Python勉強会@HACHINONE

第15章

実験データの理解

お知らせ

Python勉強会@HACHINOHEでは、ジョン・V・グッターグ『Python言語による
プログラミングイントロダクション』近代科学社、2014年をみんなで勉強しています。

この本は自分で読んで考えて調べると力が付くように書かれています。

自分で読んで考えて調べる前に、このスライドを見るのは、いわばネタバレを
聞かされるようなものでもったいないです。

是非、本を読んでからご覧ください。

実験データをどのように理解すべきか？

Python勉強会@HACHINOHE

- 実験データ ←→ 何らかの理論
  - 理論的に導かれた関数のフィッテイング
  - 実験データに含まれる誤差
  - 理論の適用範囲
- バネののび
- 発射体の軌跡
- 指数関数へのフィッティング
- その他

バネののび

Python勉強会@HACHINOHE

- フックの法則: バネののびxは、力Fに比例し、比例定数はバネ定数k

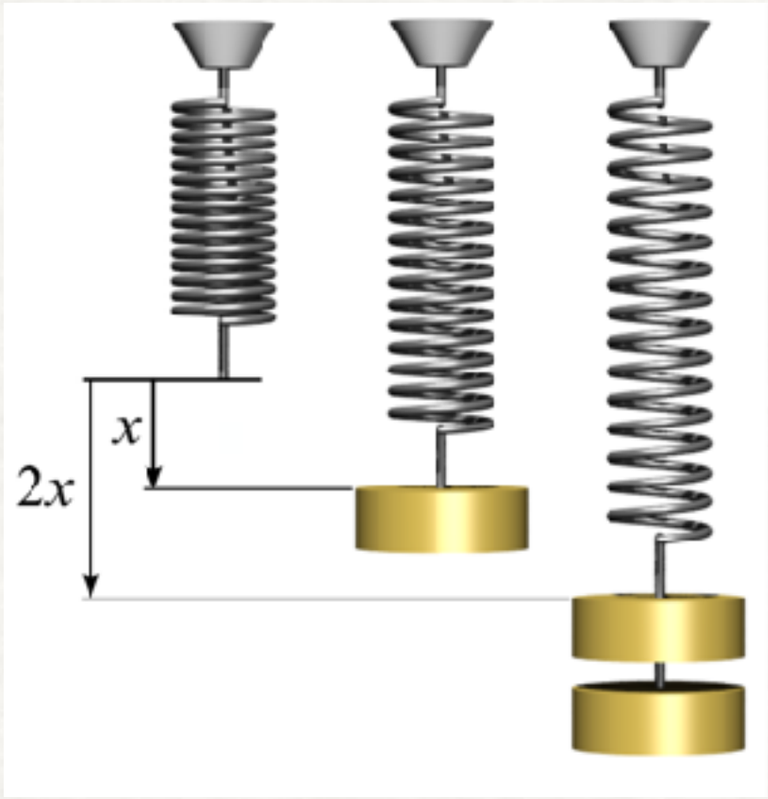
$$F = -kx$$

- 万有引力の法則: 重力Fは質量mかける重力加速度g

$$F = mg$$

- つまり、重りの分だけバネがのびる

変位（m）	質量（kg）
0.0865	0.1
0.1015	0.15
0.1106	0.2
0.1279	0.25
0.1892	0.3
0.2695	0.35
0.2888	0.4
0.2425	0.45
0.3465	0.5
0.3225	0.55
0.3764	0.6
0.4263	0.65
0.4562	0.7
0.4502	0.75
0.4499	0.8
0.4534	0.85
0.4416	0.9
0.4304	0.95
0.437	1.0



By Svjo - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=25398333

回帰分析

Python勉強会@HACHINOHE

- 回帰分析
  - データにあるモデル(関数)をあてはめる
  - 関数は、1次関数、2次関数など多項式関数、他
  - あてはめるには最小二乗法などを使う
- 最小二乗法
  - 観測値と予測値の差(残差)の二乗の和を、最小にするような予測値のパラメータを求める

$$\sum_i (\text{観測値}[i] - \text{予測値}[i])^2$$

- Pylabではpolyfitで線形回帰の計算ができる

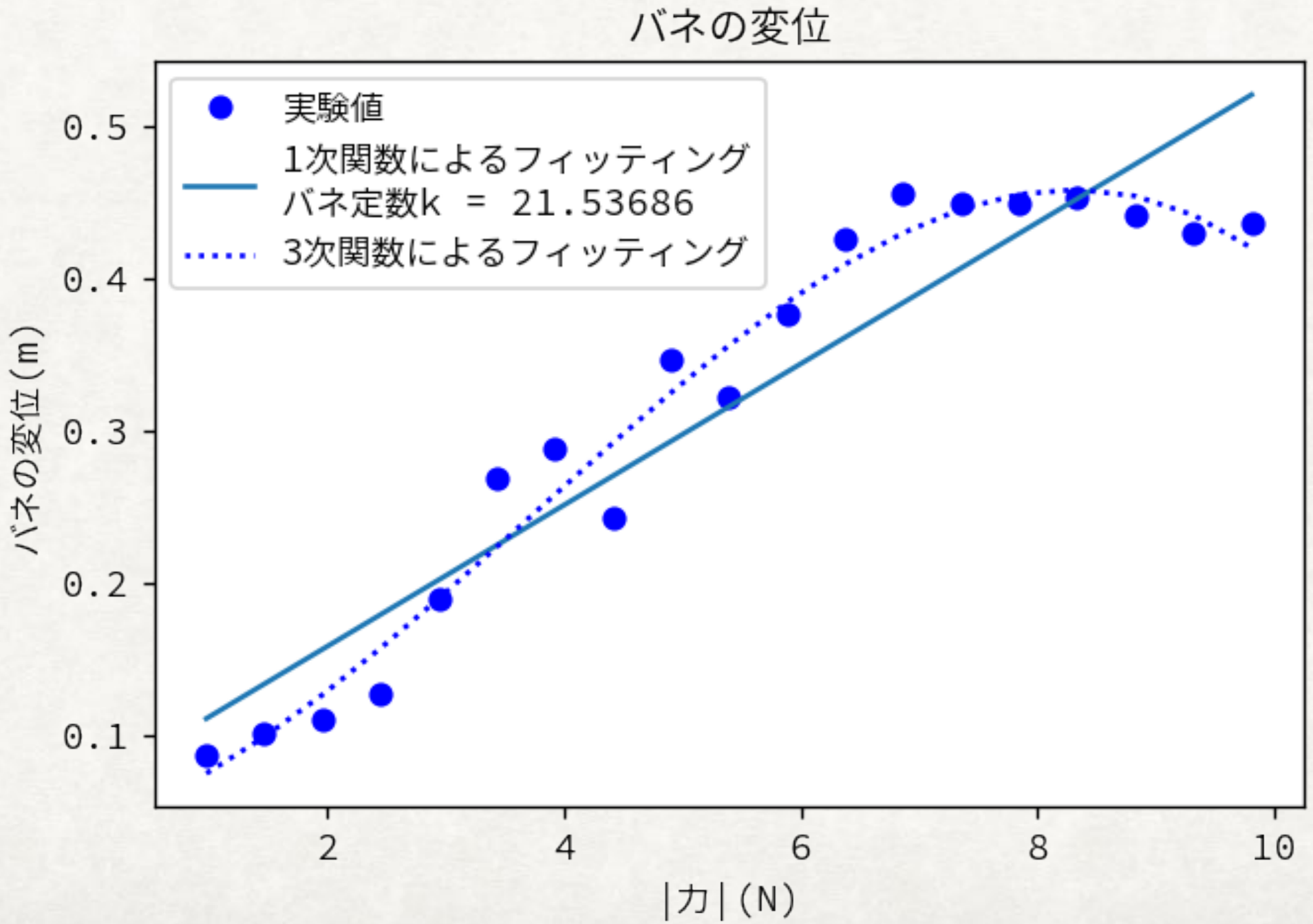
バネののびのプロット

Python勉強会@HACHINOHE

```
Import pylab

def getData(fileName):
    dataFile = open(fileName, 'r')
    distances = []
    masses = []
    discardHeader = dataFile.readline()
    for line in dataFile:
        d, m = line.split(' ')
        distances.append(float(d))
        masses.append(float(m))
    dataFile.close()
    return (masses, distances)

def plotData(inputFile):
    masses, distances = getData(inputFile)
    masses = pylab.array(masses)
    distances = pylab.array(distances)
    forces = masses * 9.81
    pylab.plot(forces, distances, 'bo', label = u'変位')
    pylab.title(u'バネの変位')
    pylab.xlabel(u'|力|(N)')
    pylab.ylabel(u'変位(m)')
```

バネののびのプロット

Python勉強会@HACHINOHE



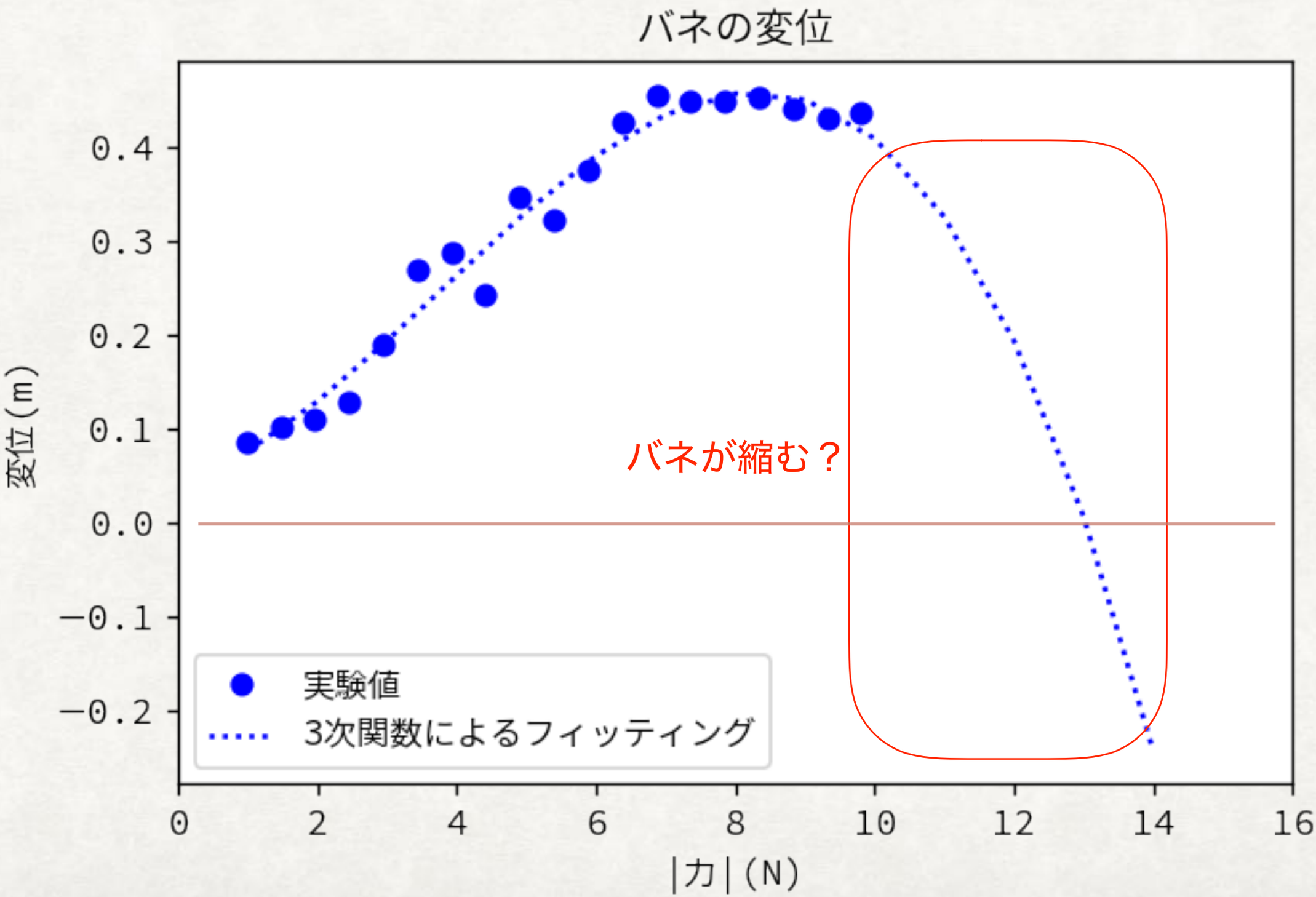
指練習

Python勉強会@HACHINOHE

```
def fitData(inputFile):
    pylab.figure(dpi=125)
    masses, distances = getData(inputFile)
    masses = pylab.array(masses)
    distances = pylab.array(distances)
    forces = masses * 9.81
    pylab.plot(forces, distances, 'bo', label = u'変位')
    pylab.title(u'ばねの変位')
    pylab.xlabel(u'|力|(N)')
    pylab.ylabel(u'距離(m)')

    # 3次の適合曲線を求める
    a, b, c, d = pylab.polyfit(forces, distances, 3)
    xs = pylab.array(range(1, 15)) # 1から15の範囲の整数
    # xs = pylab.arange(1, 15)
    predictedDistances = a * xs ** 3 + b * xs ** 2 + c * xs + d
    pylab.plot(xs, predictedDistances, 'b:', label = u'3次関数によるフィッティング')
    pylab.xlim(0, 16)
    pylab.legend(loc = 'best')
```

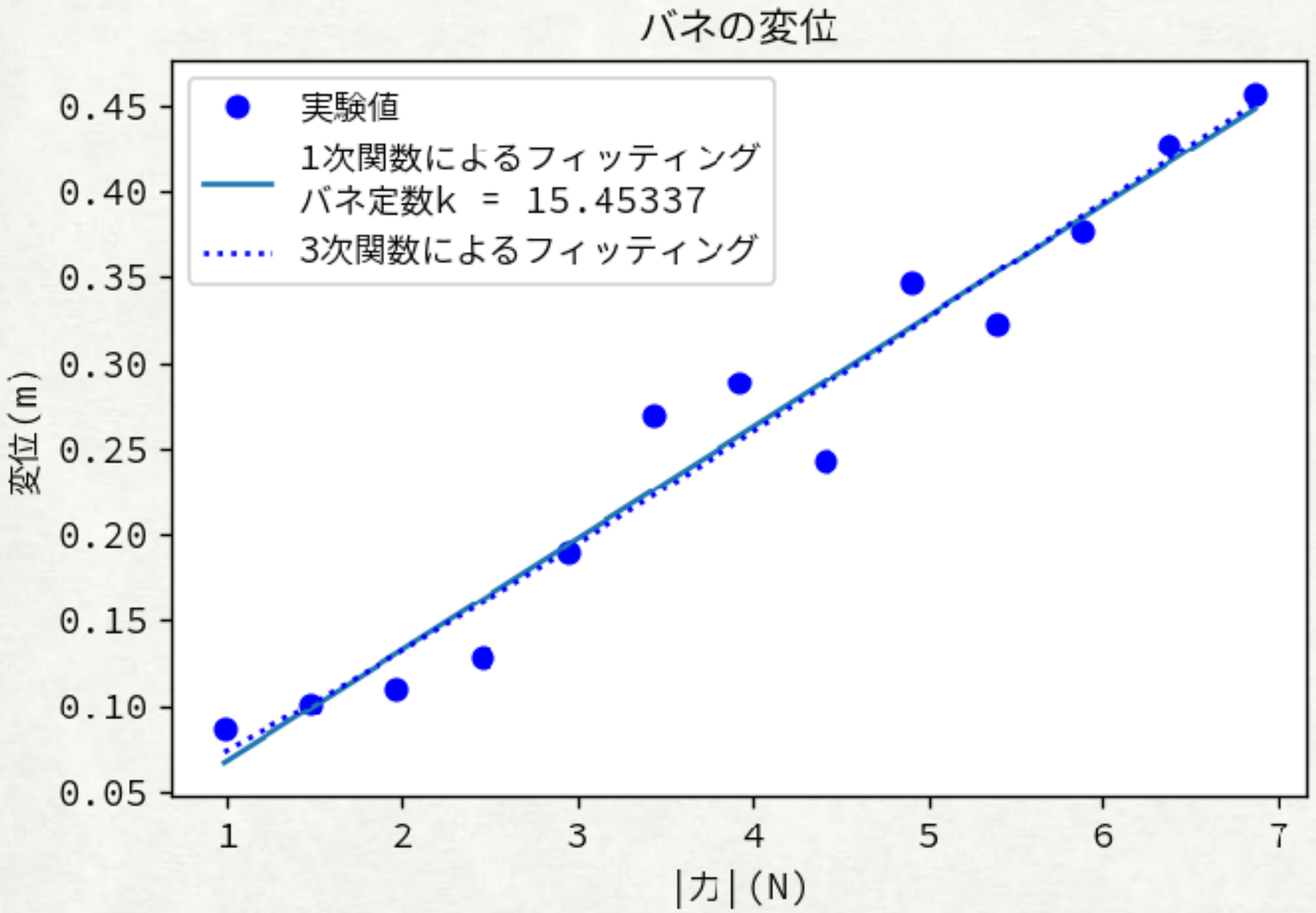
3次関数の方があっているというわけでもない

Python勉強会@HACHINOHE



弾性限界以上のデータを除去

Python勉強会@HACHINOHE



1次関数でも

3次関数でも

同じようにフィットする

つまり、1次関数で十分

バネの実験データとモデルの解釈

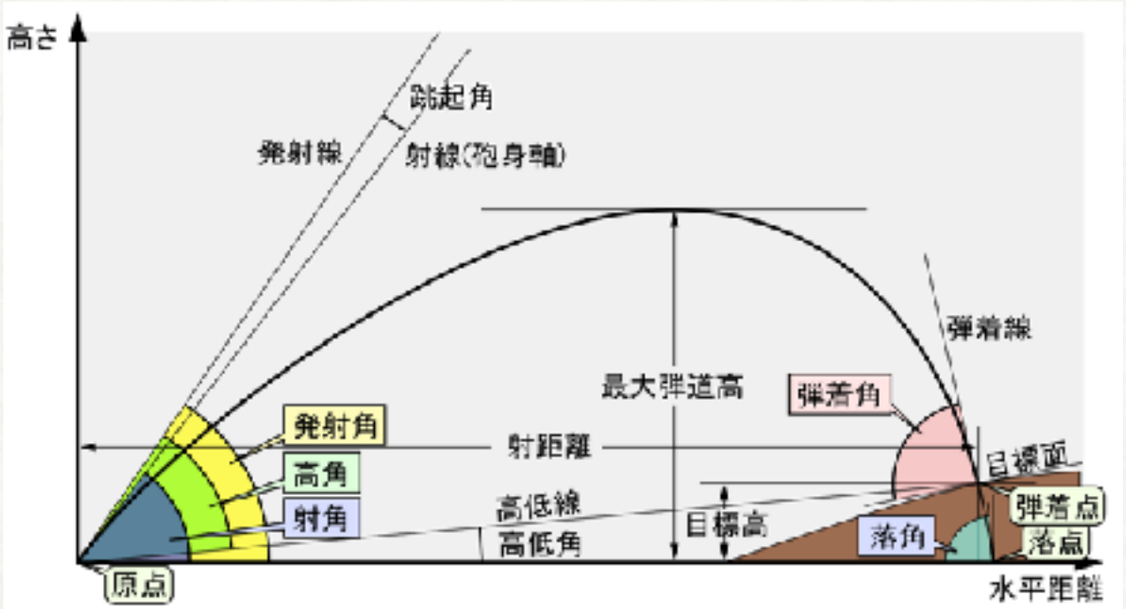
Python勉強会@HACHINOHE

- 3次関数を使うと、実験データにはとてもよく合う
- しかし、それはあくまで実験データの範囲内の話で、予測には役に立たない
  - 過剰適合で、むしろ誤差にフィットしているのかも
- 弾性限界があるという理論を元にすると、その限界内では1次関数で十分説明できる

発射体の振る舞い

Python勉強会@HACHINOHE

- 弾道: 空気抵抗がなければ放物線



By Tosaka - own work by uploader (ref:小林源文著　『武器と爆薬』、大日本絵画、2007年5月20日初版第1刷発行、ISBN 9784499229340),
CC 表示-継承 3.0, https://commons.wikimedia.org/w/index.php?curid=8036150

距離	実験1	実験2	実験3	実験4
1080	0.0	0.0	0.0	0.0
1044	2.25	3.25	4.5	6.5
1008	5.25	6.5	6.5	8.75
972	7.5	7.75	8.25	9.25
936	8.75	9.25	9.5	10.5
900	12.0	12.25	12.5	14.75
864	13.75	16.0	16.0	16.5
828	14.75	15.25	15.5	17.5
792	15.5	16.0	16.6	16.75
756	17.0	17.0	17.5	19.25
720	17.5	18.5	18.5	19.0
540	19.5	20.0	20.25	20.5
360	18.5	18.5	19.0	19.0
180	13.0	13.0	13.0	13.0
0	0.0	0.0	0.0	0.0

発射体の軌跡

Python勉強会@HACHINOHE

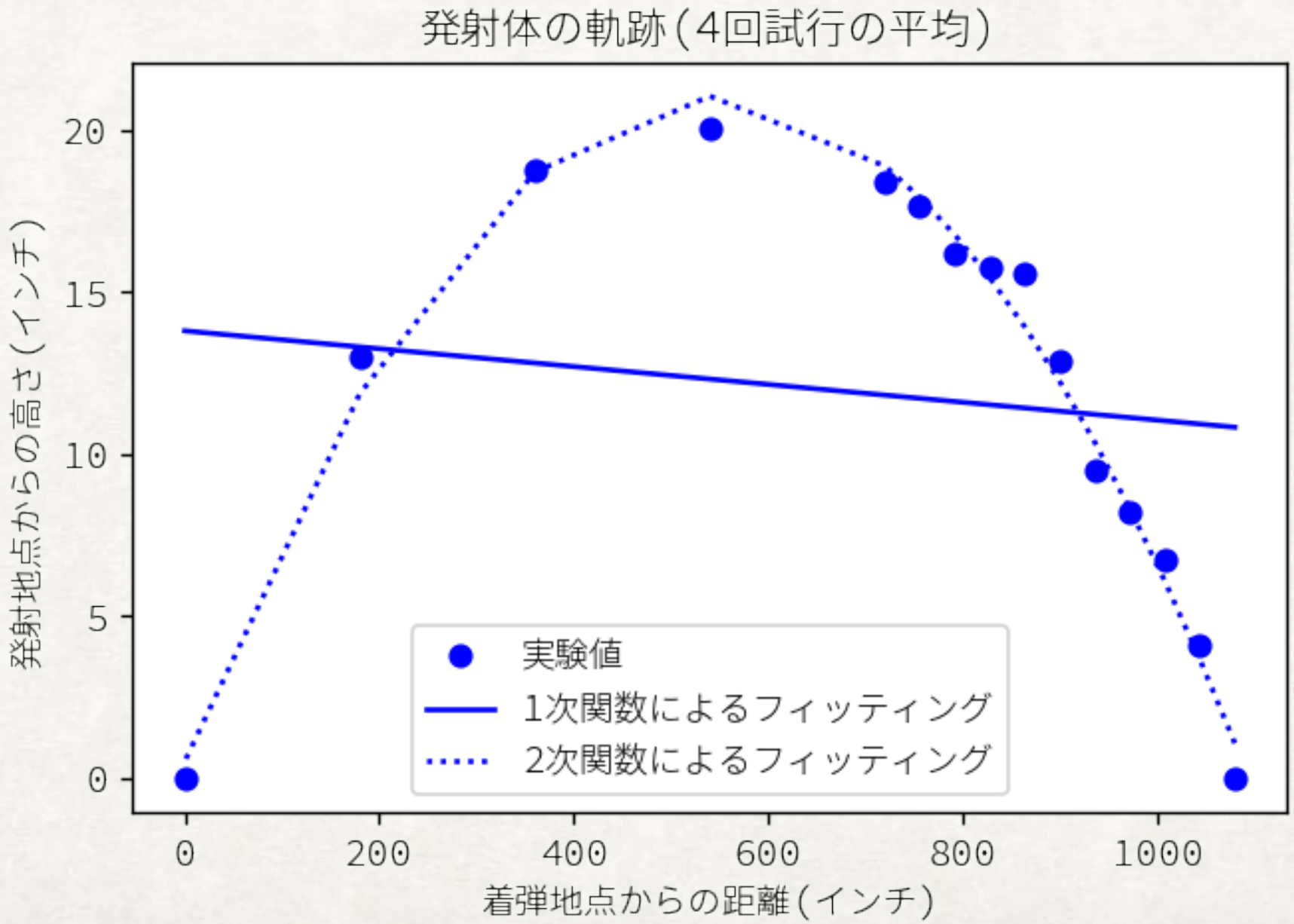
```
def getTrajectoryData(fileName):
    dataFile = open(fileName, 'r')
    distances = []
    heights1, heights2, heights3, heights4 = [], [], [], []
    discardHeader = dataFile.readline()
    for line in dataFile:
        d, h1, h2, h3, h4 = line.split()
        distances.append(float(d))
        heights1.append(float(h1))
        heights2.append(float(h2))
        heights3.append(float(h3))
        heights4.append(float(h4))
    dataFile.close()
    return (distances, [heights1, heights2, heights3, heights4])
```

```
def processTrajectories(fileName):
    distances, heights = getTrajectoryData(fileName)
    numTrials = len(heights)
    distances = pylab.array(distances)
    # 各距離における平均高さをもつ配列を得る
    totHeights = pylab.array([0] * len(distances))
    for h in heights:
        totHeights = totHeights + pylab.array(h)
    meanHeights = totHeights / len(heights)

    pylab.title(u'発射体の軌跡(' + str(numTrials) + u'回試行の平均)')
    pylab.xlabel(u'発射地点からの距離(インチ)')
    pylab.ylabel(u'発射地点からの高さ(インチ)')
    pylab.plot(distances, meanHeights, 'bo')

    a, b = pylab.polyfit(distances, meanHeights, 1)
    altitudes = a * distances + b
    pylab.plot(distances, altitudes, 'b', label = u'1次適合')

    a, b, c = pylab.polyfit(distances, meanHeights, 2)
    altitudes = a * distances * distances + b * distances + c
    pylab.plot(distances, altitudes, 'b:', label = u'2次適合')
    pylab.legend()
```

発射体の軌跡

Python勉強会@HACHINOHE

- 2次関数がよくフィットする



決定係数

Python勉強会@HACHINOHE

- 回帰曲線がどのくらいうまく当てはまるかを表す
- 観測値と予測値が適合しているほど、分子が小さくなる、つまり決定係数は1に近づく

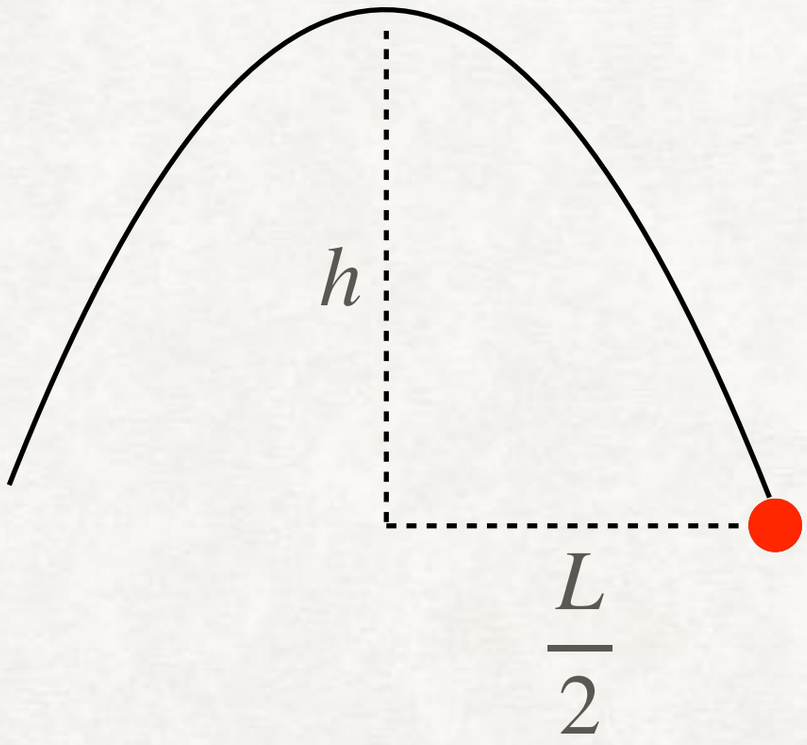
$$\text{決定係数} R^2 = 1 - \frac{\sum_i (\text{観測値}[i] - \text{予測値}[i])^2}{\sum_i (\text{観測値}[i] - \text{観測値の平均})^2}$$

- 線形回帰の場合、決定係数は観測値と予測値の相関係数の二乗
  - 1次適合のR^2 = 0.017743320544076 9
  - 2次適合のR^2 = 0.985765369286969 3

モデルによる予測

Python勉強会@HACHINOHE

- モデルにうまく当てはまれば予測にも使える
- 例えば、着弾速度
- 放物線の形状を考える

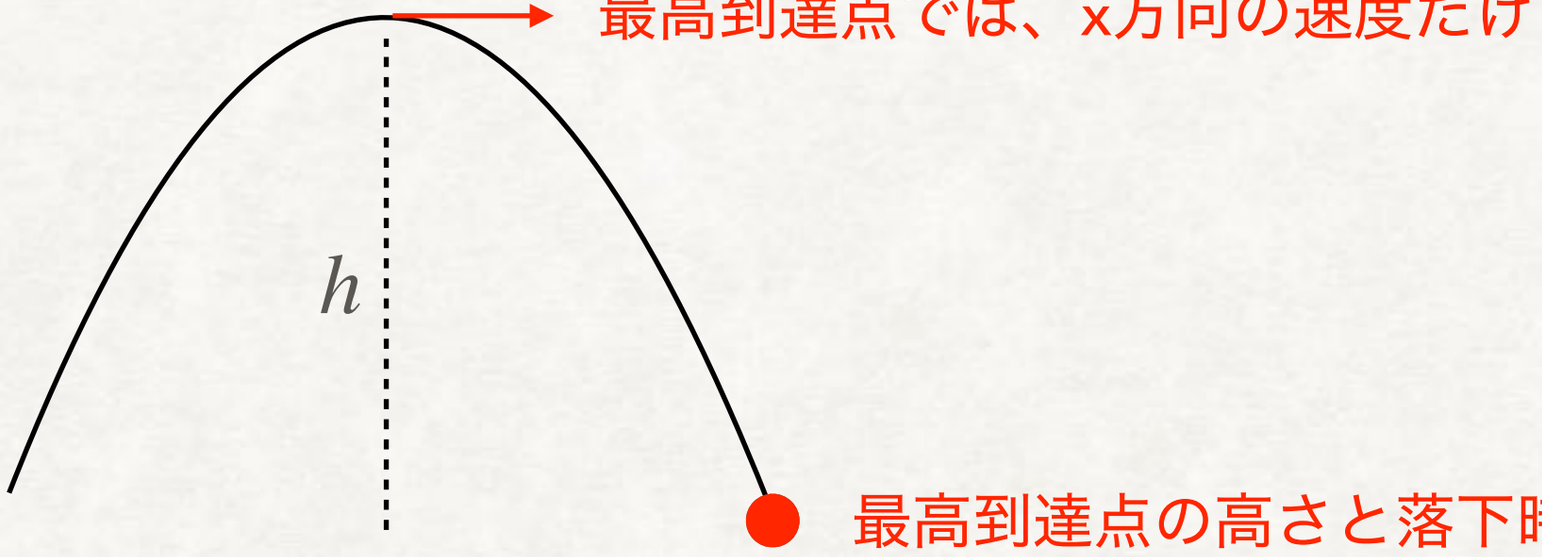


$$h = a(\frac{L}{2})^2 + b\frac{L}{2} + c$$

モデルによる予測

Python勉強会@HACHINOHE

- 上下方向を考える



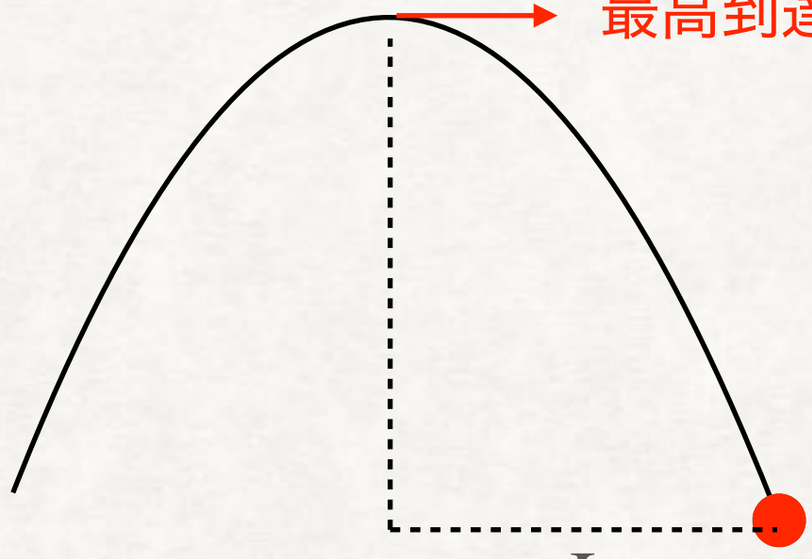
$$h = \frac{1}{2}gt^2$$

$$t = \sqrt{\frac{2h}{g}}$$

モデルによる予測

Python勉強会@HACHINOHE

- 左右方向を考える



$$\frac{L}{2} = v_x t$$

$$v_x = \frac{L}{2t}$$

モデルによる予測

Python勉強会@HACHINOHE

- Lからvxが計算できる

$$v_x = \frac{L}{2t}$$

$$t = \sqrt{\frac{2h}{g}}$$

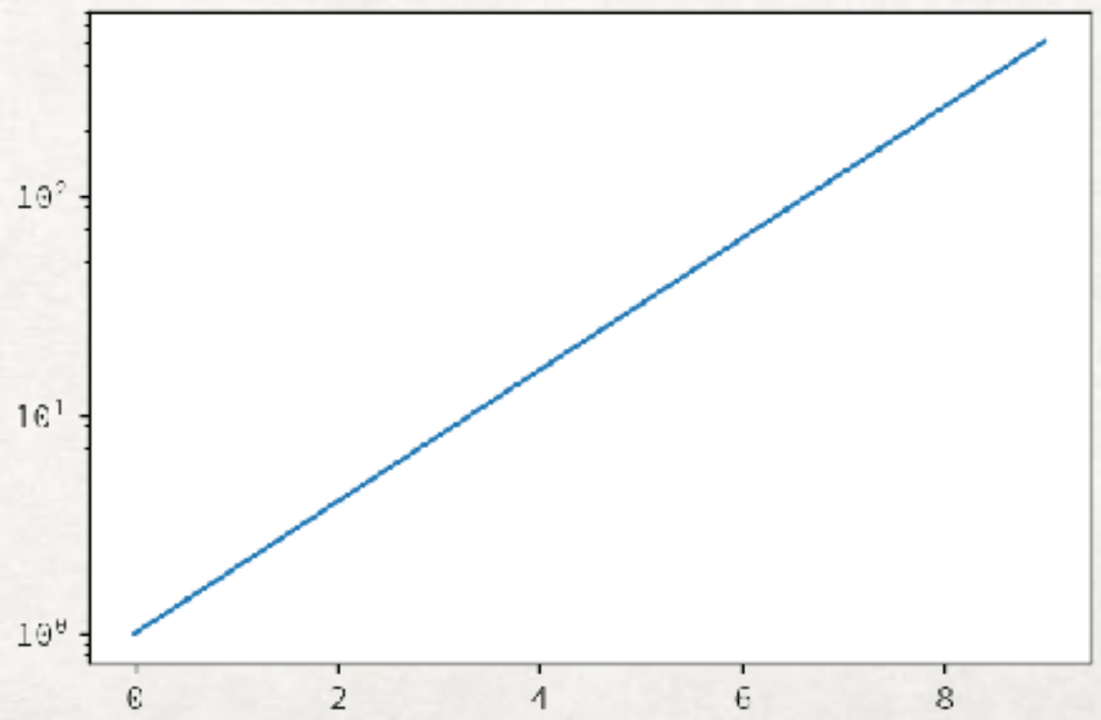
$$h = a(\frac{L}{2})^2 + b\frac{L}{2} + c$$

指数関数への線形回帰

Python勉強会@HACHINOHE

- PyLabのpolyfitは多項式の回帰曲線を求める
- $y = 2^x$ のような指数関数を、多項式で近似することはできないので、polyfitをそのまま使って回帰曲線を求めることはできない
- しかし、対数を取るなど、多項式の形に書ければpolyfitで回帰曲線を求めることができる

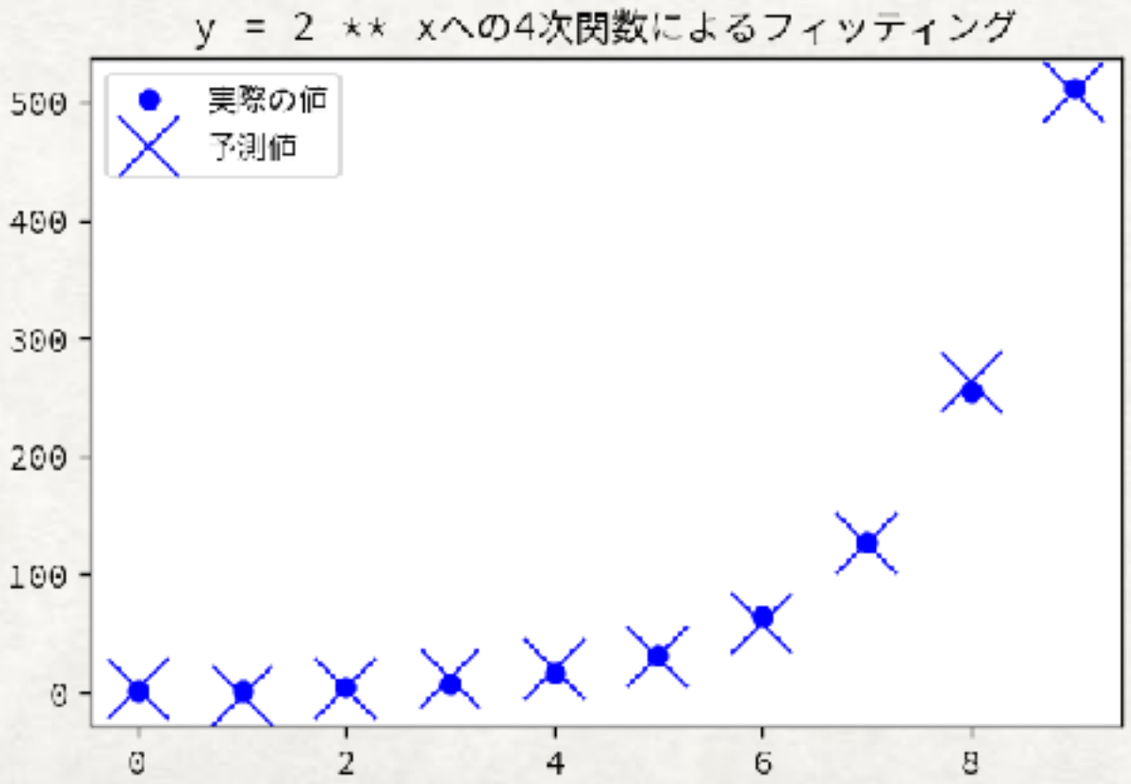
$$log(y) = log(2^x) = x \cdot log2$$



0-9の範囲で指数関数に4次関数でフィット

Python勉強会@HACHINOHE

```
vals = []
for i in range(10):
    vals.append(2 ** i)
pylab.plot(vals, 'bo', label = u'実際の値')
xVals = pylab.arange(10)
a, b, c, d, e = pylab.polyfit(xVals, vals, 4)
yVals = a * (xVals ** 4) + b * (xVals ** 3) + c * (xVals ** 2) + d * xVals + e
pylab.plot(yVals, 'bx', label = u'予測値', markersize = 20)
pylab.title(u'y = 2 ** xの適合')
pylab.legend()
```



範囲外では全く合わない

2 ** 20のモデルの大雑把な予測29796.0
2 ** 20の実際の値1048576

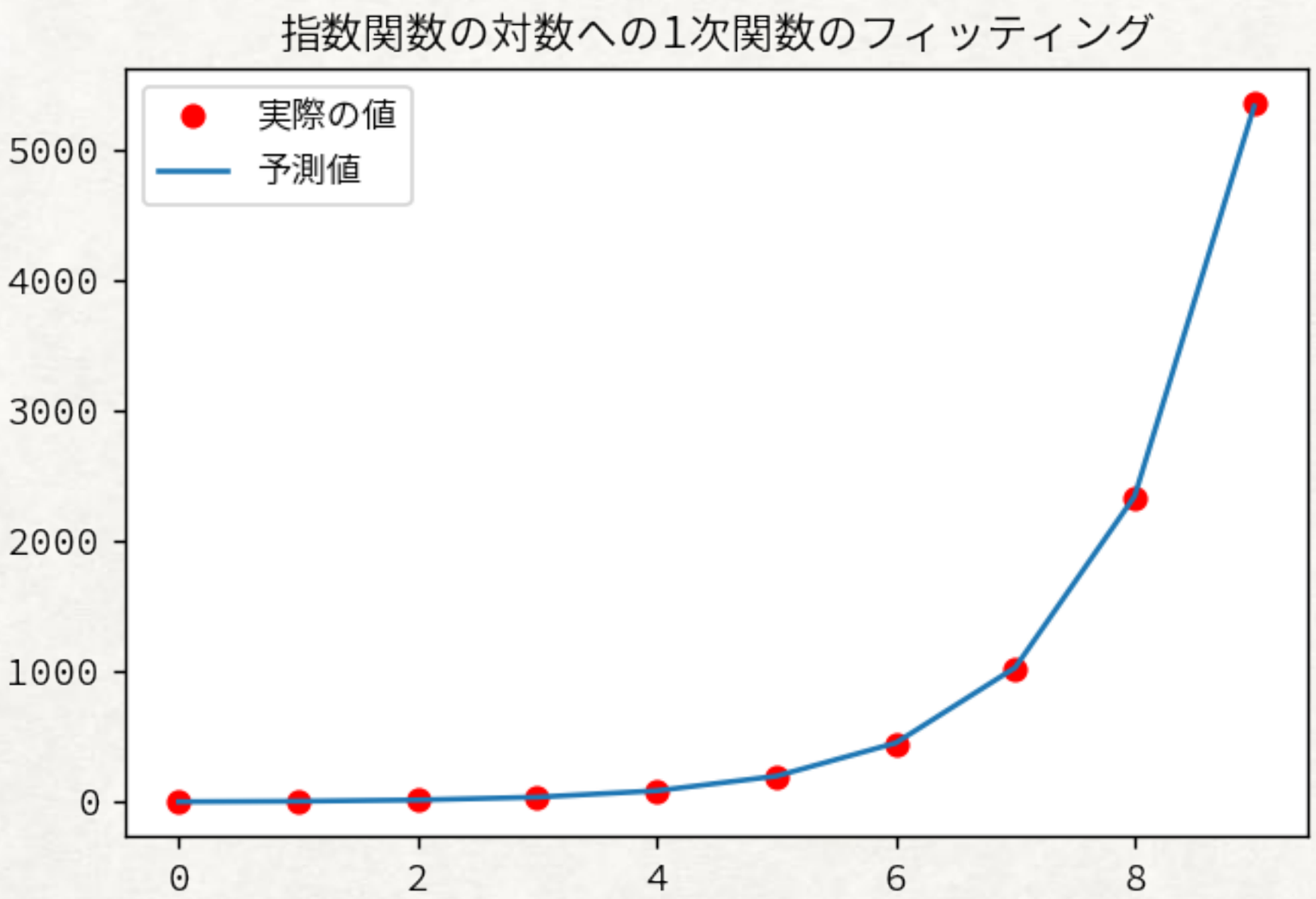
0-9の範囲で指数関数の対数に1次関数でフィット

Python勉強会@HACHINOHE

```
import math

# 任意の指数関数を定義する
def f(x):
    return 3 * (2 ** (1.2 * x))

def createExpData(f, xVals):
    """fを引数を1つ持つ指数関数と仮定する
       xValsを、fの適当な引数を持つ配列とする
       xValsの要素に関数fを適用した結果を保持する配列を返す"""
    yVals = []
    for i in range(len(xVals)):
        yVals.append(f(xVals[i]))
    return pylab.array(xVals), pylab.array(yVals)

def fitExpData(xVals, yVals):
    """xValsとyValsを
       yVals[i] == f(xVals[i])となる数を保持する配列と仮定する
       log(f(x), base) == ax + bを満たすa, bを返す"""
    logVals = []
    for y in yVals:
        logVals.append(math.log(y, 2.0)) # 底が2の対数
    a, b = pylab.polyfit(xVals, logVals, 1)
    return a, b, 2.0

xVals, yVals = createExpData(f, range(10))
pylab.plot(xVals, yVals, 'ro', label = u'実際の値')
a, b, base = fitExpData(xVals, yVals)
preditedYVals = []
for x in xVals:
    preditedYVals.append(base ** (a * x + b))
pylab.plot(xVals, preditedYVals, label = u'予測値')
pylab.title(u'指数関数の適合')
pylab.legend()
# オリジナルのデータにはないxの値を調べる
print 'f(20) =', f(20)
print u'予測 f(20) =', base ** (a * 20 + b)
```

0-9の範囲で指数関数の対数に1次関数でフィット

Python勉強会@HACHINOHE



範囲外でもよく合う

```
f(20) = 50331648.0
予測 f(20) = 50331647.999999814
```

理論が得られないとき

Python勉強会@HACHINOHE

- 実験→理論化→異なる独立変数で実験して確認(前向き実験

- 実験データ→学習データと検定データに分ける
- 学習データから理論を作る
- 検定データで理論を確かめる

- 実験データを複数に分ける
- それぞれのデータ群で理論を作る
- それぞれの理論がどのくらい近いかを確認(交差確認