linear systems + eigenvalue problems
  - Statistics
    - Descriptive statistics + probability distributions + hypothesis tests

SciPy: Scientific Computing

- SciPy
  - Scientific Python
  - Collection of toolboxes
    - Dedicated to common issues in scientific computing
    - Examples: interpolation, optimization, linear algebra, image processing, ...
    - Scipy’s routines are optimized and tested
    - Similar to: Matlab’s toolboxes, GSL (GNU Scientific Library for C++)



SciPy: Scientific Computing

Università di Roma Tor Vergata

- SciPy

`scipy.cluster`	vector quantization / kmeans
`scipy.constants`	physical and mathematical constants
`scipy.fftpack`	Fourier transform
`scipy.integrate`	integration
`scipy.interpolate`	interpolation
`scipy.io`	data input and output
`scipy.linalg`	linear algebra
`scipy.ndimage`	n-dimensional images
`scipy.odr`	orthogonal distance regression
`scipy.optimize`	optimization
`scipy.signal`	signal processing
`scipy.sparse`	sparse matrices
`scipy.spatial`	spatial data structures and algorithms
`scipy.special`	special mathematical functions
`scipy.stats`	statistics

SciPy: Scientific Computing

Università di Roma Tor Vergata

- Interpolation
  - Module `interpolate`
    - Useful for fitting a function from experimental data
    - Based on the Fortran Library FITPACK

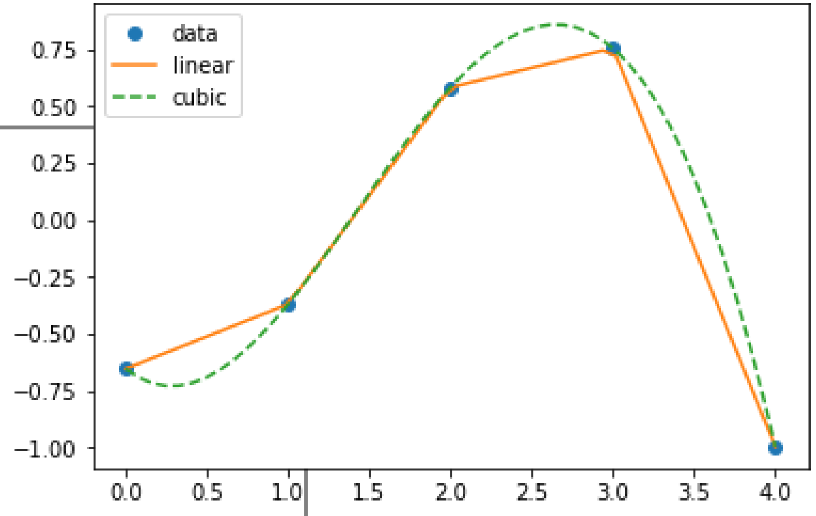
```
from scipy import interpolate
```

  - 1D data interpolation
    - Basic syntax: `f = interpolate.interp1d(x, y)`
    - Choose interpolant
      - Linear spline: `kind='linear'` (default)
      - Cubic spline: `kind='cubic'`

SciPy: Scientific Computing

- Interpolation
  - Example

```
In [1]: x = np.linspace(0.0, 4.0, 5)
   ...: y = np.cos(x**2 / 3.0 + 4.0)
In [2]: f1 = interpolate.interp1d(x, y)
In [3]: f2 = interpolate.interp1d(x, y,
   ...:                          kind='cubic')
In [4]: t = np.linspace(0.0, 4.0, 50)
   ...: plt.plot(x, y, 'o')
   ...: plt.plot(t, f1(t), '-')
   ...: plt.plot(t, f2(t), '--')
   ...: plt.legend(('data', 'linear', 'cubic'))
```



SciPy: Scientific Computing

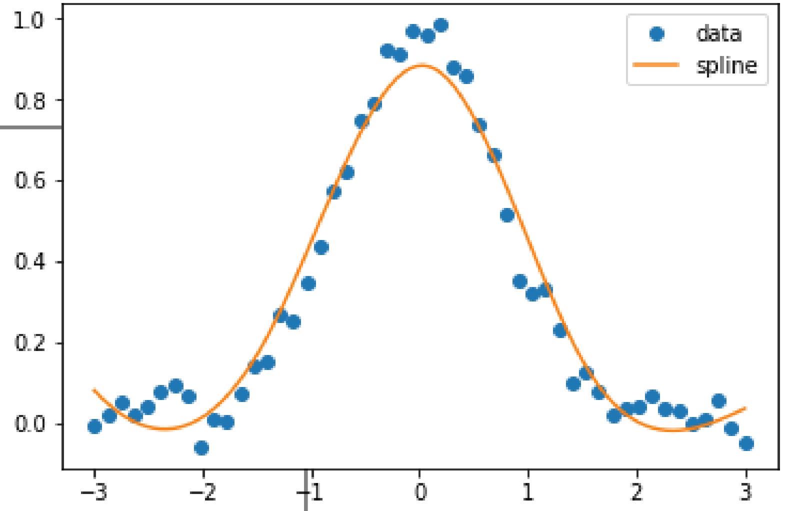
Università di Roma Tor Vergata

- Interpolation
  - 1D spline smoothing
    - Spline representation
      - `tck = interpolate.splrep(x, y, w=None, k=3, s=None, t=None)`
      - Input: data points `(x, y)`, weights `w`, degree `k`, smoothing factor `s`, knots `t`
      - Output: `tck = (t, c, k)` collects knots `t`, coefficients `c`, degree `k`
    - Spline evaluation
      - `y = interpolate.splev(x, tck, der=0)`
      - Input: points `x`, spline data `tck`, order of derivative `der`
      - Output: spline values `y`
    - Other: `interpolate.splint`, `interpolate.sproot`, ...

SciPy: Scientific Computing

- Interpolation
  - Example

```
In [1]: x = np.linspace(-3.0, 3.0, 50)
   ...: r = 0.05 * np.random.randn(50)
   ...: y = np.exp(-x**2) + r
In [2]: tck = interpolate.splrep(x, y,
   ...:                           s=0.2)
In [3]: t = np.linspace(-3.0, 3.0, 100)
   ...: spl = interpolate.splev(t, tck)
In [4]: plt.plot(x, y, 'o')
   ...: plt.plot(t, spl, '-')
   ...: plt.legend(('data', 'spline'))
```



SciPy: Scientific Computing

Università di Roma Tor Vergata

- Optimization
  - Module `optimize`

```
from scipy import optimize
```

  - 1D optimization
    - Basic syntax: `res = optimize.minimize_scalar(fun, bounds=None)`
    - Choose method
      - Brent method: `method='Brent'` (default)
      - Golden section: `method='Golden'`
      - Bounded Brent: `method='Bounded'`

- Optimization
  - Local optimization
    - `res = optimize.minimize(fun, x0, jac=None, hess=None, bounds=None)`
    - Choose method
      - Quasi-Newton method: `method='BFGS'` (default)
      - Conjugate gradients: `method='CG', 'Newton-CG'`
      - Trust-region methods: `method='trust-ncg', 'trust-krylov', 'trust-exact'`
      - ...
  - Global optimization
    - `res = optimize.basinhopping(fun, x0)`
    - Based on local method: `minimizer_kwargs={'method':'BFGS'}`

SciPy: Scientific Computing

Università di Roma Tor Vergata

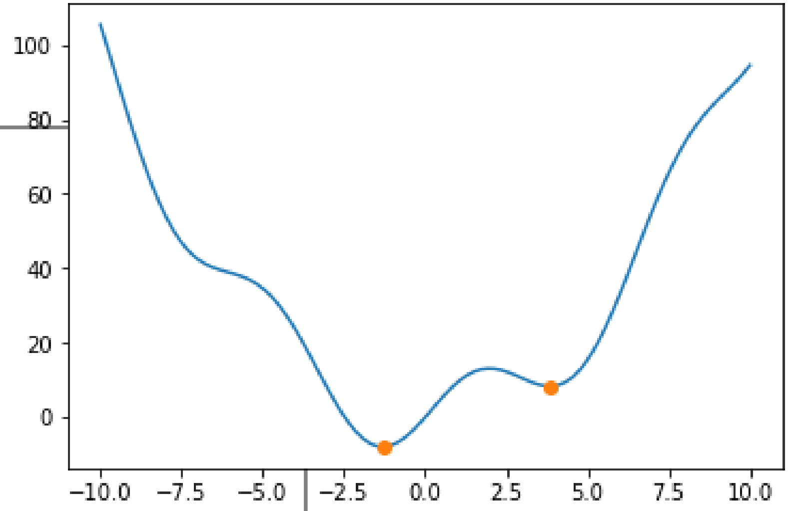
- Optimization
  - Linear programming
    - $\min_x \, c^T x$
      s.t. $A_{ub} x \le b_{ub}, \;\; A_{eq} x = b_{eq}, \;\; l \le x \le u$
    - ```
      res = optimize.linprog(c, A_ub=None, b_ub=None,
                                A_eq=None, b_eq=None, bounds=None)
      ```
    - Choose method
      - Quasi-Newton method: `method='interior-point'` (default)
      - Simplex method: `method='revised simplex', 'simplex'`
  - Other
    - Root finding: `optimize.root_scalar`, `optimize.root`
      ```

SciPy: Scientific Computing

Università di Roma Tor Vergata

- Optimization
  - Example

```
In [1]: def f(x):
   ...:     return x**2 + 10.0*np.sin(x)
In [2]: rg = optimize.minimize_scalar(f)
In [3]: rg.x
Out[3]: -1.3064400120612139
In [4]: rl = optimize.minimize_scalar(f,
   ...:      bounds=(2, 7), method='bounded')
In [5]: rl.x
Out[5]: 3.8374681582973356
```



SciPy: Scientific Computing

- Linear algebra
  - Module `linalg`
    - Standard linear algebra operations
    - Relying on efficient implementations (BLAS, LAPACK)

```
from scipy import linalg
```

  - Basic operations
    - Determinant: `linalg.det(a)`
    - Inverse: `linalg.inv(a)`
    - Norm: `linalg.norm(a, ord=None, axis=None)`

SciPy: Scientific Computing

- Linear algebra
  - Matrix decompositions
    - LU: `p, l, u = linalg.lu(a)`
    - Cholesky: `c = linalg.cholesky(a)`
    - QR: `q, r = linalg.qr(a)`
    - SVD: `u, s, v = linalg.svd(a)`
    - Schur: `t, z = linalg.schur(a)`
  - Special matrices
    - Block diagonal: `linalg.block_diag(*arrs)`
    - Toeplitz: `linalg.toeplitz(c, r=None)`
    - Hankel: `linalg.hankel(c, r=None)`

SciPy: Scientific Computing

- Linear algebra
  - Solve linear system *a * x = b*
    - Basic syntax: `x = linalg.solve(a, b)`
    - Choose solver by specifying type
      - Generic: `assume_a='gen'` (default) → LAPACK solver ?GESV
      - Symmetric: `assume_a='sym'` → LAPACK solver ?SYSV
      - Hermitian: `assume_a='her'` → LAPACK solver ?HESV
      - Positive definite: `assume_a='pos'` → LAPACK solver ?POSV
  - Other solvers
    - Banded: `x = linalg.solve_banded((l, u), ab, b)`
    - Triangular: `x = linalg.solve_triangular(a, b, lower=False)`

SciPy: Scientific Computing

Università di Roma Tor Vergata

- Linear algebra
  - Eigenvalue problem $a * v = w * b * v$
    - Eigenvalues: `w = linalg.eigvals(a, b=None)`
    - Eigenvectors: `w, v = linalg.eig(a, b=None)`
  - Symmetric eigenvalue problem
    - Symmetric: `w, v = linalg.eigh(a, b=None)`
    - Symmetric banded: `w, v = linalg.eig_banded(a_band, lower=False)`
  - Singular value problem
    - Singular values: `s = linalg.svdvals(a)`
    - SVD: `u, s, v = linalg.svd(a)`

SciPy: Scientific Computing

Università di Roma Tor Vergata

- Linear algebra
  - Example

```
In [1]: a = np.array([[1, 2], [3, 4]])
   ...: b = np.array([[5], [6]])
In [2]: linalg.inv(a).dot(b)    # slow
Out[2]: array([[-4. ],
   ...:        [ 4.5]])
In [3]: linalg.solve(a, b)   # fast
Out[3]: array([[-4. ],
   ...:        [ 4.5]])
In [4]: linalg.det(a)
Out[4]: -2.0000000000000004
```

Avoid inverting a matrix unless strictly needed!

SciPy: Scientific Computing

Università di Roma Tor Vergata

- Statistics
  - Module `stats`

```
from scipy import stats
```

  - Many other statistical modules in Python
    - **statsmodels**: regression, linear models, time series analysis
    - **Pandas**: tabular data, time series functionality, interfaces to other languages
    - **PyMC**: Bayesian statistical modeling, probabilistic machine learning
    - **Seaborn**: statistical data visualization

SciPy: Scientific Computing

Università di Roma Tor Vergata

- Statistics
  - Descriptive statistics
    - Stats summary: `stats.describe(a, axis=0)`
    - Min and max: `np.min(a, axis=None), np.max(a, axis=None)`
    - Basic stats: `np.mean(a, axis=None), np.var(a, axis=None)`
    - Means: `stats.gmean(a, axis=0), stats.hmean(a, axis=0)`
    - Modal value: `stats.mode(a, axis=0)`
    - Skewness: `stats.skew(a, axis=0)`
    - Kurtosis: `stats.kurtosis(a, axis=0)`
    - Z score: `stats.zscore(a, axis=0)`
    - Standard error of mean: `stats.sem(a, axis=0)`

SciPy: Scientific Computing

- Statistics
  - Normal distribution

```
from scipy.stats import norm
```

- Random variates: `norm.rvs(loc=0, scale=1, size=1)`
- Probability density function: `norm.pdf(x, loc=0, scale=1)`
- Cumulative distribution function: `norm.cdf(x, loc=0, scale=1)`
- Survival function: `norm.sf(x, loc=0, scale=1)`
- Moments: `norm.stats(loc=0, scale=1, moments='mv')`
- Parameter estimates: `norm.fit(data)`

SciPy: Scientific Computing

Università di Roma Tor Vergata

- Statistics
  - Uniform distribution

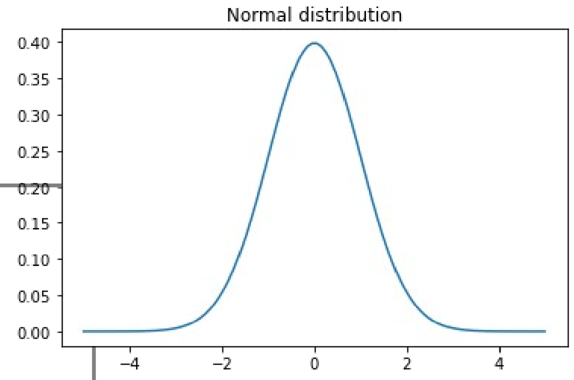
```
from scipy.stats import uniform
```

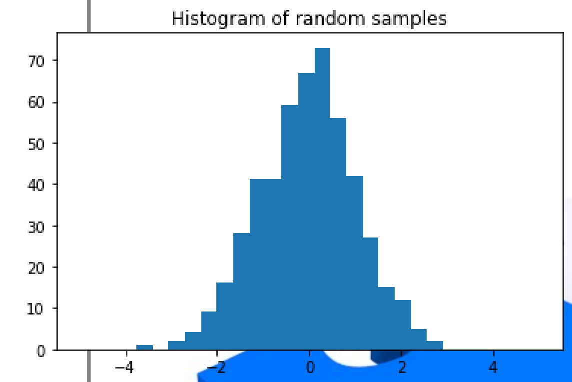
- Random variates: `uniform.rvs(loc=0, scale=1, size=1)`
- Probability density function: `uniform.pdf(x, loc=0, scale=1)`
- Cumulative distribution function: `uniform.cdf(x, loc=0, scale=1)`
- Survival function: `uniform.sf(x, loc=0, scale=1)`
- Moments: `uniform.stats(loc=0, scale=1, moments='mv')`
- Parameter estimates: `uniform.fit(data)`

SciPy: Scientific Computing

- Statistics
  - Example

```
In [1]: x = np.linspace(-5, 5, 100)
In [2]: pdf = norm.pdf(x)
In [3]: plt.plot(x, pdf)
   ...: plt.title('Normal distribution')
In [4]: var1, skew1 = norm.stats(moments='vs')
In [4]: r = norm.rvs(size=500)
In [5]: plt.hist(r, bins='auto')
   ...: plt.title('Histogram of random samples')
In [6]: var2, skew2 = np.var(r), stats.skew(r)
```





SciPy: Scientific Computing

Università di Roma Tor Vergata

- Statistics
  - Student's T-tests
    - One sample test: `stats.ttest_1samp(a, popmean, axis=0)`
    - Independent samples test: `stats.ttest_ind(a, b, axis=0)`
  - Example

```
In [1]: rvs1 = norm.rvs(loc=5, scale=2, size=500)
   ...: rvs2 = norm.rvs(loc=5, scale=2, size=500)
In [2]: stats.ttest_ind(rvs1, rvs2)
Out[2]: Ttest_indResult(statistic=0.0301384142742
   ...: 89083, pvalue=0.9759626902441914)
```