Manipulating Polynomials in Python with SciPy

R.G. Erdmann
In science and engineering computing, one frequently needs to manipulate polynomials in various ways:

- Evaluation at a given point
- Scalar-polynomial and polynomial-polynomial operations
  - Addition
  - Subtraction
  - Multiplication
  - Division
  - Multiplication
  - Powers
- Differentiation
- Integration
Scipy provides a class for manipulation of arbitrary-order univariate polynomials capable of all of these operations. **Name:** scipy.poly1d
A nth-order polynomial

\[ p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 = \sum_{i=0}^{n} a_i x^i \]

is represented as a list of the coefficients

\[ [a_n, a_{n-1}, \ldots, a_1, a_0] \]

and constructed as

\[ \text{poly1d}([a_n, a_{n-1}, \ldots, a_1, a_0]) \]
The polynomial

\[ p(x) = \frac{x^4}{5} + 3x^2 - 7x + 5 \]

is constructed as

\[ \text{poly1d}([0.2, 0, 3, -7, 5]) \]
Polynomials can be evaluated at a single value of $x$ or at multiple $x$ values simultaneously.

$$p(x) = x + 1$$

```python
>>> from scipy import *
>>> p = poly1d([1, 1])
>>> p(3)
4
>>> p([-2, -1, 0, 1, 2])  # evaluated over a list
array([-1, 0, 1, 2, 3])
>>> x = linspace(-2, 2, 5)
>>> p(x)  # evaluated with array as input
array([-1., 0., 1., 2., 3.])
```
Addition, subtraction, multiplication, division, and powers are defined between `poly1d` and scalars:

\[ p(x) = x + 1 \]

>>> p = poly1d([1, 1])  # \( p(x) = x + 1 \)
>>> p+2
poly1d([1, 3])  # \( p(x) + 2 = x + 3 \)
>>> 3*p
poly1d([3, 3])  # \( 3p(x) = 3x + 3 \)
>>> p/2
poly1d([ 0.])  # careful! integer division!
>>> p/2.
poly1d([ 0.5, 0.5])  # \( p(x)/2 = x/2 + 1/2 \)
>>> p**2  # power to non-negative integers only
poly1d([1, 2, 1])  # \( p^2(x) = x^2 + 2x + 1 \)
Addition, subtraction, multiplication, and division between poly1d objects:

```python
>>> p = poly1d([1, 1])  # \( p(x) \equiv x + 1 \)
>>> q = poly1d([2, -1, 3])  # \( q(x) \equiv 2x^2 - x + 3 \)
>>> p+q
poly1d([2, 0, 4])  # \( p(x) + q(x) = 2x^2 + 4 \)
>>> p-q
poly1d([-2, 2, -2])  # \( p(x) - q(x) = -2x^2 + 2x - 2 \)
>>> p*q
poly1d([2, 1, 2, 3])  # \( p(x) \times q(x) = 3x^3 + x^2 + 2x + 3 \)
>>> q/p  # quotient and remainder
(poly1d([ 2., -3.]), poly1d([6]))  # \( (2x - 3) \times p(x) + 6 = q(x) \)
>>> poly1d([ 2., -3.]) * p + poly1d([6])
poly1d([ 2., -1., 3.])
```
Differentiation

\[ q(x) = 2x^2 - x + 3 \]
\[ q'(x) = 4x - 1 \]
\[ q''(x) = 4 \]

```python
>>> q = poly1d([2, -1, 3])
>>> q.deriv()
poly1d([ 4, -1])
>>> q.deriv(2)  # n-th derivative
poly1d([4])
```
Integration

\[ q(x) = 2x^2 - x + 3 \]

\[
\int q(x) \, dx = \frac{2x^3}{3} - \frac{x^2}{2} + 3x + C_1
\]

\[
\int \int q(x) \, dx \, dx = \frac{x^4}{6} - \frac{x^3}{6} + \frac{3x^2}{2} + C_1 x + C_2
\]

>>> q.integ()
poly1d([ 0.66666667, -0.5, 3., 0.])

>>> q.integ(k=2) # set \( C_1 = 2 \)
poly1d([ 0.66666667, -0.5, 3., 2.])

>>> q.integ(m=2, k=[2, 1]) # set \( C_1 = 2, \ C_2 = 1 \)
poly1d([ 0.16666667, -0.16666667, 1.5, 2., 1.])
The coefficients for a least-squares polynomial fit to $x$-$y$ data can be obtained with `polyfit`, which is suitable for construction of `poly1d` objects:

```python
>>> x = [0., 1., 2., 3., 4.]
>>> y = [1., 2., 5., 10., 17.]
>>> p = poly1d(polyfit(x, y, 2))  # 2nd-order polynomial
>>> p
poly1d([ 1., -0., 1.])
>>> print p
  2
1 x - 2.974e-16 x + 1
```
The roots of a polynomial can be obtained with the `roots` function, which returns an array of the (possibly complex) roots:

```python
# print very small values in arrays as 0
>>> set_printoptions(suppress=True)
# make \( p(x) = x^3 + x^2 + x + 1 \)
>>> p = poly1d([1, 1, 1, 1])
>>> r = roots(p)
>>> print r # note: complex roots
[-1.+0.j -0.+1.j -0.-1.j]
>>> p(r) # evaluate \( p \) at each of the roots
array([-0.+0.j, 0.-0.j, 0.+0.j])
```