計算物理学II

第10回
課題演習

秋山 進一郎

2025年12月19日

授業日の確認

・全10回

- ・第1回：10月3日（金）
- ・第2回：10月10日（金）
- ・第3回：10月17日（金）
- ・第4回：10月24日（金）★
- ・第5回：10月31日（金）
- ・第6回：11月14日（金）★
- ・第7回：11月21日（金）
- ・第8回：12月5日（金）★
- ・第9回：12月12日（金）
- **・第10回：12月19日（金）★**

・★の付いた授業にてレポート課題を配布予定

本日の演習内容

- 行列の特異値分解を使った情報圧縮
  - 特異値分解とは？
  - 特異値分解を使って画像データを圧縮する
- 統計誤差を解析する方法
  - ダーツを投げて円周率を求める
  - 不偏推定量，エラーバーとは？
  - ダーツ投げで求めた円周率にエラーバーをつける

特異値分解を使った画像圧縮

特異値分解（Singular Value Decomposition: SVD）

- m行n列の任意の行列Aは，$A = U\Sigma V^{\dagger}$ のように特異値分解することができる
  - Uはm行m列のユニタリ行列
  - Vはn行n列のユニタリ行列
  - Σは以下のようなm行n列の行列

(i) $m > n$ならば $\Sigma = \begin{bmatrix} \Delta \\ 0 \end{bmatrix}$　　(ii) $m = n$ならば $\Sigma = \Delta$　　(iii) $m < n$ならば $\Sigma = [\Delta \quad 0]$

  - $\Delta = \mathrm{diag}(\sigma_1, \sigma_2, \cdots \sigma_{\min(m,n)})$であり，必ず$\sigma_i \geq 0$（$\forall i$）
  - σ_iのことを行列Aの特異値と呼ぶ

特異値分解できる理由

- m行n列の任意の行列Aに対して，$AA^\dagger$，$A^\dagger A$はいずれも必ずHermite行列(半正定値)
  - Hermite行列なのでユニタリ行列で対角化できる
    - $AA^\dagger = U\lambda U^\dagger$
    - $A^\dagger A = V\lambda' V^\dagger$
  - 半正定値なので固有値は全てゼロ以上
  - 固有値の平方根が特異値

特異値分解の重要な性質

① 行列Aが正定値Hermite行列の場合，Aの特異値分解はAの固有値分解と等価になる

② 行列Aのランク = ゼロでないAの特異値の個数

③ 特異値分解はFrobenius（フロベニウス）ノルムの意味で最適な行列の低ランク近似を与える（この意味は後で説明）

④ 行列Aの特異値分解$A = U\Sigma V^{\dagger}$は，行列Aの擬似逆行列A^{+}を与える

- A^{+}が以下の4つを全て満たす時，A^{+}はAの擬似逆行列である

  (i) $AA^{+}A = A$　　(ii) $A^{+}AA^{+} = A^{+}$　　(iii) $(AA^{+})^{\dagger} = AA^{+}$　　(iv) $(A^{+}A)^{\dagger} = A^{+}A$

- 特異値分解$A = U\Sigma V^{\dagger}$の結果を使うと，$A^{+} := V\Sigma^{+}U^{\dagger}$で擬似逆行列が得られる
  - Σ^{+}はΣを転置し非ゼロ要素（特異値）だけ逆数に変えたもの

応用上の注

- 行列Aを$A = U\Sigma V^{\dagger}$のように特異値分解した場合，U, Vのいずれかを矩形行列とし，Σを正方行列として定式化する場合がある
  - U：m行$\min(m,n)$列，$V^{\dagger}$：$\min(m,n)$行n列，$\Sigma = \Delta = \mathrm{diag}(\sigma_1, \sigma_2, \cdots \sigma_{\min(m,n)})$
- この流儀に従うと，特異値分解$A = U\Sigma V^{\dagger}$の成分表示がやりやすい
  - $A_{ij} = \sum_{k=1}^{\min(m,n)} U_{ik}\sigma_k V^{\dagger}_{kj}$
- 特異値の順序は任意だが，必ず非負の実数なので降順に並べておくことが多い
  - $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min(m,n)} \geq 0$
- 次頁以降では，これらの流儀を常に仮定しています

特異値分解による低ランク近似

- 以下のような問題を考える
  - 行列AをAよりもランクが低い行列$\widetilde{A}$で「上手く」近似できないだろうか？
- 特異値分解はFrobenius（フロベニウス）ノルムの意味で最適な行列の低ランク近似を与える
  - m行n列の行列AのFrobeniusノルム$|A|_F$の定義は$|A|_F \coloneqq \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}\left|A_{ij}\right|^2}$
  - 行列Aの特異値分解の表式から，$|A|_F = \sqrt{\mathrm{Tr}(A^\dagger A)} = \sqrt{\sum_{i=1}^{\min(m,n)}\sigma_i^2}$
- ランクχの行列の中で，Frobeniusノルムの意味での行列Aの最も良く近似する行列$\widetilde{A}$は，特異値分解$A = U\Sigma V^\dagger$の結果から以下のように直ちに得られる

$$\widetilde{A}_{ij} = \sum_{i=1}^{\chi} U_{ik}\sigma_k V_{kj}^\dagger$$

計算物理学における特異値分解の応用例

- 主成分解析（Principal Component Analysis: PCA）
  - 多次元データの持つ情報をできるだけ保って低次元データへと落とし込む
- テンソルネットワーク（Tensor Network）
  - エンタングルメント・エントロピー(量子的な相関をはかる量)を指標に使った情報圧縮に基づく量子多体系の数値計算手法
  - 量子コンピュータ(量子回路)の古典コンピュータによるシミュレーションにも広く応用されている
  - 機械学習への応用も盛ん

SciPyを使って特異値分解（SVD）を行ってみよう

- linalg.svdを使えばよい
  - リンクから公式マニュアルを見て使い方やオプションを見ておこう
- 右のようなコードを書いて実行してみよう
  - 6行目でSVDを実行
    - full_matrices=Falseで$A = U\Sigma V^{\dagger}$のΣを正方行列$\Sigma = \Delta$とする流儀になる
    - U, S, VtにU, Δ, $V^{\dagger}$が格納される．Sは一次元のNumPy配列で返ってくることに注意
  - min(m,n)=4だが，ゼロでない特異値は3個だけ（行列Aのランクは3）

```
1  import numpy as np
2  from scipy import linalg
3
4  A = np.array([[1,1,1,1],[2,2,1,3],[1,1,1,-1],[2,2,1,-1]])
5
6  U, S, Vt = linalg.svd(A, full_matrices=False)
7
8  print(S)
```

低ランク近似をしてみよう

```
1   import numpy as np
2   from scipy import linalg
3
4   A = np.array([[1,1,1,1],[2,2,1,3],[1,1,1,-1],[2,2,1,-1]])
5
6   U, S, Vt = linalg.svd(A, full_matrices=False)
7
8   print(S)
9
10  norm_full = np.sum(S*S)
11  print(norm_full)
12
13  S_compressed = np.zeros_like(S)
14
15  chi = 3
16  S_compressed[:chi] = S[:chi]
17
18  norm_compressed = np.sum(S_compressed*S_compressed)
19  print(norm_compressed)
```

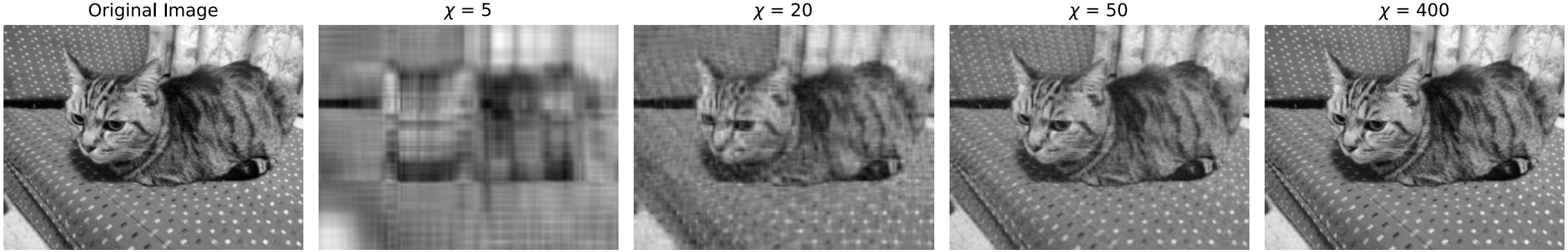
- SVDを行い，Frobeniusノルムを計算しよう
- 右のようなコードを書いて実行してみよう
  - 10行目でFrobeniusノルムを計算
  - 13行目以下で低ランク近似を実行
  - np.zeros_like(S)で，Sと同じshapeで全成分が0であるNumPy配列を用意できる
  - 16行目でchi+1個の特異値$\sigma_0, \sigma_1, \cdots, \sigma_{\mathrm{chi}}$をS_compressedへ代入
  - chi=3では特異値の打ち切りが存在しないので，19行目の出力結果は元のFrobeniusノルム(11行目の出力結果)と一致する
  - chiを2や1に変え，低ランク近似後のFrobeniusノルムを見てみよう

低ランク近似を応用した画像圧縮（レポート課題）

- 高さがmピクセル，幅がnピクセルの画像はm行n列の行列とみなすことができる
  - 各色彩に異なる数値を対応させて考えれば，行列の各要素における数値が元の画像における各ピクセルの色彩を表す
- 元の画像を表す行列AをSVDし，低ランク近似を実行後，ランクχの行列$\widetilde{A}$を構成することで画像圧縮ができる
  - やってみましょう（→続きはレポート課題で）
  - 画像を行列Aに変換する関数，ランクχの行列$\widetilde{A}$を画像に変換する関数はレポート課題に記載してあります
- 次頁で実際の画像圧縮の精度を見てみよう

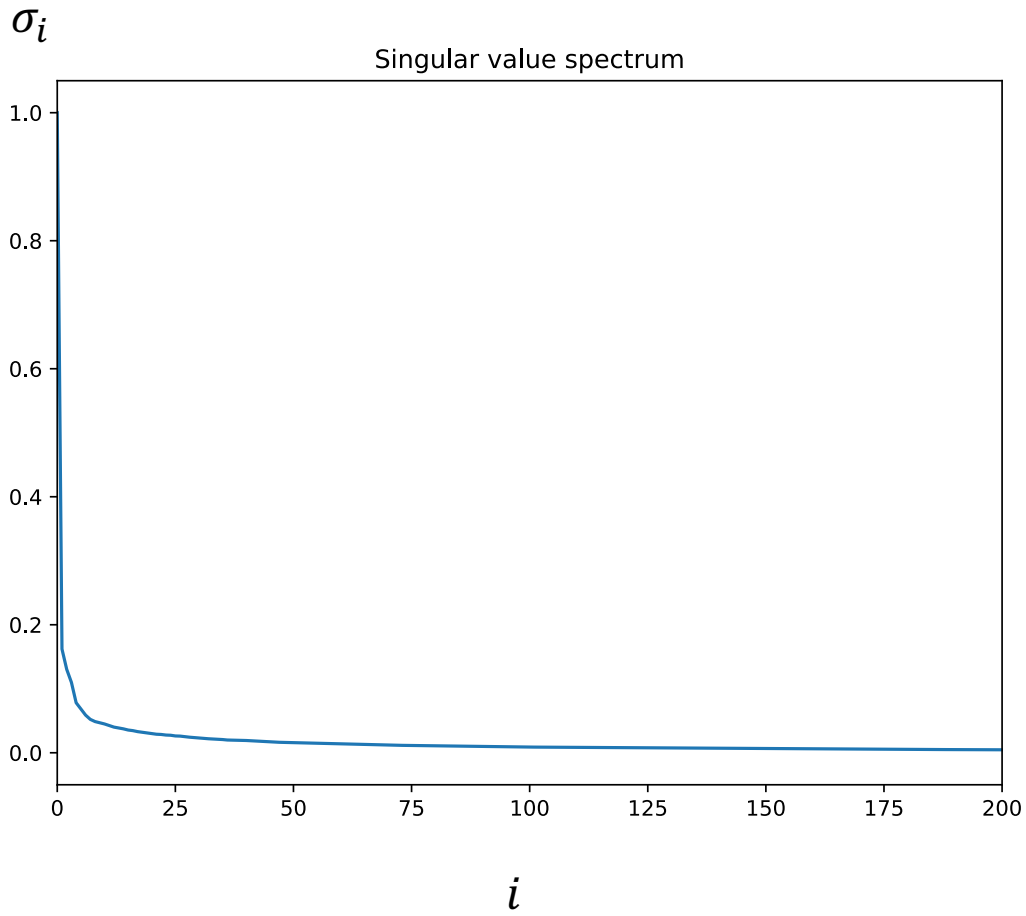
SVDによる低ランク近似を応用した画像圧縮

- 実際に画像を圧縮してみる
  - 以下の例では，元の画像（左）は1536行2048列の行列
  - χを大きくしていくと画像の細部がより精密に再現されるようになる
  - 素朴には，特異値は全部で1536個あるので，χ=400でもかなり小さいと言える



特異値の分布を見てみると…

- 先ほどの画像の特異値分布は右の通り
- 特異値に急激な減衰が見られている（図には201番目の特異値までしかplotされていないことに注意）
- このような場合，十分小さいχの値で元画像の再現が可能

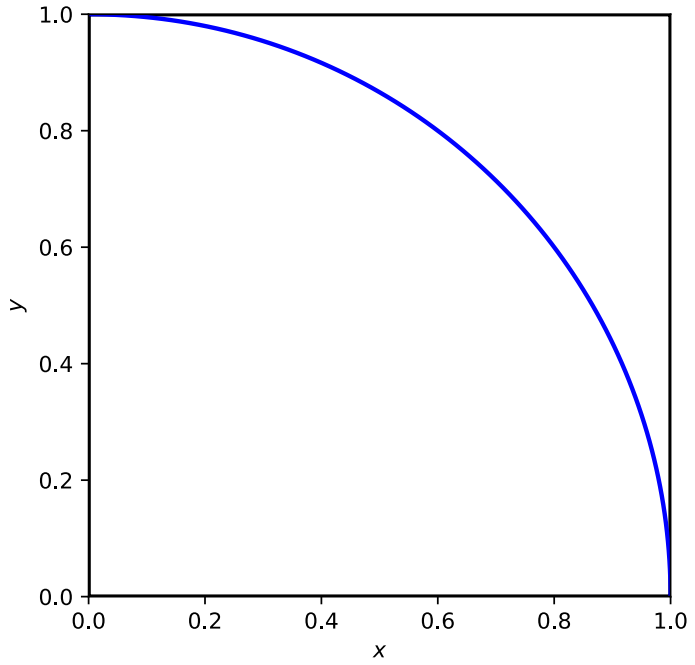


統計誤差を解析する方法

ダーツ投げで円周率を推定しよう 1/3

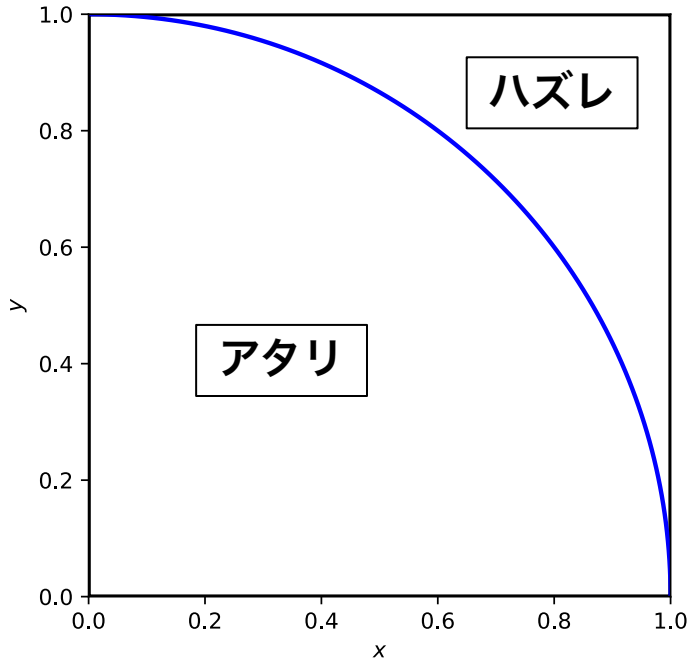
- 以下のような一辺の長さが1の正方形の中に存在する四分円を考え, これに向かってランダムにダーツを投げることを考える

  - ただし, 投げたダーツは必ず以下の図のどこかに刺さるものとする



ダーツ投げで円周率を推定しよう 2/3

- ダーツが四分円の中に刺さったら「アタリ」, 外に刺さったら「ハズレ」とする
- ランダムなダーツ投げの場合, アタリの確率は(四分円の面積)/(正方形の面積)=$\pi/4$ になると期待される

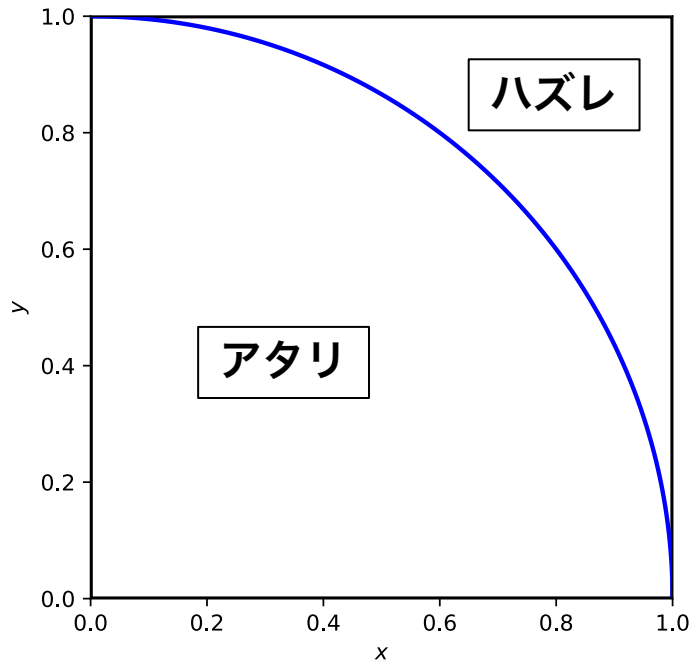


ダーツ投げで円周率を推定しよう 3/3

・N回ダーツを投げ, アタリの回数をMとする

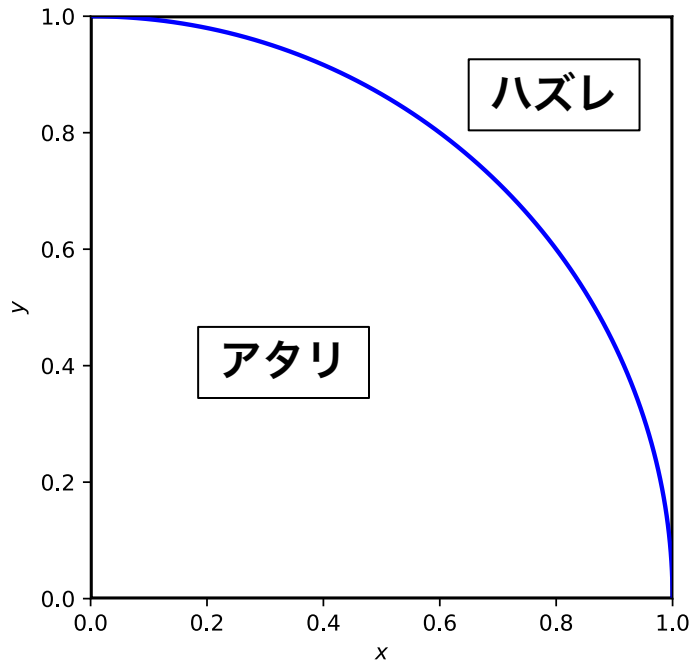
・十分大きなNに対して, M/N~$\pi/4$になるだろう

・アタリの回数を数えれば, 4M/Nが円周率をよく近似するはず



プログラムを書いて確かめよう 1/2

- 以下の的にランダムにダーツを投げることは, 0以上1以下の実数値x, yをランダムに生成することと等価である
  - $x^2+y^2<1$ならアタリ, そうでなければハズレである



プログラムを書いて確かめよう 2/2

- 右のようなコードを書けば, N回のダーツ投げから円周率を推定できる
  - np.random.uniform(0,1)は0以上1未満の実数をランダムに生成してくれる
- throw_darts.pyの名前で右のコードを書き, 円周率の推定値を見てみよう
  - 15行目のnum_dartsを変えて結果を見てみよう
    - num_darts=10^6程度だとどうか？

```python
import numpy as np
import matplotlib.pyplot as plt

def throw_darts(N):
    num_hit = 0
    for _ in range(N):
        x = np.random.uniform(0, 1)
        y = np.random.uniform(0, 1)
        if x*x + y*y < 1.0:
            num_hit = num_hit + 1
    return 4*num_hit/N

if __name__ == "__main__":

    num_darts = 100
    print(throw_darts(num_darts))
```

ヒストグラムを作ってみよう

- 「N回のダーツ投げから円周率を推定する」という実験をK回行い, 得られた円周率に関数するヒストグラムを作ってみよう

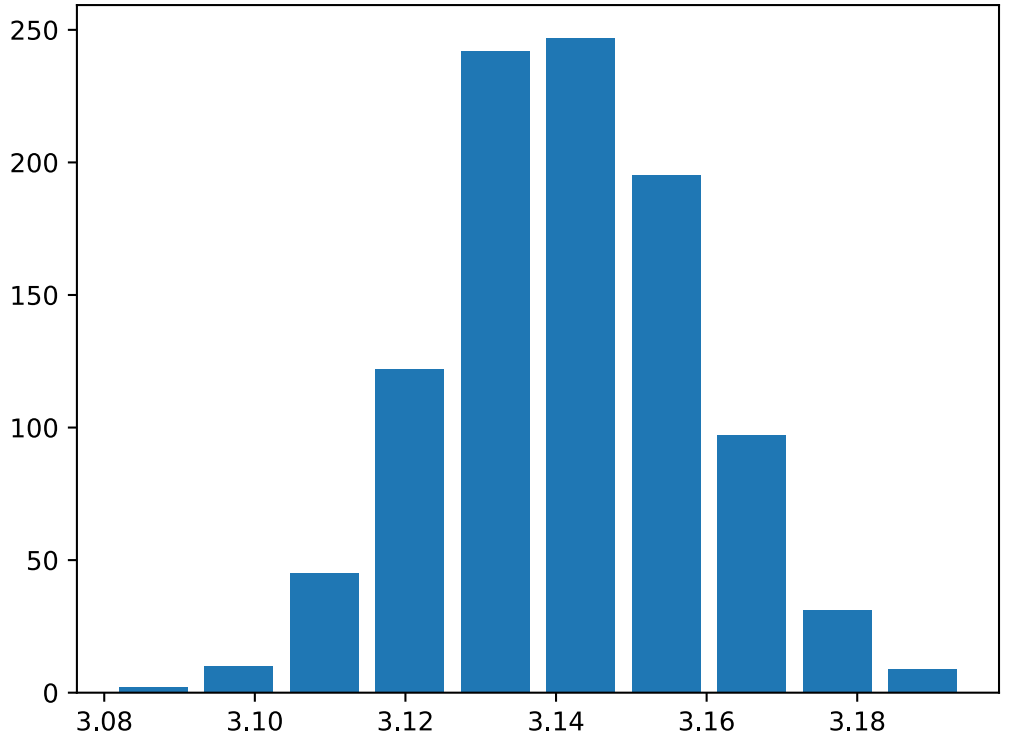
- throw_darts.pyを右のように修正しよう

- plt.histにリストを渡すとヒストグラムを描いてくれる
  rwidth=0.8は棒グラフの幅を調節している(単に見栄えのため)

```
13  if __name__ == "__main__":
14
15      #num_darts = 100
16      #print(throw_darts(num_darts))
17
18      num_sample = 1000
19      num_darts = 10000
20
21      results = []
22      for _ in range (num_sample):
23          results.append(throw_darts(num_darts))
24
25      plt.figure()
26      plt.hist(results, rwidth=0.8)
27      plt.savefig(f"hist_pi_N{num_darts}_K{num_sample}.pdf")
28
```

ヒストグラムを見てみよう

- 「N=10^4回のダーツ投げから円周率を推定する」という実験をK=10^3回行った結果
  - 3.14辺りを中心とする分布になっている
- 有限の試行回数に起因した誤差の影響まで含め, 円周率の推定を行えないだろうか?



不偏推定量

- 実験や数値シミュレーションでは, 母集団から有限個の標本(データ)しか抽出(サンプル)できないため, 得られた有限個の標本から母集団の特徴を推定したい場合が多々ある

- 数学的には「母集団が従う確率分布を標本から推定する」という問題になる

- この問題を考える上で役に立つのが不偏推定量

標本平均，不偏標本分散

- ある母集団からn個の標本 $X_1, X_2, \cdots, X_n$ (以下では常にすべて独立とする)をサンプルしたとする

- ある統計量$\theta(X_1, X_2, \cdots, X_n)$の期待値が推定したい母集団の母パラメタ$\theta$に一致する場合，統計量$\theta(X_1, X_2, \cdots, X_n)$は母パラメタ$\theta$の不偏推定量であるという

- 重要な不偏推定量として次の二つがある

  - 標本平均 $\bar{X} = (X_1 + X_2 + \cdots + X_n)/n$ は母平均の不偏推定量

  - 不偏標本分散 $U^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2$ は母分散の不偏推定量

注)通常の標本分散 $S^2 = \frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^2$ は母分散の不偏推定量ではない

乱数を使った簡単な実験

- 例として正規分布に従う母集団（正規母集団）を考えよう
  - 母平均をμ, 母分散をσ^2とする
- np.rand.normalを使うと, 母平均μ, 母分散σ^2の正規母集団からn個のサンプルを得ることができる
  - 標本平均と不偏標本分散を計算し, 真の母平均μ, 母分散σ^2と比べてみよう（次頁）

プログラムの例

- 右のようなコードを書いて実行してみよう
- 母平均の値をmu, 母分散の値をsigma2, サンプル数をNで指定
- サンプルをNumPy配列dataに格納
- dataの標本平均はnp.average(data), 不偏標本分散はnp.std(data,ddof=1) で計算できる
  - オプションddof=1を指定しない場合，np.std(data)はdataに対する通常の標本分散を返すことに注意
- 通常の標本分散 np.std(data)も計算し, 不偏標本分散の結果と見比べてみよう （小さなNほど差が見やすいはず）

```python
import numpy as np

mu = 0
sigma2 = 0.01
N = 10000

data = np.random.normal(mu,sigma2,N)
estimated_mean = np.average(data)
estimated_variance = np.std(data,ddof=1)

print("True mean : ", mu)
print("Estimated mean : ", estimated_mean)

print("True variance : ", sigma2)
print("Estimated variance : ", estimated_variance)
```

中心極限定理と統計誤差 1/2

- **中心極限定理**：確率変数 X_i (すべて独立とする) の平均をμ，分散をσ^2とする．
  サンプル数nが無限大の極限で，確率変数 $X = (X_1 + \cdots + X_n)/n$ は平均μ，分散$\sigma_X^2 \coloneqq \sigma^2/n$ の正規分布に分布収束する

- 確率変数Xが平均μ，分散σ_X^2の正規分布に従う場合，$\mu - \sigma_X < X < \mu + \sigma_X$となる確率は

$$P(\mu - \sigma_X < X < \mu + \sigma_X) = \int_{\mu-\sigma_X}^{\mu+\sigma_X} \mathrm{d}x \frac{1}{\sqrt{2\pi\sigma_X^2}} \exp\left[\frac{(x-\mu)^2}{2\sigma_X^2}\right] \sim 0.6827$$

- $\mu - \sigma_X < X < \mu + \sigma_X$を「1シグマの範囲」などという

- 同様に $\mu - n\sigma_X < X < \mu + n\sigma_X$を「$n$シグマの範囲」などと呼ぶ

$$P(\mu - n\sigma_X < X < \mu + n\sigma_X) = \int_{\mu-n\sigma_X}^{\mu+n\sigma_X} \mathrm{d}x \frac{1}{\sqrt{2\pi\sigma_X^2}} \exp\left[\frac{(x-\mu)^2}{2\sigma_X^2}\right] \sim \begin{cases} 0.6827 & (n=1) \\ 0.9545 & (n=2) \\ 0.9973 & (n=3) \end{cases}$$

中心極限定理と統計誤差 2/2

- $X = (X_1 + \cdots + X_n)/n$ は母平均μの不偏推定量であり, サンプル数nを増やしていけばその期待値は母平均μに漸近する

- 実際の実験や数値シミュレーションでは, 真の母平均μは不明であり, 得られるのは標本平均$\bar{X}$である. 逆に考えると, 真の母平均μは得られた標本平均$\bar{X}$を中心とする分散σ_X^2の正規分布に従って確率的に分布しているはずである

- 実験や数値シミュレーションで得られた標本平均$\bar{X}$に対して「nシグマの範囲」をつけておけば, どの程度の確率で真の値μがその範囲に存在するかの良い指標になる
  $\rightarrow$ これがいわゆるエラーバーであり, 統計誤差の目安を与える

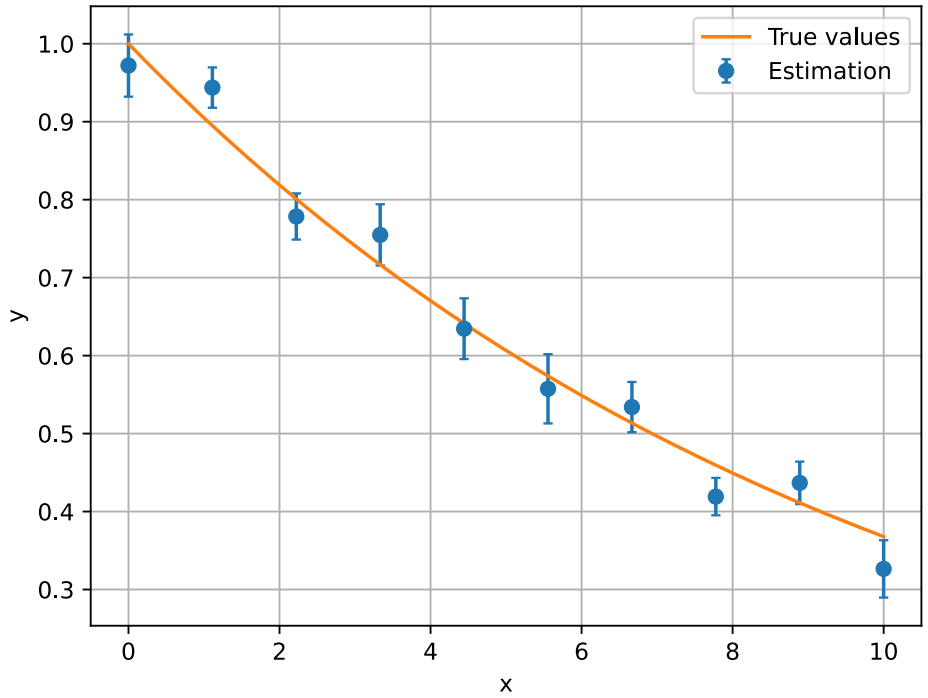
- $\sigma_X = \sigma/\sqrt{n}$なので, エラーバー(統計誤差)を$1/k$に低減するには$k^2$倍のサンプル数が必要

エラーバーを求める際の注意

- エラーバーを求めるには $\sigma_X = \sigma/\sqrt{n}$ の値を知る必要があるが, 標準偏差 σ も未知である
- よく使われる手の一つは, 不偏標本分散の平方根($U = \sqrt{U^2}$)を使って$U/\sqrt{n}$でエラーバーをつける方法. ただし, サンプル数nが十分大きい場合などには, 通常の標本分散の平方根($S = \sqrt{S^2}$)が使われていることもある
- 他にも, 「nシグマの範囲」ではなく信頼度zの信頼区間でエラーをつける場合もある. 興味のある人は確率・統計論の教科書などを見てほしい

エラーバーをつけてみる

- サンプルコードとして, /lecture10/src/error_analysis.py を使ってみよう
- このコードでは，$y = \mathrm{e}^{-x/10}$という関数に，np.rand.normalを使ってわざとノイズをのせ，ノイズのついた値を実験で得られた数値と見立ててN回求め，その平均と1シグマのエラーバーを計算しています
- 乱数のseedを固定していないので，実行する度に違う結果が得られます
- 1シグマのエラーバーをつけたので，約68%の確率でエラーバーの範囲内に真の値（オレンジ）が存在するはずです
- コードを実行してみましょう



サンプルコードの説明 1/2

- 最初に実験を行うx点を10箇所均等に選び, その点での厳密な値$y = \mathrm{e}^{-x/10}$を計算します

```
4    sampling_points = np.linspace(0,10,10)
5    original_values = np.exp(-sampling_points/10)
```

- 次に実験を行う回数Nを設定し, 後で実験で得られた数値を格納するためのNumPy配列dataを準備します

```
7    N = 10
8    data = np.zeros((len(sampling_points),N))
```

- N回実験を行い, 毎回異なる乱数を生成し, その乱数を0.1倍して厳密な値に足します.
  measured_valuesがi番目の実験で得られたデータです.
  13行目では, measured_valuesの全要素をdataのi列目に代入しています.

```
10   for i in range(N):
11       random_values = np.random.normal(0, 1, len(sampling_points)) * 0.1
12       measured_values = original_values + random_values
13       data[:,i] = measured_values[:]
```

サンプルコードの説明 2/2

- 各x点上で得られたN個の数値を使って, 平均とエラーバーを計算します.
  dataは10(x点の数)行N列の行列になっているので, dataの各行ごとにmeanとstdを計算すればOK. オプションでaxis=1とすると, 各行ごとにmeanとstdが計算されます.
  ちなみにaxis=0とすると列ごとに処理が行われます

```
15    estimated_means = np.mean(data, axis=1)
16    estimated_errors = np.std(data, axis=1, ddof=1) / np.sqrt(N)
```

- データ点にエラーバーをつける場合, plt.errorbar()が便利です. yerrorにエラーバーの値を渡すことで, y軸のデータにエラーバーがつきます.
  capsizeでエラーバーの両端の線の大きさを変更できます

```
21    plt.errorbar(sampling_points, estimated_means, yerr=estimated_errors, fmt='o', capsize=2, label='Estimation')
```

ダーツ投げで求めた円周率にエラーバーをつけてみよう 1/2

- 新しく, throw_darts_error.pyという名前のファイルを作って次の関数を書こう
- throw_darts(N)はN回ダーツを投擲し, 円周率の推定値と統計誤差を返す関数
- 先ほどのthrow_darts.pyで書いた関数を少し変えている
  - アタリの回数を単一の整数値(num_hit)で記録せず, dataというサイズのNのNumPy配列を用意し, i番目の試行の結果をアタリなら1, ハズレなら0としてi成分data[i]に保存
  - NumPy配列で結果を保持しているので, 前の例と同様に, np.averageやnp.stdにそのまま渡すことができる

```python
import numpy as np
import matplotlib.pyplot as plt

def throw_darts(N):

    data = np.zeros(N)
    for i in range(N):
        x = np.random.uniform(0, 1)
        y = np.random.uniform(0, 1)
        if x*x + y*y < 1.0:
            data[i] = 1

    inside_sphere = np.average(data)
    estimated_pi= 4 * inside_sphere
    error = 4 * np.std(data,ddof=1)/np.sqrt(N)

    return estimated_pi, error
```

ダーツ投げで求めた円周率にエラーバーをつけてみよう 2/2

・続けて, 以下のコードを書いてみよう. (これまでの演習により, コードが読めるはず)
ダーツの投擲回数を10の冪乗で変え, 円周率の推定値と統計誤差を求めている

```
19  if __name__ == "__main__":
20
21      N_values = [10,10**2,10**3,10**4,10**5,10**6]
22
23      estimated_values = []
24      estimated_errors = []
25
26      for N in N_values:
27          estimated_pi, error = throw_darts(N)
28          estimated_values.append(estimated_pi)
29          estimated_errors.append(error)
30
31      plt.figure(figsize=(10, 6))
32      plt.errorbar(N_values, estimated_values, yerr=estimated_errors, fmt="o", capsize=5,
        label="Estimation")
33      plt.axhline(np.pi, color='red', linestyle='--', label="Exact")
34      plt.xlabel(r'$N$', fontsize=12)
35      plt.xscale('log')
36      plt.legend()
37      plt.grid(alpha=0.3)
38      plt.savefig(f"estimated_pi_.pdf")
```

書いたコードを実行し, 数値結果を見てみよう

- 投擲回数Nを上げていくと厳密解へ近づくはずである
- エラーバーの大きさは$o\left(\frac{1}{\sqrt{N}}\right)$になるはずである

さらに勉強したい人向けに

- Pythonによる計算物理学の実践的かつ比較的新しい教科書として
  - 野本拓也, 是常隆, 有田亮太郎「実践計算物理学」
  - 大槻純也「Pythonによる計算物理」
- どちらかといえばPython言語そのものに重きをおいた教科書として
  - B. Miller, D. Ranum, J. Anderson「Pythonで学ぶプログラミング入門」
- Pythonらしいコーディングの作法をはじめ、Python言語について体系的に学びたい人向けの文献として
  - Brett Slatkin「Effective Python」
  - Bill Lubanovic「入門 Python3」