FK4026, Statistik:

Bakgrundsmaterial till de första två datorövningarna

Anton Ljungdahl
anton.ljungdahl@fysik.su.se

21 augusti 2019

Till att börja med ska jag säga att det här kursmomentet antar att man är hyfsat bekväm med NumPy/SciPy och programmeringskoncept överlag. Om det känns helt främmande när jag pratar om t. ex "en funktion som returnerar en array med N element", eller motsvarande kodexempel:

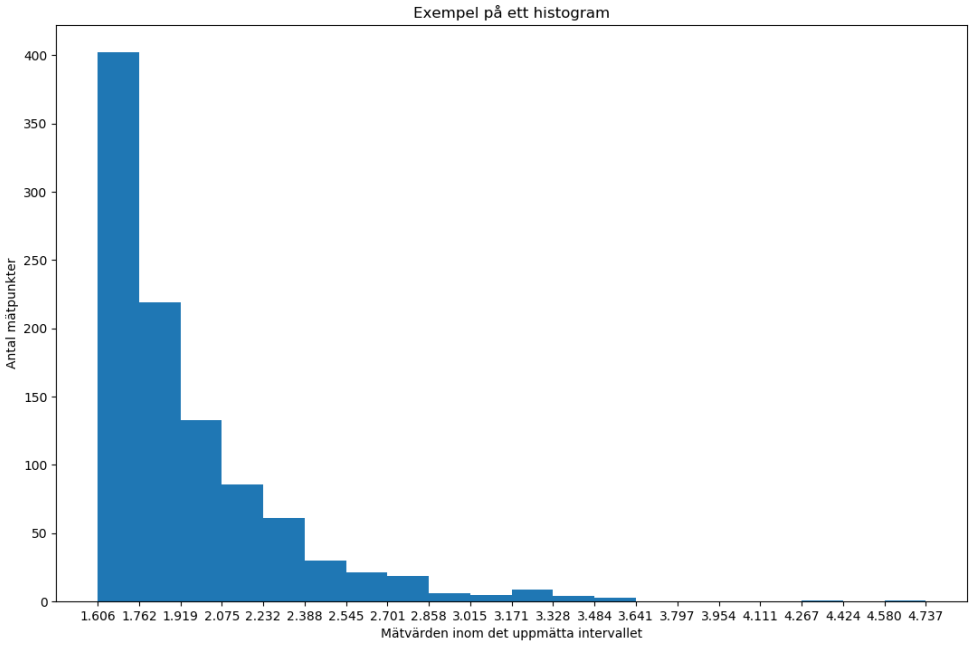
```
array = someFunctionReturningArray(length=N),
```

så behöver grunderna för programmering och Python repeteras.

Vill man ta reda på specifika saker angående NumPy/SciPy (eller bara Python) är det oftast enklast att göra en sökning på valfri sökmotor (t. ex Google eller DuckDuckGo). Vill jag hitta dokumentationen för hur man plottar histogram söker jag på t. ex på "matplotlib histogram". Det är också bra att söka på eventuella felmeddelanden man stöter på. Om jag t. ex får ett fel i stil med "could not load data from file 'dist1.txt'" så skulle jag göra en sökning i stil med: "np.loadtxt could not load from file". Då får man vanligtvis upp trådar på stackoverflow där folk har liknande problem och fått hjälp med att lösa dem.

1 Histogram

Ett histogram beskriver hur en numerisk datamängd är fördelad. Fördelad över vad? Jo över det intervall i vilket datamängden ligger. Det naturliga för fysiker är att datamängden består av mätdata från något experiment. Det vi gör för att konstruera ett histogram är att dela upp intervallet i mindre intervall, så kallade "binnar" (eng. "bins"). Sen räknar vi hur många mätpunkter vi fick i en viss bin. Vi ser ett exempel på ett histogram i Figur 1. I det här fallet är mätintervallet $[1.606, 4.737]$, dvs våra mätdata ligger i det intervallet (x-axeln i figuren). På y-axeln har vi *antalet* mätpunkter i en bin. I det här exemplet har vi 401 mätpunkter som ligger i den första "binnen", dvs i intervallet $[1.606, 1.762]$.



Figur 1: Exempel på ett histogram

Det kan vara bra att skaffa sig lite intuition om varför histogram är värdefulla att hålla på med. Antag att vår mätdata representerar resultaten av en kontinuerlig slumpvariabel X. I gränsen då antalet mätpunkter går mot oändligheten, och då längden på bin-intervall går mot noll, så representerar histogrammet precis sannolikhetsfördelningen för X. Som tur är behöver vi inte gå mot den här gränsen för att bilda oss en uppfattning om vilken sannolikhetsfördelning som är representerad. Det är precis det här vi försöker bekanta oss med labben.

Om vi vet vilken sannolikhetsfördelning (eller gissar någon som verkar stämma bra) som ligger bakom våra mätningar, så vet vi också vilka statistiska verktyg vi kan använda för att analysera resultaten.

1.1 Histogram i matplotlib (Python)

I labben ska vi plotta histogram med matplotlib. Förutsatt att vi har importerat pyplot från matplotlib enligt standarden "`from matplotlib import pyplot as plt`" kan vi plotta ett histogram enligt Figur 1.1.

```
In [13]: counts, edges, _ = plt.hist(data,bins=100)

In [5]: len(counts), len(edges)
Out[5]: (100, 101)
```

Figur 2: Exempel på anrop av `plt.hist()`. Vi ser också att arrayen `edges` har ett element mer än `counts`.

`plt.hist(data,bins=100)` returnerar en array `counts` som innehåller antalet mätpunkter i varje bin (det första elementet är antalet mätpunkter i den första binnen osv). Vi får också tillbaka en array `edges`, i vilken varje element är en gräns till ett bin-intervall. I det tidigare exemplet (Figur 1) skulle vi alltså ha `edges[0]` $= 1.606$ och `edges[1]` $= 1.762$ och så vidare. Notera att om vi har 100 binnar måste vi ha 101 "kanter", eller intervallgränser, för binnarna. Vi kan ignorera den tredje outputen "_" i vårt sammanhang (se hjälpen för `pyplot.hist()` om ni vill veta mer).

Som alltid är det bra att kolla dokumentationen för alla biblioteksfunktioner vi använder. `pyplot.hist()` har ett sant-eller-falskt argument som heter "density". Läser vi om det här argumentet i dokumentationen får vi reda på att `density=True` *normaliserar* histogrammet. I det här sammanhanget betyder det att `sum(counts)` $= 1$. Är det värdefullt för oss att representera histogrammen på det här viset?

2 Matplotlib: flera plottar i samma figur

Nedan (Figur 2) ser vi ett exempel på kod som ritar upp två figurer. I varje figur plottas ett histogram, och ovanpå histogrammet plottas någon funktion $y = f(x)$.

```
In [ ]: import numpy as np
        from matplotlib import pyplot as plt

        ...
        ...
        ...

        plt.figure(1)
        plt.hist(data1,bins=Nbins1)
        plt.plot(x1,y1)
        plt.show()

        figure(2)
        plt.hist(data2,bins=Nbins2)
        plt.plot(x2,y2)
        plt.show()
```

Figur 3: Kodexmpel på hur man ritar upp två figurer, där varje figur har två plottar (ett histogram och en "vanlig" $y = f(x)$ plot)

Precis som att ett histogram kräver en array med data som input, så behöver vi två array för en x, y-plot. En array för x, och en array för y, och båda arrayerna måste ha *lika många element*. I NumPy/SciPy finns det många funktioner som tar en array x som argument, och spottar ut en ny array (som vi lämpligen kan kalla för y), precis en representation av $y = f(x)$. För att kunna använda en sådan funktion måste vi alltså konstruera en array x. Ett vanligt sätt att göra det är att använda `linspace()`-funktionen i NumPy (sök på "numpy linspace").

Det vi vill göra i den här labben är ju att rita upp en gissning av en sannolikhetsfördelning ovanpå ett histogram. Vi kan då konstuera vår x-axel med hjälp av t. ex `linspace()`. Men vi har också fått en annan lämplig array direkt genom att konstruera histogrammet. Vilken då?

3 Teoretiska sannolikhetsfördelningar i NumPy/SciPy

Ett sätt att rita upp en teoretisk täthetsfunktion (engelska "probability density function" (PDF)) är att skriva in det analytiska uttrycket direkt. För en exponentialfördelning skulle vi till exempel kunna skriva `y = exp(-x)`. Om x är en array får vi då tillbaka en array y där element i har värde `y[i] = exp(-x[i])`.

Men det finns också en mängd biblioteksfunktioner som gör det här åt oss, och många andra saker. Jag rekommenderar *klasserna* rörande sannolikhets-

fördelningar i `scipy.stats`-biblioteket (sök på "scipy stats"). De här klasserna har flera *metoder* ("medlemsfunktioner") som är användbara i den här labben. I Figur 3 nedan ser vi några exempel på hur den teoretiska exponentialfördelningen, så som den är implementerad i `scipy.stats`, kan användas. Notera att de fördelningar vi tittar på i labben inte nödvändigtvis är exponentialfördelningen.

```
In [ ]: import numpy as np
        import scipy.stats as stats

        ...

        # skapa en array motsvarande
        # täthetsfunktionen y = exp(-x)
        y = stats.expon.pdf(x)

        # fördelningsfunktionen F(t) := P(X <= t),
        # på engelska:
        # Cumulative Distribution Function (CDF).
        # Dvs integralen av täthetsfunktionen
        # från -inf till t
        F_at_t = stats.expon.cdf(t)

        # skapa en array med N element där
        # varje element är ett slumptal
        # taget ur exponentialfördelningen
        vi = stats.expon.rvs(size=N)
```

Figur 4: Kodexempel på hur en sannolikhetsfördelning i `scipy.stats` kan användas.