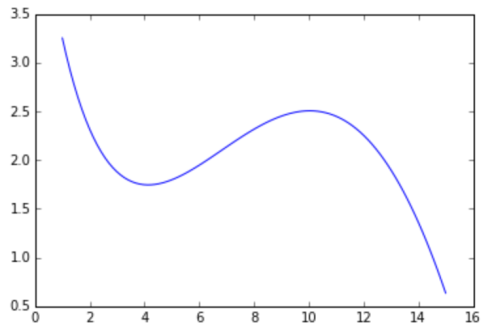
**Метод наименьших квадратов**

Рассмотрим сложную математическую функцию на отрезке :





Задача состоит в том, чтобы приблизить сложную зависимость с помощью функции из определенного семейства. В этом задании мы будем приближать указанную функцию с помощью многочленов.

Как известно, многочлен степени  (то есть ) однозначно определяется любыми  различными точками, через которые он проходит. Это значит, что его  неизвестных коэффициентов  можно определить из следующей системы линейных уравнений:

 (1)

где через  обозначены точки, через которые проходит многочлен, а через  — значения, которые он должен принимать в этих точках.

**1. Полиномиальная регрессия (5 баллов)**

Воспользуемся описанным свойством и будем находить приближение функции многочленом, решая систему линейных уравнений (1). Сформируйте систему линейных уравнений (то есть задайте матрицу коэффициентов  и вектор правых частей ) для многочлена, который должен совпадать с функцией  в  точках, эквидистантно расположенных на отрезке  (для этого удобно использовать функцию np.linspace()). Решите данную систему с помощью функции np.linalg.solve().

Постройте графики 1) исходной функции, 2) точек, по которым строился многочлен, 3) полиномиальную аппроксимацию.

Хорошо ли полином приближает исходную функцию?

Рассмотрите несколько различных значений . Как меняется приближение с ростом числа точек (и соответственно степени полинома)? Хорошо ли ведёт себя полином при ? А при ?

**2. Полиномиальная регрессия с зашумлённым данными (5 баллов)**

Рассмотрим случай, когда значение функции  известно с некоторой неточностью. Для этого добавьте к значениям функции  случайный шум. Шум сгенерируйте из гауссового распределения с нулевым средним и стандартным отклонением .

Повторите те же шаги, что и в пункте 1 для многочленов различной степени. Как изменилась полиномиальная аппроксимация в этом случае? Хорошо ли ведёт себя полином при ? А при ?

**3. Метод наименьших квадратов (5 баллов)**

Третья часть задания посвящена собственно *методу наименьших квадратов*.

Выше мы строили полиномиальную регрессию той степени, которая соответствовала числу точек. Т.е. при количестве точек равных  мы получали полином степени . На практике обычно ограничиваются полиномами степени не выше третьей, независимо от количества имеющихся точек.

Рассмотрим теперь случай, когда точек много. Пусть количество точек равно , а полином по-прежнему имеет вид , причём . Тогда в системе линейных уравнений вида (1) количество уравнений будет больше, чем число неизвестных. Подобного рода система уравнений может не иметь точного решения, особенно в случае, когда правая часть является приближённой, т.е. с шумом.

В таких случаях, вместо точного решения системы уравнений обычно решают задачу минимизации невязки уравнений:

 (2)

где в качестве нормы  обычно используют квадратичную норму. Поэтому такой подход называют методом наименьших квадратов.

Для решения данной задачи (2) используйте готовую функцию np.linalg.lstsq(), которая аналогично функции np.linalg.solve() принимает два аргумента: матрицу коэффициентов  и вектор правых частей . Данная функция возвращает несколько объектов, первый из которых есть решение задачи.

Возьмите  точек на отрезке  и посчитайте в них значение функции , добавьте гауссов с большим шумом с . Постройте методом наименьших квадратов кубический полином по данным точкам.

Постройте графики 1) исходной функции, 2) точек, по которым строился многочлен, 3) полиномиальную аппроксимацию.

Сделайте выводы по всей работе.

Решение в формате Jupyter notebook.