Интерполяция

Полином какой степени нужен?

Полином степени n - это функция $Pn(x) = a_n*x^n + a_{n-1}*x^{n-1} + ... + a_2*x^2 + a_1*x + a_0$
Возьмем точку на плоскости. Через нее можно провести единственную прямую вида y = a. Кривых бОльших порядков можно провести неограниченно много (наклонные прямые, параболы и тп).
Возьмем 2 точки на плоскости. Через них можно провести единственную прямую вида y = ax+b. Провести через произвольные 2 точки прямую вида y=a невозможно, а парабол через 2 точки можно провести сколь угодно много.
Для N точек нужно взять интерполяционный полином степени N-1.

Import

```
In [1]: from scipy import interpolate
In [2]: from numpy import polynomial as P
In [3]: import numpy as np
In [4]: from scipy import linalg
In [5]: import matplotlib.pyplot as plt
```

Полиномы

В **numpy** есть пакет polynomial, который мы импортировали как P.

Зададим полином

В примерах задаем полином $1+2x+3x^2$
Вариант 1: задаем через список коэффициентов a0, a1, ... an.

```
In [6]: p1 = P.Polynomial([1, 2, 3])
In [7]: p1
Out[7]: Polynomial([ 1., 2., 3.], [-1, 1], [-1, 1])
```

Вариант 2: задаем корни уравнения f(x) = 0. Например, x = -1 и x = 1

```
In [8]: p2 = P.Polynomial.fromroots([-1, 1])
In [9]: p2
Out[9]: Polynomial([-1., 0., 1.], [-1., 1.], [-1., 1.])
```

Трансформации

Не обязательный материал. Подробнее смотрите в документации.
Polynomial([-1., 0., 1.], [-1., 1.], [-1., 1.])
В полном описании полинома кроме коэффициентов есть еще два параметра: domain и window. Они задают как с помощью масштабирования и трансформации исходный интервал [domain[0], domain[1]] превращается в результирующий интервал [window[0], window[1]].
Коэффициенты, исходный и результирующий интервал можно получить из полинома с помощью методов:

```
In [12]: p1.coef
Out[12]: array([ 1., 2., 3.])
In [13]: p1.domain
Out[13]: array([-1, 1])
In [14]: p1.window
Out[14]: array([-1, 1])
```

Найдем корни полинома roots()

```
In [10]: p1.roots()
Out[10]: array([-0.33333333-0.47140452j, -0.33333333+0.47140452j])
In [11]: p2.roots()
Out[11]: array([-1., 1.])
```

Вычислим значения полинома в точках

Найдем значения полинома в нескольких точках. Например, в точках 1.5, 2.5, 3.5.
Просто вызовем p1 от списка точек и получим массив значений.

```
In [15]: p1([1.5, 2.5, 3.5])
Out[15]: array([ 10.75, 24.75, 44.75])
```

Арифметические операции над полиномами

Для полиномов определены арифметические операции +, -, *, / и так далее.
Операция // используется для полиномиального деления.
Пусть один полином будет p1(x) = (x-3)(x-2)(x-1), другой p2(x) = (x-2). Тогда деление одного полинома на другой будет p3 = (x-3)(x-2)(x-1)/(x-2) = (x-3)(x-1)
Убедимся, что это так. Зададим полиномы через их корни. Разделим один на другой и найдем корни полученного полинома.

```
In [16]: p1 = P.Polynomial.fromroots([1, 2, 3])
In [17]: p1
Out[17]: Polynomial([ -6., 11., -6., 1.], [-1., 1.], [-1., 1.])
In [18]: p2 = P.Polynomial.fromroots([2])
In [19]: p2
Out[19]: Polynomial([-2., 1.], [-1., 1.], [-1., 1.])
In [20]: p3 = p1 // p2
In [21]: p3
Out[21]: Polynomial([ 3., -4., 1.], [-1., 1.], [-1., 1.])
In [22]: p3.roots()
Out[22]: array([ 1., 3.])
```

Другие типы многочленов

В дополнение к классу Polynomial (хранящему многочлены в стандартном базисе по степеням X) в модуле polynomial также есть классы для представления в базисах

- Чебышева (класс Chebyshev),
- Лагранжа (класс Legendre),
- Лаггера (класс Laguerre),
- Эрмита (класс Hermite - физическое определение и HermiteE? - вероятностное определение).

Аналогично можем задать многочлен Чебышева с коэффициентами 1, 2, 3, т.е. полином $1T_1+2T_2+3T_3$, где $T_i(x)$ - многочлен Чебышева порядка i

```
In [23]: c1 = P.Chebyshev([1, 2, 3])
In [24]: c1
Out[24]: Chebyshev([ 1., 2., 3.], [-1, 1], [-1, 1])
```

вычислить его корни

```
In [25]: c1.roots()
Out[25]: array([-0.76759188, 0.43425855])
```

Все многочлены других классов имеют те же методы, что и класс Polynomial, который описан выше. Используют их тоже аналогично.
Задать многочлены из корней:

```
In [26]: c1 = P.Chebyshev.fromroots([-1, 1])
In [27]: c1
Out[27]: Chebyshev([-0.5, 0. , 0.5], [-1., 1.], [-1., 1.])
In [28]: l1 = P.Legendre.fromroots([-1, 1])
In [29]: l1
Out[29]: Legendre([-0.66666667, 0. , 0.66666667], [-1., 1.], [-1., 1.])
```

Подсчитать значения многочленов в точках:

```
In [30]: c1([0.5, 1.5, 2.5])
Out[30]: array([-0.75, 1.25, 5.25])
In [31]: l1([0.5, 1.5, 2.5])
Out[31]: array([-0.75, 1.25, 5.25])
```

Интерполяция алгебраическими многочленами

Теория

Классы многочленов, описанные выше, дают полезные функции для интерполяции многочленами. Например, решим линейное уравнение для задачи интерполяции F(x)c = y, где x и y - вектора с координатами xi и yi. Для решения задачи интерполяции мы должны вычислить матрицу F(x) в выбранном базисе и далее решить систему линейных уравнений. Каждый класс многочленов имеет функции для вычисления (обобщенной) матрицы Вандермонда для соответствующего базиса.

Например, для степенного базиса мы используем np.polynomial.polynomial.polyvander, для базиса из многочленов Чебышева берем функцию np.polynomial.chebyshev.chebvander и так далее.

Полный список функций приведен в документации по модулю np.polynomial и его подмодулям.

Используя эти функции для создания матриц Вандерморта, мы можем легко представить интерполяционный многочлен в любом базисе.

Например, рассмотрим набор точек с координатами (1, 1), (2, 3), (3, 5) и (4, 4). Создадим соответствующие NumPy? массивы для x и y координат:

```
In [32]: x = np.array([1, 2, 3, 4])
In [33]: y = np.array([1, 3, 5, 4])
```

Для интерполяции многочленом через эти точки, надо использовать полином 3 степени (количество точек - 1). Для интерполяции в степенном базисе найдем коэффициенты c_i такие что $f(x) = c_1x^0+c_2x^1+c_3x^2+c_4x^3$ Чтобы найти эти коэффициенты, вычислим матрицу Вандермонда и решим систему интерполяционных уравнений:

```
In [34]: deg = len(x) - 1
In [35]: A = P.polynomial.polyvander(x, deg)
In [36]: c = linalg.solve(A, y)
In [37]: c
Out[37]: array([ 2. , -3.5, 3. , -0.5])
```

Получен вектор коэффициентов [2, -3.5, 3, -0.5], то есть интерполяционный полином $f(x) = 2 - 3.5x + 3x^2 - 0.5x^3$.

Используем найденный массив коэффициентов для создания интерполяционного полинома и дальнейшего вычисления значения в точке:

```
In [38]: f1 = P.Polynomial(c)
In [39]: f1(2.5)
Out[39]: 4.1875
```

Найдем интерполяционный многочлен в другом базисе, например, многочленов Чебышева:

```
In [40]: A = P.chebyshev.chebvander(x, deg)
In [41]: c = linalg.solve(A, y)
In [42]: c
Out[42]: array([ 3.5 , -3.875, 1.5 , -0.125])
```

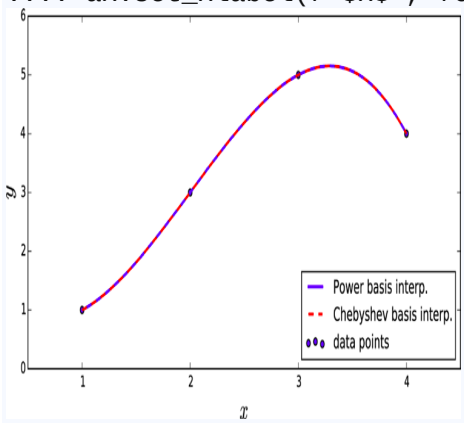
Как и ожидалось, в другом базисе получились другие коэффициенты и интерполяционный многочлен в базисе Чебышева $f(x) = 3.5T_0(x)- 3.875T_1(x)+1.5T_2(x)- 0.125T_3(x)$ Однако, не зависимо от базиса, интерполяционный многочлен единственный и вычисляя значение в точке получаем такое же число:

```
In [43]: f2 = P.Chebyshev(c)
In [44]: f2(2.5)
Out[44]: 4.1875
```

Посмотрим на эти два полинома вместе на одном графике:

```
In [45]: xx = np.linspace(x.min(), x.max(), 100) # supersampled [x[0], x[-1]]
interval
In [45]: fig, ax = plt.subplots(1, 1, figsize=(12, 4))
...: ax.plot(xx, f1(xx), 'b', lw=2, label='Power basis interp.')
...: ax.plot(xx, f2(xx), 'r--', lw=2, label='Chebyshev basis interp.')
...: ax.scatter(x, y, label='data points')
...: ax.legend(loc=4)
...: ax.set_xticks(x)
```

```
...: ax.set_ylabel(r"$y$", fontsize=18)
...: ax.set_xlabel(r"$x$", fontsize=18)
```



Это не очень удобно. Хотим увеличить порядок полинома - нужно переписывать уравнение или программировать на эту тему какую-то автоматику. Может, есть что готовое?
Используем метод fit для степенного полинома и полинома Чебышева:

```
In [46]: f1b = P.Polynomial.fit(x, y, deg)
In [47]: f1b
Out[47]: Polynomial([ 4.1875, 3.1875, -1.6875, -1.6875], [ 1., 4.], [-1., 1.])

In [48]: f2b = P.Chebyshev.fit(x, y, deg)
In [49]: f2b
Out[49]: Chebyshev([ 3.34375 , 1.921875, -0.84375 , -0.421875], [ 1., 4.], [-
1., 1.])
```

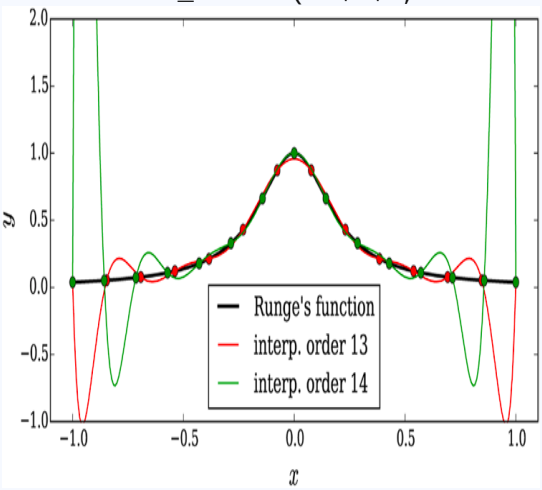
Заметим, что этот метод автоматически устанавливает атрибут domain в правильное значение (в примере отрезок был от 1 до 4), коэффициенты устанавливаются тоже соответствующим образом.
Отображение интерполяционных данных в отрезок, который лучше подходит для выбранного базиса, может существенно улучшить численную стабильность интерполяции. Например, использование полиномов Чебышева на отрезке [-1,1] а не на заданном [1,4] уменьшает число обусловленности с почти 4660 до 1.85:

```
In [50]: np.linalg.cond(P.chebyshev.chebvander(x, deg))
Out[50]: 4659.7384241399586
In [51]: np.linalg.cond(P.chebyshev.chebvander((2*x-5)/3.0, deg))
Out[51]: 1.8542033440472896
```

? откуда взялись эти x и (2*x-5)/3.0
Интерполяция многочленами небольшого количества точек - мощный и полезный математический инструмент. Но если количество точек растет, то растет и наибольшая степень интерполяционного многочлена, что приводит к проблемам.
Полученная функция резко изменяется за пределами отрезка интерполяции. Но что более неприятно, полиномы высокой степени нехорошо себя ведут и между точками интерполяции. Хотя в точках значения полиномов разных степеней совпадают, но между ними начинают появляться биения.
Возьмем функцию Рунге f (x) = 1/(1+25x^2), достаточно ровную на интервале [-1,1]. Результат интерполяции почти расходится в промежутке между точками у конца интервала интерполяции.
Для иллюстрации этого поведения напишем функцию runge, которая реализует функцию Рунге, и функцию runge_interpolate, которая находит интерполяцию n-той степени в степенном базисе.

```
In [52]: def runge(x):
...: return 1/(1 + 25 * x**2)
In [53]: def runge_interpolate(n):
...: x = np.linspace(-1, 1, n)
```

```
...: p = P.Polynomial.fit(x, runge(x), deg=n)
...: return x, p
```
Нарисуем графики исходной функции и интерполяционного многочлена 13 и 14 порядков:
```
In [54]: xx = np.linspace(-1, 1, 250)
In [55]: fig, ax = plt.subplots(1, 1, figsize=(8, 4))
...: ax.plot(xx, runge(xx), 'k', lw=2, label="Runge function")
...: # 13th order interpolation of the Runge function
...: n = 13
...: x, p = runge_interpolate(n)
...: ax.plot(x, runge(x), 'ro')
...: ax.plot(xx, p(xx), 'r', label='interp. order %d' % n)
...: # 14th order interpolation of the Runge function
...: n = 14
...: x, p = runge_interpolate(n)
...: ax.plot(x, runge(x), 'go')
...: ax.plot(xx, p(xx), 'g', label='interp. order %d' % n)
...: ax.legend(loc=8)
...: ax.set_xlim(-1.1, 1.1)
...: ax.set_ylim(-1, 2)
...: ax.set_xticks([-1, -0.5, 0, 0.5, 1])
...: ax.set_ylabel(r"$y$", fontsize=18)
...: ax.set_xlabel(r"$x$", fontsize=18)
```



Заметим, что у нас появляются сильные биения между точками на границах интервала интерполяции. Что делает такие функции неприемлемыми для задачи интерполяции. Для решения проблемы возьмем полиномы небольшой степени, если интерполяция считается по большому количеству исходных точек. Другими словами, вместо подгонки одного полинома под множество точек, соединим две соседние точки низкоуровневым многочленом. Такой подход называется сплайн-интерполяция или интерполяция сплайнами.

Интерполяция сплайнами

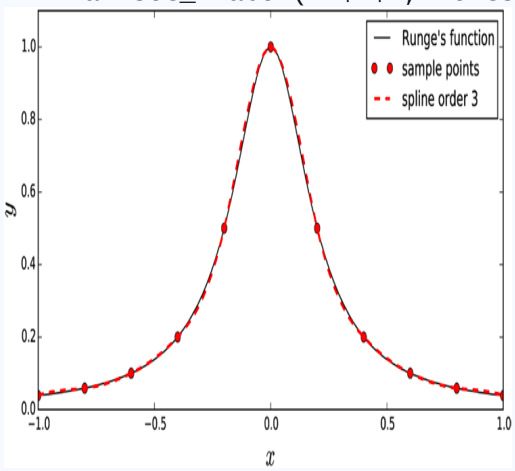
Для набора из n точек $\{x_i, y_i\}$, на интервале $[x_0, x_{n-1}]$ получим n-1 подинтервал $[x_i, x_{i+1}]$. Внутренняя точка, которая соединяет два подинтервала, будем называть узлом в терминах кусочно-полиномиальной интерполяции. Для интерполяции n точек, будем использовать кусочное полиномы степени k на каждом подинтервале, т.е. мы должны определить (k+1)(n-1) неизвестных параметра. Значения в узлах дают 2(n-1) уравнения. Этих уравнений недостаточно для нахождения всех неизвестных. Но требование непрерывности производных более высоких порядков дадут нам необходимые уравнения. Это условие гарантирует, что кусочно-полиномиальный интерполянт будет достаточно гладким.

Сплайн - это специальный тип кусочно-полиномиального интерполянта: кусочный полином степени k является сплайном, если он непрерывно дифференцируем k-1 раз. Наиболее популярным является сплайн 3 степени, k=3, который требует 4(n-1) параметр. В этом случае непрерывность двух производных в n-2 точках дает 2(n-2) дополнительных уравнения, повышая общее количество уравнений до 2 (n -1)+ 2(n - 2) = 4(n -1)- 2. Остается 2 неопределенных параметра, которые мы должны определить другим способом. Обычно задают дополнительное требование, чтобы производная второго порядка на концах точек была равна 0 (превращая в настоящий сплайн). Это дает еще 2 уравнения. Модуль интерполяции SciPy? предоставляет несколько функций и классов для представления сплайн-интерполяции. Например, мы можем использвать функцию interpolate.interp1d, которая берет массивы x и y для точек как первый и второй аргументы. Дополнительные аргументы позволяют задать тип и порядок интерполяции. Обычно задают kind=3 (или, что тоже самое, kind='cubic') для вычисления кубической интерполяции. Эта функция возвращает класс объектов, которые можно вызывать как функцию и которая может вычислять значения для различных значений x, используя вызовы функции.
Альтернативная сплайн-функция - это interpolate.InterpolatedUnivariateSpline, которая так же принимает массивы x и y в качестве первых двух аргументов, но которая использует аргумент k (вместо kind) для задания порядка сплайнов.
Посмотрим, как можно использовать функцию interpolate.interp1d для фунцкии Рунгде и возьмем сплайны 3 степени. Для этого мы сначала создадим массивы NumPy? для x и y координат в тех же точках. Дальше вызовем interpolate.interp1d с k=3 для получения кубических сплайнов на заданных данных.

```
In [56]: x = np.linspace(-1, 1, 11)
In [57]: y = runge(x)
In [58]: f_i = interpolate.interp1d(x, y, kind=3)
```

Для того, чтобы посмотреть насколько хороши сплайны, нарисуем исходную функцию Рунге, точки и кубические сплайны.

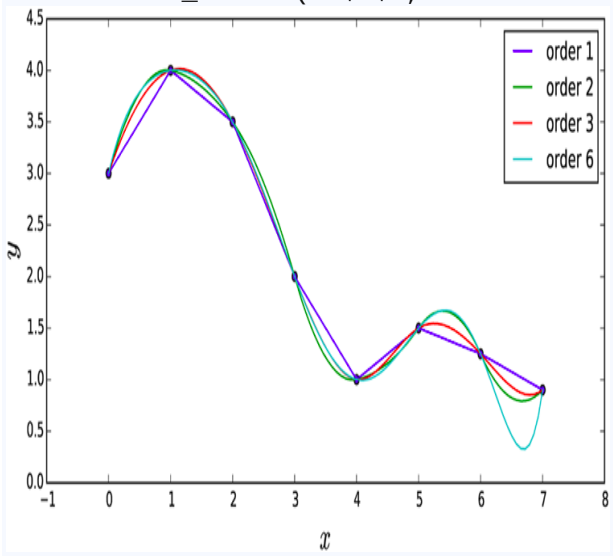
```
In [59]: xx = np.linspace(-1, 1, 100)
In [60]: fig, ax = plt.subplots(figsize=(8, 4))
...: ax.plot(xx, runge(xx), 'k', lw=1, label="Runges function")
...: ax.plot(x, y, 'ro', label='sample points')
...: ax.plot(xx, f_i(xx), 'r--', lw=2, label='spline order 3')
...: ax.legend()
...: ax.set_xticks([-1, -0.5, 0, 0.5, 1])
...: ax.set_ylabel(r"$y$", fontsize=18)
...: ax.set_xlabel(r"$x$", fontsize=18)
```



Здесь мы используем 11 заданных точек и сплайн 3 порядка. Заметим, что интерполянт очень хорошо совпадает с оригинальной функцией. Обычно сплайн-интерполяция порядка 3 или меньше не подвержена биениям, которые мы наблюдали в полиномах

высокой степени. И достаточно использовать сплайны 3 порядка, если у нас достаточно много данных.
Для иллюстрации влияния порядка сплайна, решим задачу интерполяции на точках (0,3), (1, 4), (2, 3.5), (4, 2), (5, 1.5), (6, 1.25) и (7, 0.7) со сплайнами возрастающей степени. Сначала определим массивы x и y, далее в цикле будем изменять требуемый порядок интерполяции, вычисляя интерполянт и рисуя его:

```
In [61]: x= np.array([0, 1, 2, 3, 4, 5, 6, 7])
In [62]: y= np.array([3, 4, 3.5, 2, 1, 1.5, 1.25, 0.9])
 In [63]: xx = np.linspace(x.min(), x.max(), 100)
In [64]: fig, ax = plt.subplots(figsize=(8, 4))
...: ax.scatter(x, y)
...:
...: for n in [1, 2, 3, 6]:
...:     f = interpolate.interp1d(x, y, kind=n)
...:     ax.plot(xx, f(xx), label='order %d' % n)
...:
...: ax.legend()
...: ax.set_ylabel(r"$y$", fontsize=18)
...: ax.set_xlabel(r"$x$", fontsize=18)
```



Можно проще?

Можно. Если точек немного используйте интерполяцию (степенную) модуля sympy:
http://docs.sympy.org/latest/modules/polys/reference.html
Для точек {(1,1), (2, 4), (3, 9), (4, 16)}

```
>>> interpolate([1, 4, 9, 16], x)
x**2
```

или явно задайте список координат:

```
>>> interpolate([(1, 1), (2, 4), (3, 9)], x)
x**2
```

x и y координаты могут задаваться как ключ и значение словаря.

```
>>> interpolate([(-1, 2), (1, 2), (2, 5)], x)
x**2 + 1
>>> interpolate({-1: 2, 1: 2, 2: 5}, x)
x**2 + 1
```

Для конструирования полиномов Лагранжа по n точкам используйте

```
  sympy.polys.specialpolys.interpolating_poly(n, x, X='x', Y='y')[source]
```