stats_ch08_probability

August 2, 2024

1 Modern statistics: Intuition, Math, Python, R

1.1 Mike X Cohen (sincxpress.com)

1.1.1 https://www.amazon.com/dp/B0CQRGWGLY

Code for chapter 8

---

2 About this code file:

2.0.1 This notebook will reproduce most of the figures in this chapter (some figures were made in Inkscape), and illustrate the statistical concepts explained in the text. The point of providing the code is not just for you to recreate the figures, but for you to modify, adapt, explore, and experiment with the code.

2.0.2 Solutions to all exercises are at the bottom of the notebook.

This code was written in google-colab. The notebook may require some modifications if you use a different IDE.

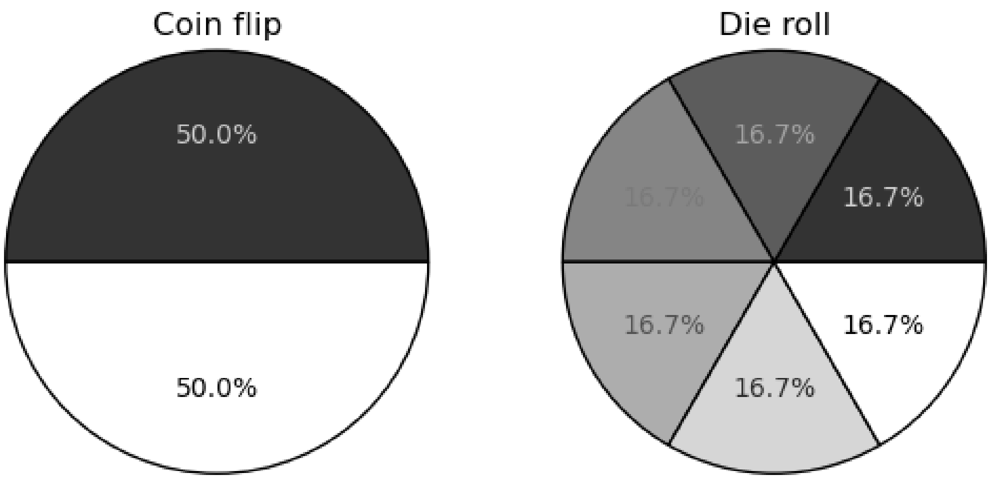
[1]:
```
# import libraries and define global settings
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt

# define global figure properties used for publication
import matplotlib_inline.backend_inline
```

3 Figure 8.1: Pie charts for margin figure

[2]:
```
# a bit of code to show how to get equal areas for some number of pie slices
k = 2
np.tile(1/k,k)
```

[2]:
```
array([0.5, 0.5])
```

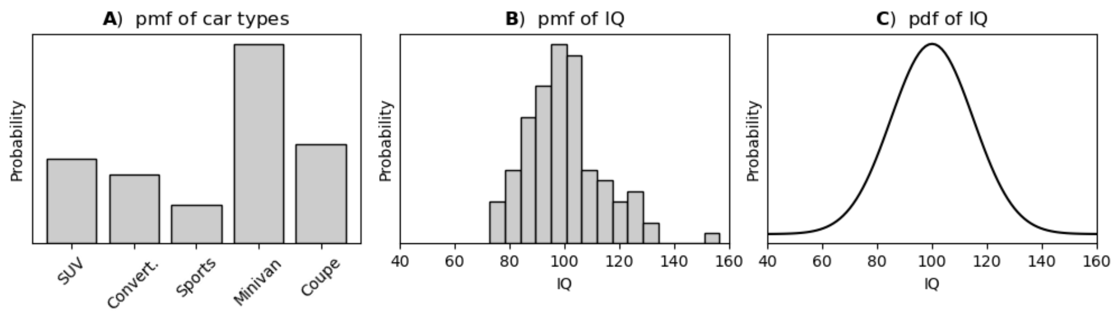
[4]:
```
_,axs = plt.subplots(1,2,figsize=(6,4))

for a,k in zip(axs,[2,6]):

  # draw the pie (and export the patches and text to update the color)
  patches,_,autotexts = a.pie(np.tile(1/k,k),autopct='%.
↪1f%%',wedgeprops={'edgecolor':'k'},
        colors=np.linspace((.2,.2,.2),(1,1,1),k))

  for autotext, patch in zip(autotexts,patches):
    inverse_color = 1 - np.array(patch.get_facecolor())
    inverse_color[-1] = 1 # invert the color, but not the alpha
    autotext.set_color(inverse_color)

axs[0].set_title('Coin flip',y=.9)
axs[1].set_title('Die roll',y=.9)

plt.tight_layout()
#plt.savefig('prob_probsInPies.png')
plt.show()
```



4 Figure 8.4: Visualizing probability masses and densities

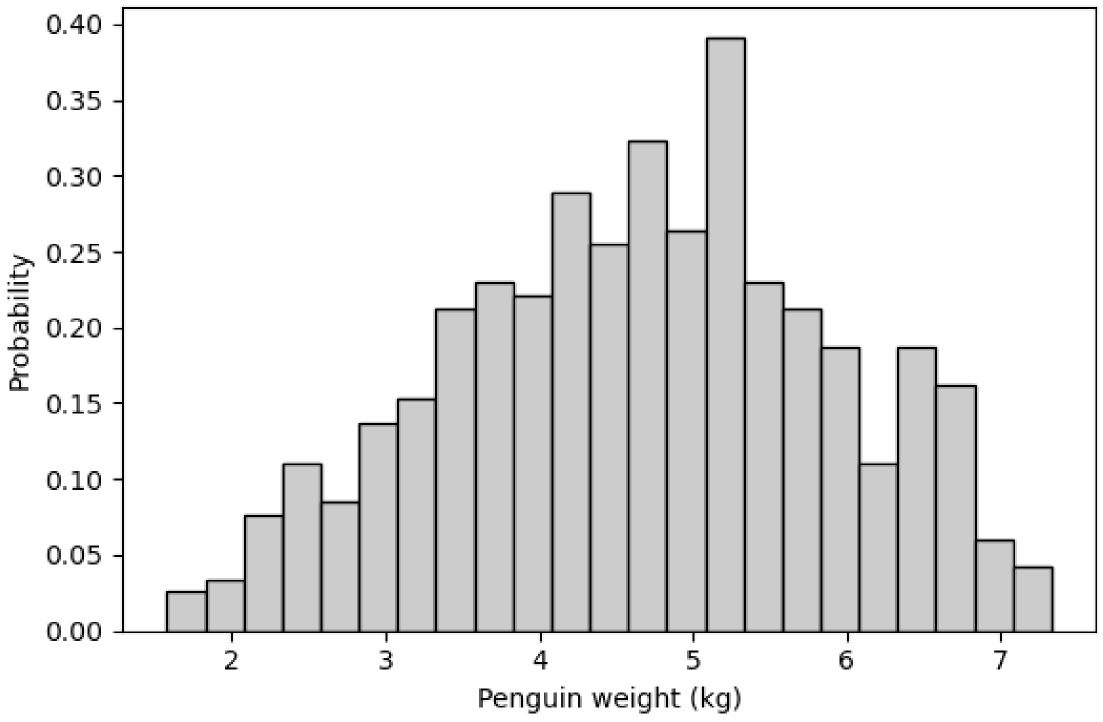
[5]:
```
# categorical probability data
categoryLabels = [ 'SUV','Convert.','Sports','Minivan','Coupe' ]
categoryData = np.random.randint(low=5,high=30,size=len(categoryLabels)).
↪astype(np.float64)
```

```python
categoryData /= np.sum(categoryData)

# discrete numerical probability data
empiricalIQ = np.random.normal(loc=100,scale=15,size=100)

# continuous (analytic) probability data
x = np.linspace(-4,4,101)
continuousData = stats.norm.pdf(x)*15 + 100

### visualize!
_,axs = plt.subplots(1,3,figsize=(10,3))

# categorical data in bars
axs[0].bar(categoryLabels,categoryData,color=[.8,.8,.8],edgecolor='k')
axs[0].set_title(r'$\bf{A}$)  pmf of car types')
axs[0].set(ylabel='Probability',yticks=[])
axs[0].tick_params(axis='x',rotation=45)

# empirical probability data that estimate a density, still in bars
axs[1].hist(empiricalIQ,bins=15,color=[.8,.8,.8],edgecolor='k')
axs[1].set(xlabel='IQ',ylabel='Probability',yticks=[],xlim=[40,160])
axs[1].set_title(r'$\bf{B}$)  pmf of IQ')

# analytical probability density as a line
axs[2].plot(x*15+100,continuousData,'k')
axs[2].set(xlabel='IQ',ylabel='Probability',yticks=[],xlim=[40,160])
axs[2].set_title(r'$\bf{C}$)  pdf of IQ')

plt.tight_layout()
#plt.savefig('prob_visualizeMassDensity.png')
plt.show()
```



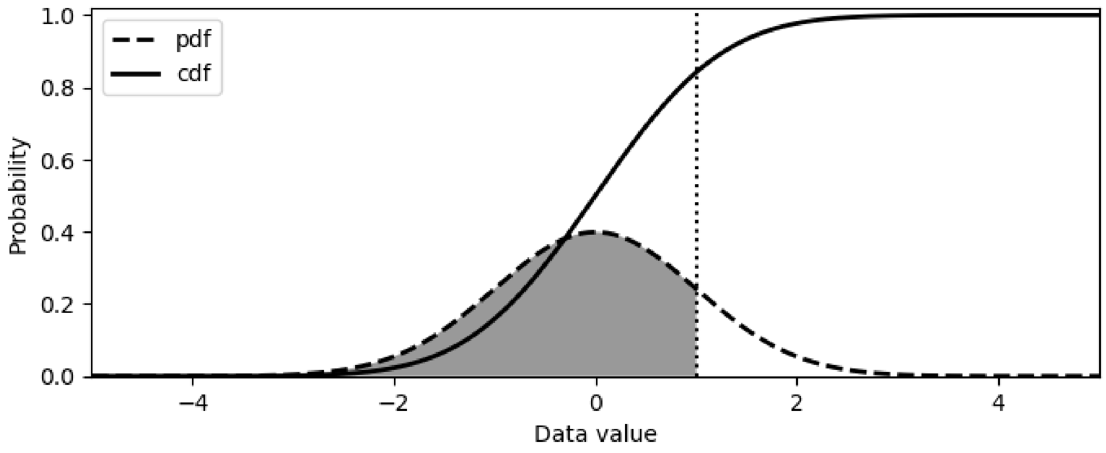
5 Figure 8.5: Probability mass function of penguin weights

[6]:
```
penguins = np.arctanh(np.random.uniform(size=473)*1.8-.9)*2+4.5

bin_edges = np.arange(np.min(penguins),np.max(penguins),step=.25)

plt.figure(figsize=(6,4))
plt.hist(penguins,bins=bin_edges,density=True,
         color=[.8,.8,.8],edgecolor='k')
plt.xlabel('Penguin weight (kg)')
plt.ylabel('Probability')

plt.tight_layout()
#plt.savefig('prob_penguinWeightProb.png')
plt.show()
```



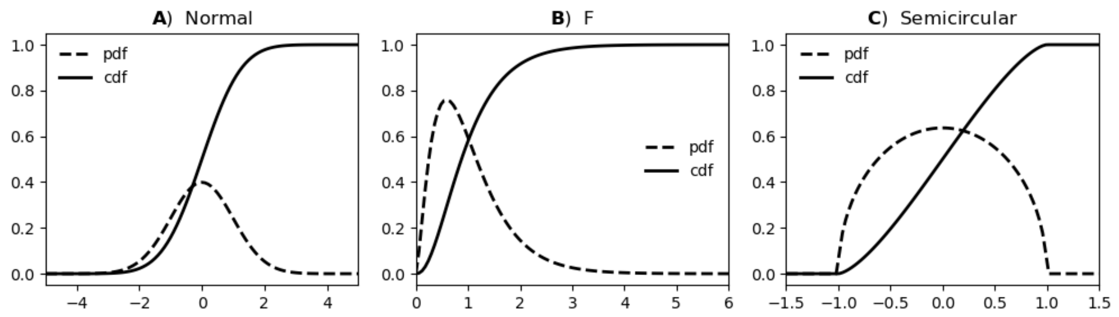
6 Figure 8.6: pdf and cdf

[7]:
```python
# distribution
x = np.linspace(-5,5,501)
pdf = stats.norm.pdf(x)
cdf = stats.norm.cdf(x)

_,ax = plt.subplots(1,figsize=(7,3))

# patch for the summed area
from matplotlib.patches import Polygon
bndi = np.argmin(np.abs(x-1))
dots = np.zeros((bndi+2,2))
for i in range(bndi+1):
  dots[i,:] = x[i],pdf[i]
dots[-1,:] = x[bndi],0
ax.add_patch(Polygon(dots[:,[0,1]],facecolor='k',alpha=.4))

# plot the functions
ax.plot(x,pdf,'k--',linewidth=2,label='pdf')
ax.plot(x,cdf,'k',linewidth=2,label='cdf')
ax.axvline(1,color='k',linestyle=':')

# make the plot a bit nicer
ax.set(xlim=x[[0,-1]],ylim=[0,1.02],xlabel='Data value',ylabel='Probability')
plt.legend(loc='upper left')

plt.tight_layout()
#plt.savefig('prob_pdf2cdf.png')
plt.show()
```



7 Figure 8.7: pdf/cdf combos of some example distributions

[8]:
```
_,axs = plt.subplots(1,3,figsize=(10,3))

# Gaussian
x = np.linspace(-5,5,101)
axs[0].plot(x,stats.norm.pdf(x),'k--',linewidth=2)
axs[0].plot(x,stats.norm.cdf(x),'k',linewidth=2)
axs[0].set_title(r'$\bf{A}$)  Normal')
axs[0].set_xlim(x[[0,-1]])

# F
x = np.linspace(0,6,101)
axs[1].plot(x,stats.f.pdf(x,5,100),'k--',linewidth=2)
axs[1].plot(x,stats.f.cdf(x,5,100),'k',linewidth=2)
axs[1].set_title(r'$\bf{B}$)  F')
axs[1].set_xlim(x[[0,-1]])

# semicircular
x = np.linspace(-1.5,1.5,101)
axs[2].plot(x,stats.semicircular.pdf(x),'k--',linewidth=2)
axs[2].plot(x,stats.semicircular.cdf(x),'k',linewidth=2)
axs[2].set_title(r'$\bf{C}$)  Semicircular')
axs[2].set_xlim(x[[0,-1]])

# legends
for a in axs: a.legend(['pdf','cdf'],frameon=False)

plt.tight_layout()
#plt.savefig('prob_examplePdfCdf.png')
plt.show()
```

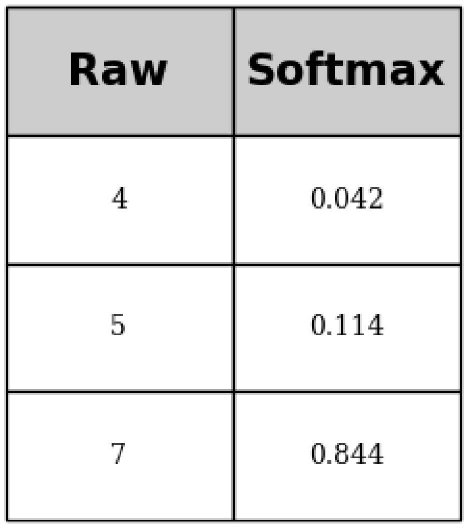


8 Figure 8.11: Softmax vs. "raw" probability

[9]:
```
x = [4,5,7]

# softmax transformation
num = np.exp(x)
den = np.sum( np.exp(x) )
sigma = num / den

print(sigma)
```

```
[0.04201007 0.1141952  0.84379473]
```

[10]:
```
# table data
colLabs = ['Raw','Softmax']

tabdat = []
for xi,si in zip(x,sigma):
  tabdat.append([f'{xi:.0f}',f'{si:.3f}'])

# draw the table
fig, ax = plt.subplots(figsize=(2.7,3))
ax.set_axis_off()
ht = ax.table(
        cellText   = tabdat,
        colLabels  = colLabs,
        colColours = [(.8,.8,.8)] * len(colLabs),
        cellLoc    = 'center',
        loc        = 'upper left',
        )

# some adjustments to the fonts etc
ht.scale(1,3.8)
ht.auto_set_font_size(False)
ht.set_fontsize(14)

from matplotlib.font_manager import FontProperties
for (row, col), cell in ht.get_celld().items():
  cell.set_text_props(fontproperties=FontProperties(family='serif'))
  if row==0: cell.
↪set_text_props(fontproperties=FontProperties(weight='bold',size=16))

# export
plt.tight_layout()
#plt.savefig('prob_table_softmax.png', bbox_inches='tight')
plt.show()
```

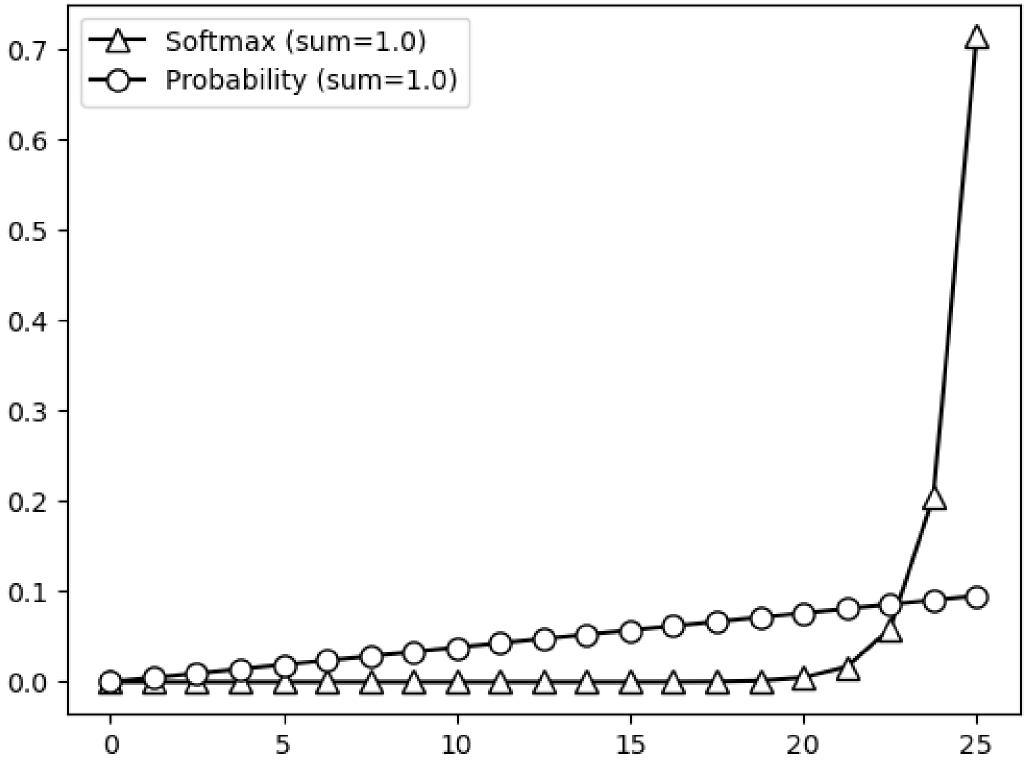


Raw	Softmax
4	0.042
5	0.114
7	0.844

[11]:
```python
# the data (marble color counts)
counts = np.array([ 40,30,20 ])

# softmax
num = np.exp(counts)
den = np.sum( np.exp(counts) )
sigma = num / den

# standard probabilities
probs = 100*counts / np.sum(counts)

# print the results
print('Softmax:')
print(sigma)

print(' ')
print('Probabilities:')
print(probs)
```

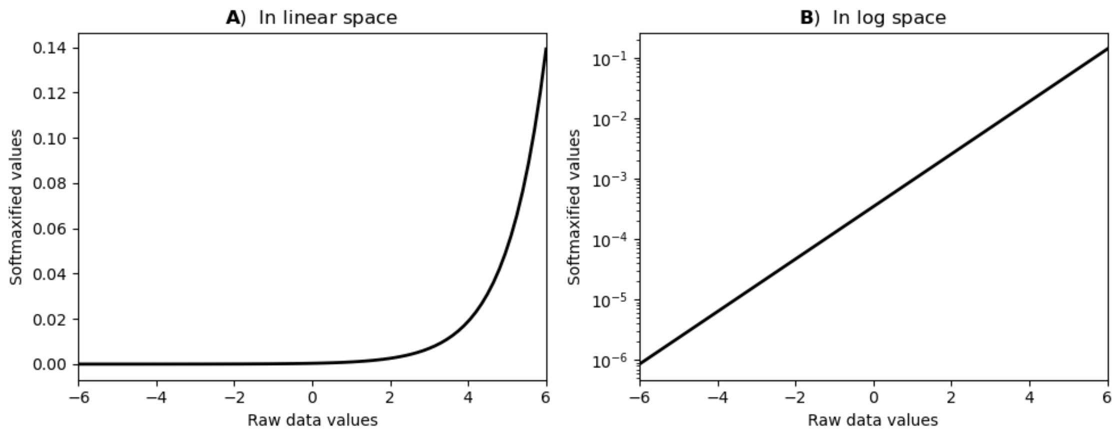
```
Softmax:
[9.99954600e-01 4.53978686e-05 2.06106005e-09]

Probabilities:
[44.44444444 33.33333333 22.22222222]
```

[12]:
```
# with raw counts
x = np.linspace(0,25,21)
s = np.exp(x) / np.sum(np.exp(x))
p = x / np.sum(x)

plt.plot(x,s,'k^-',markerfacecolor='w',markersize=8,label=f'Softmax (sum={np.
 ↪sum(s)})')
plt.plot(x,p,'ko-',markerfacecolor='w',markersize=8,label=f'Probability (sum={np.
 ↪sum(p)})')
plt.legend()
plt.show()
```



9 Figure 8.10: Softmax in linear and log space

[13]:
```
# with raw numerical data values
x = np.linspace(-6,6,81)
s = np.exp(x) / np.sum(np.exp(x))

_,axs = plt.subplots(1,2,figsize=(10,4))

for a in axs:
  a.plot(x,s,'k-',linewidth=2)
  a.set(xlabel='Raw data values',ylabel='Softmaxified values',xlim=x[[0,-1]])

axs[1].set_yscale('log')
axs[0].set_title(r'$\bf{A}$)  In linear space')
axs[1].set_title(r'$\bf{B}$)  In log space')

plt.tight_layout()
#plt.savefig('prob_softmaxNumbers.png')
plt.show()
```



10 Exercise 1

[14]:
```
# re-create the pdf
# (note: I'm using "4" in the variable names for comparisons in subsequent␣
 ↪exercises)
x4 = np.linspace(-4,4,400)
pdf4 = stats.norm.pdf(x4)

# normalize by dx
pdf4N = pdf4*(x4[1]-x4[0])

# print sums
print(f'Sum over pdf: {np.sum(pdf4):.3f}')
print(f'Sum over normalized pdf: {np.sum(pdf4N):.3f}')
```

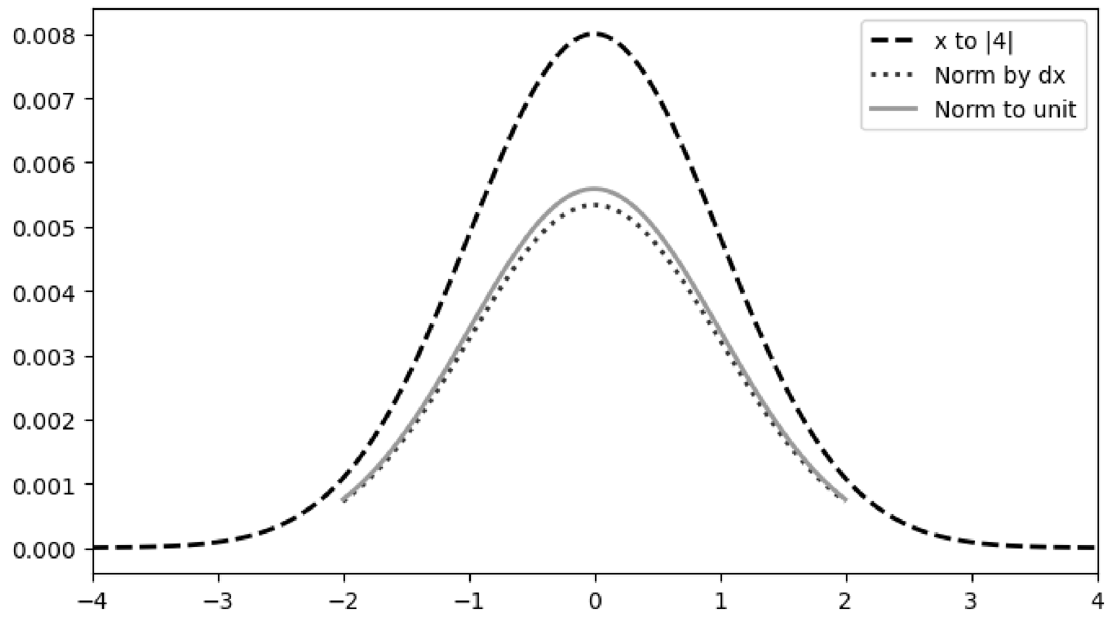
```
Sum over pdf: 49.872
Sum over normalized pdf: 1.000
```

11 Exercise 2

[15]:
```
# now with a restricted range
x2 = np.linspace(-2,2,300)
pdf2 = stats.norm.pdf(x2)

# normalize by dx
pdf2N = pdf2*(x2[1]-x2[0])

# normalize to sum=1 ("U" is for "unit")
pdf2U = pdf2 / np.sum(pdf2)

# print sums
print(f'Sum over pdf normalized by dx : {np.sum(pdf2N):.3f}')
print(f'Sum over pdf normalized by sum: {np.sum(pdf2U):.3f}')
```

```
Sum over pdf normalized by dx : 0.955
Sum over pdf normalized by sum: 1.000
```

[16]:
```
# plot
plt.figure(figsize=(7,4))
plt.plot(x4,pdf4N,'k--',linewidth=2,label='x to |4|')
plt.plot(x2,pdf2N,':',color=(.2,.2,.2,),linewidth=2,label='Norm by dx')
plt.plot(x2,pdf2U,color=(.6,.6,.6),linewidth=2,label='Norm to unit')
plt.xlim(x4[[0,-1]])
plt.legend()

plt.tight_layout()
#plt.savefig('prob_ex2.png')
plt.show()
```

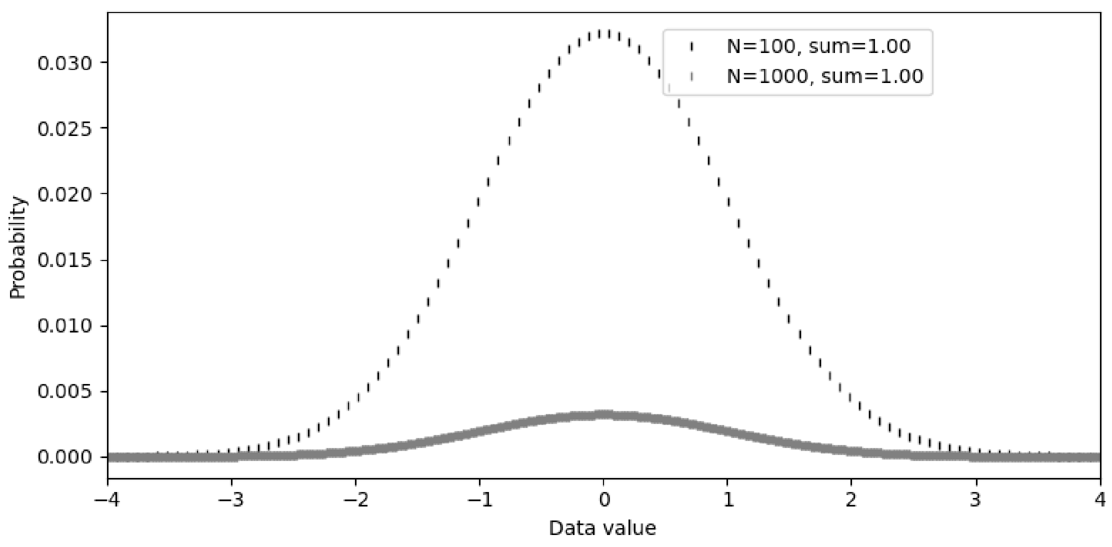


[17]:
```python
# From the explanations about exercise 1:
print(f'Sum over normalized pdf: {np.sum(pdf4N):.9f}')
```

```
Sum over normalized pdf: 0.999939305
```

12 Exercise 3

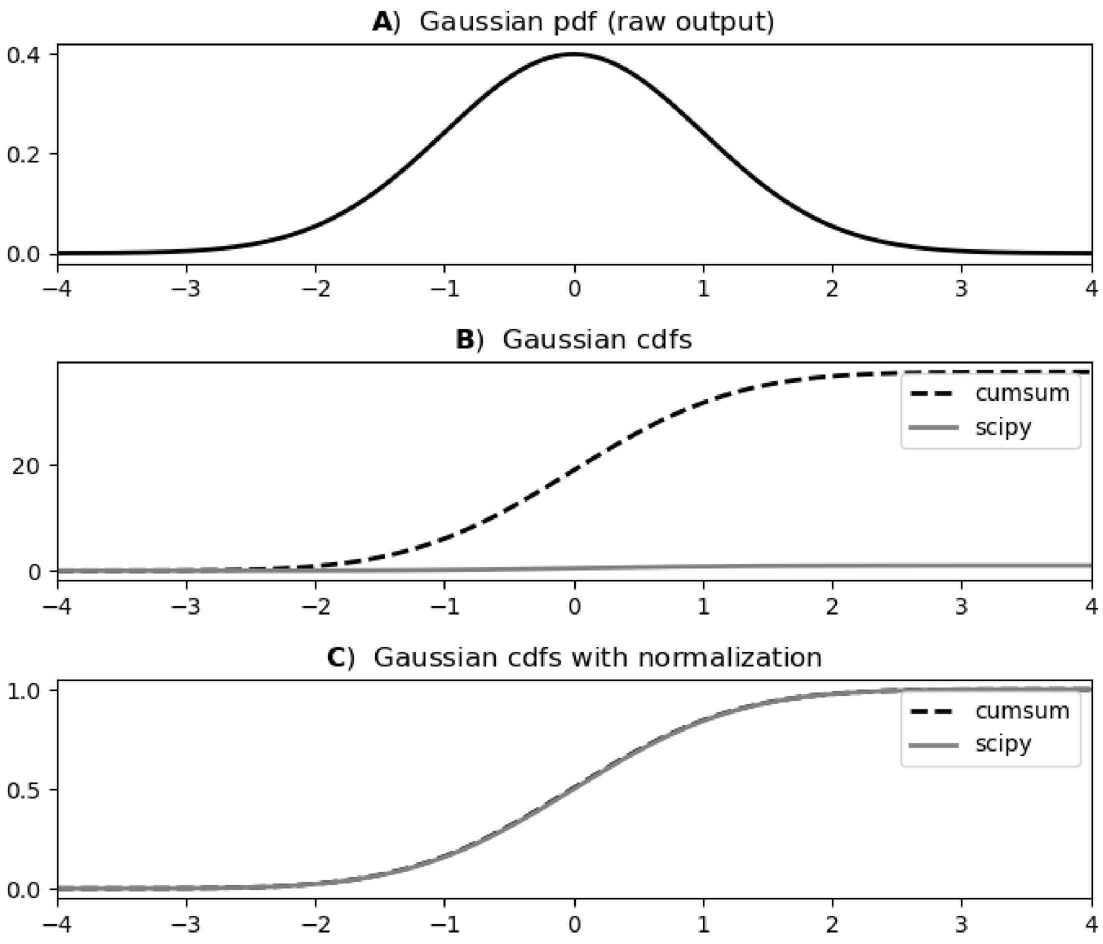
[18]:
```python
# parameters
npnts = [100,1000]
colors = ['k','gray']

plt.figure(figsize=(8,4))

# the loop that does it all
for resolution,linecolor in zip(npnts,colors):

  # create x-axis grid
  x = np.linspace(-4,4,resolution)

  # evaluate "raw" pdf
  pdf = stats.norm.pdf(x)

  # normalize by dx
  pdfN = pdf*(x[1]-x[0])

  # plot the normalized pdf
  plt.plot(x,pdfN,'|',linewidth=2,color=linecolor,markersize=4,
           label=f'N={resolution}, sum={np.sum(pdfN):.2f}')

plt.xlim(x[[0,-1]])
plt.legend(bbox_to_anchor=[.55,.8])
plt.xlabel('Data value')
plt.ylabel('Probability')

plt.tight_layout()
#plt.savefig('prob_ex3.png')
plt.show()
```



13 Exercise 4

[19]:
```python
# create cdf from pdf
x = np.linspace(-4,4,300)
pdf = stats.norm.pdf(x)

# python's cdf
cdf_sp = stats.norm.cdf(x)

# manual computation
cdf_my = np.cumsum(pdf)

_,axs = plt.subplots(3,1,figsize=(7,6))

axs[0].plot(x,pdf,'k',linewidth=2)
axs[0].set(xlim=x[[0,-1]])
axs[0].set_title(r'$\bf{A}$)  Gaussian pdf (raw output)')


axs[1].plot(x,cdf_my,'k--',linewidth=2,label='cumsum')
axs[1].plot(x,cdf_sp,color='gray',linewidth=2,label='scipy')
axs[1].set(xlim=x[[0,-1]])
axs[1].set_title(r"$\bf{B}$)  Gaussian cdfs")
axs[1].legend()

# normalized by dx
cdf_myN = np.cumsum(pdf) * (x[1]-x[0])
```

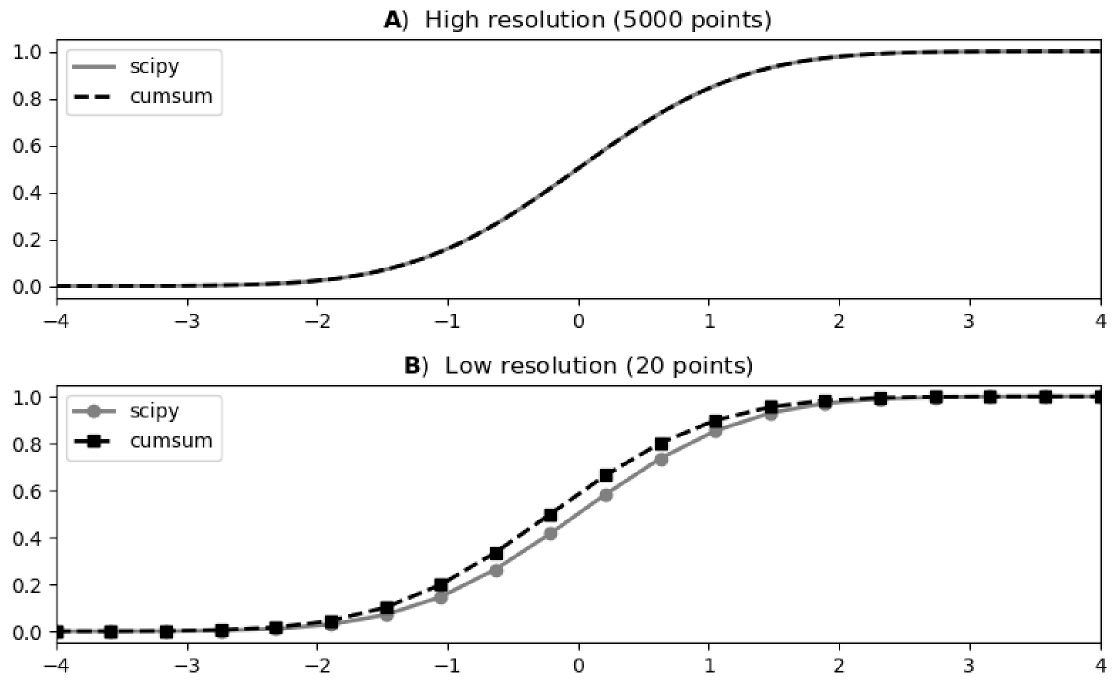
```
axs[2].plot(x,cdf_myN,'k--',linewidth=2,label='cumsum')
axs[2].plot(x,cdf_sp,color='gray',linewidth=2,label='scipy')
axs[2].set(xlim=x[[0,-1]])
axs[2].set_title(r"$\bf{C}$)  Gaussian cdfs with normalization")
axs[2].legend()

plt.tight_layout()
#plt.savefig('prob_ex4.png')
plt.show()
```



14 Exercise 5

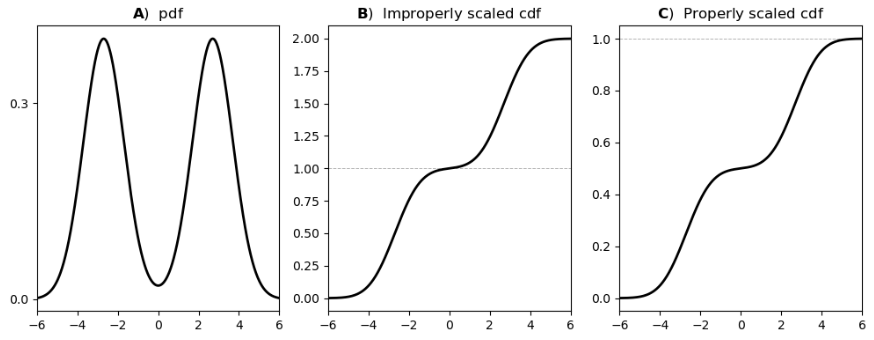
[20]:
```
# cdfs with different resolutions
x = np.linspace(-4,4,5000)
pdf = stats.norm.pdf(x)
```

```
# python's cdf
cdf_sp = stats.norm.cdf(x)

# manual computation (normalized)
cdf_my = np.cumsum(pdf) * (x[1]-x[0])

_,axs = plt.subplots(2,1,figsize=(8,5))

axs[0].plot(x,cdf_sp,color='gray',linewidth=2,label='scipy')
axs[0].plot(x,cdf_my,'k--',linewidth=2,label='cumsum')
axs[0].set(xlim=x[[0,-1]])
axs[0].set_title(r"$\bf{A}$)  High resolution (5000 points)")
axs[0].legend()

x = np.linspace(-4,4,20)
cdf_sp = stats.norm.cdf(x)
cdf_my = np.cumsum(stats.norm.pdf(x)) * (x[1]-x[0])

axs[1].plot(x,cdf_sp,'o-',color='gray',linewidth=2,label='scipy')
axs[1].plot(x,cdf_my,'ks--',linewidth=2,label='cumsum')
axs[1].set(xlim=x[[0,-1]])
axs[1].set_title(r"$\bf{B}$)  Low resolution (20 points)")
axs[1].legend()

plt.tight_layout()
#plt.savefig('prob_ex5.png')
plt.show()
```



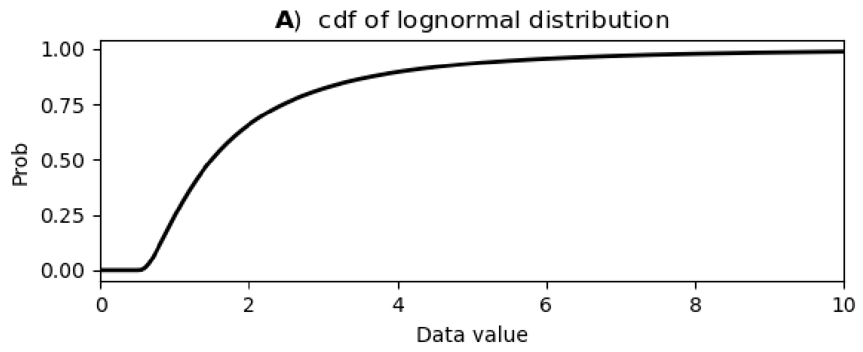
15 Exercise 6

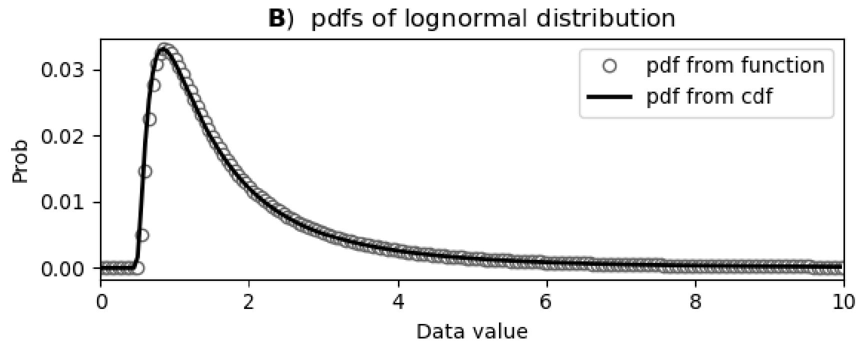
[24]:
```python
# the non-standard pdf
x = np.linspace(-6,6,1001)
pdf = stats.norm.pdf(x-2.7) + stats.norm.pdf(x+2.7)

# simple scaling by dx
cdf = np.cumsum(pdf) * np.mean(np.diff(x))

# better scaling by first unit-sum-normalizing pdf
pdfN = pdf / np.sum(pdf)
cdfN = np.cumsum(pdfN)

# another option: scale cdf to have a final value of 1
# cdfN = cdf / cdf[-1]

axs = plt.subplots(1,3,figsize=(10,4))[1]

axs[0].plot(x,pdf,'k',linewidth=2)
axs[0].set(yticks=[0,.3])
axs[0].set_title(r'$\bf{A}$)  pdf')

axs[1].plot(x,cdf,'k',linewidth=2)
axs[1].axhline(1,color=[.7,.7,.7],linewidth=.7,linestyle='--',zorder=-1)
axs[1].set_title(r'$\bf{B}$)  Improperly scaled cdf')

axs[2].plot(x,cdfN,'k',linewidth=2)
axs[2].axhline(1,color=[.7,.7,.7],linewidth=.7,linestyle='--',zorder=-1)
axs[2].set_title(r'$\bf{C}$)  Properly scaled cdf')

for a in axs:
  a.set(xlim=x[[0,-1]])

plt.tight_layout()
#plt.savefig('prob_ex6.png')
plt.show()
```



16 Exercise 7

[25]:
```
# create a cdf
x = np.linspace(0,10,200)
cdf = stats.lognorm.cdf(x,1,1/2)

# empirical pdf via difference (D=difference)
pdfD = np.diff(cdf)

# analytical pdf (A=analytical)
pdfA = stats.lognorm.pdf(x,1,1/2)
pdfA *= x[1]-x[0]

_,axs = plt.subplots(2,1,figsize=(6,5))
axs[0].plot(x,cdf,'k',linewidth=2)
axs[0].set_title(r'$\bf{A}$)  cdf of lognormal distribution')

axs[1].plot(x,pdfA,'o',color=(.4,.4,.4),markerfacecolor='w',
            linewidth=2,label='pdf from function')
axs[1].plot(x[:-1],pdfD,'k',linewidth=2,label='pdf from cdf')
axs[1].set_title(r'$\bf{B}$)  pdfs of lognormal distribution')
axs[1].legend()

for a in axs:
  a.set(xlim=x[[0,-1]],xlabel='Data value',ylabel='Prob')

plt.tight_layout()
#plt.savefig('prob_ex7.png')
plt.show()
```



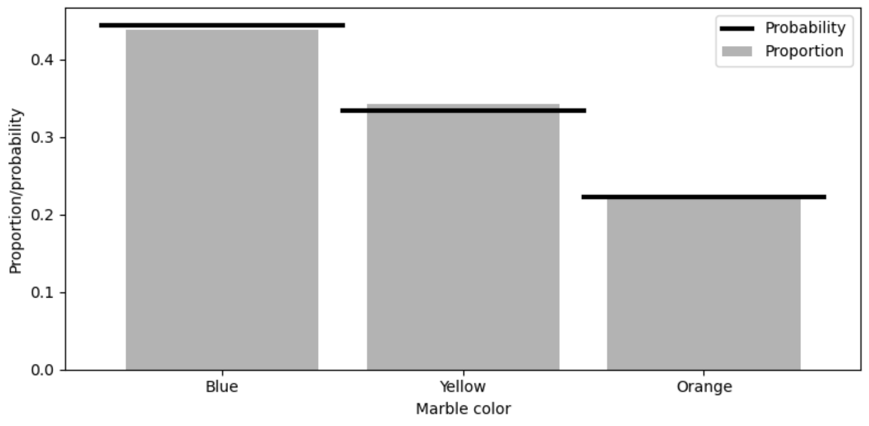


```
[ ]: # Note: The two pdf's appear to have some mismatch.
     #       Try changing the resolution from 200 to 2000, and then to 20.
```

17 Exercise 8

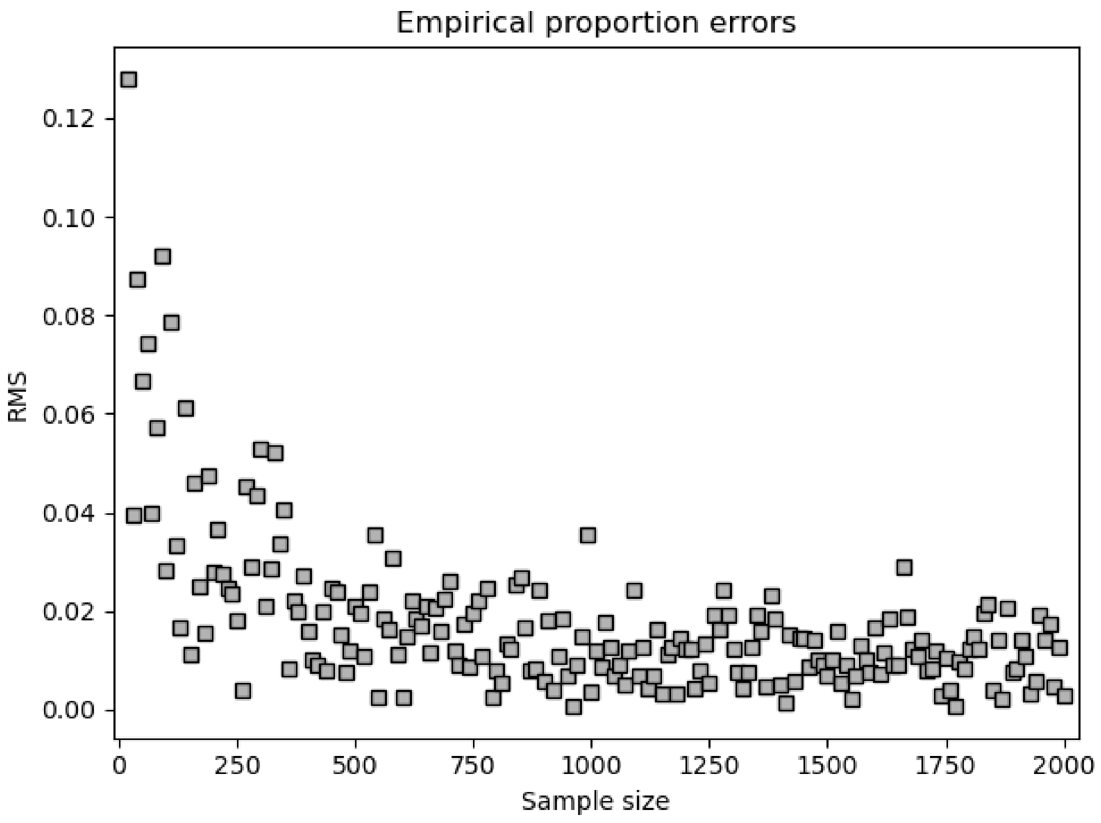
```
[26]: # colored marble counts
      blue   = 40
      yellow = 30
      orange = 20
      totalMarbs = blue + yellow + orange

      # put them all in a jar
      jar = np.hstack((1*np.ones(blue),2*np.ones(yellow),3*np.ones(orange)))

      # now we draw 500 marbles (with replacement)
      numDraws = 500
      marbSample = np.zeros(numDraws)

      # I wrote the experiment in a loop
      # to show how it maps onto a real-world experiment
      # of data points being gathered one at a time.
      for drawi in range(numDraws):
```

```
    # generate a random integer to draw
    randmarble = int(np.random.rand()*len(jar))

    # store the color of that marble
    marbSample[drawi] = jar[randmarble]

# but in practice, you can implement it without a loop
marbSample = np.random.choice(jar,size=numDraws,replace=True)

# now we need to know the proportion of colors drawn
propBlue = sum(marbSample==1) / numDraws
propYell = sum(marbSample==2) / numDraws
propOran = sum(marbSample==3) / numDraws

# plot those against the theoretical probability
plt.figure(figsize=(8,4))
plt.bar([1,2,3],[ propBlue, propYell, propOran ],label='Proportion',color=(.7,.
 ↪7,.7))
plt.plot([0.5, 1.5],[blue/totalMarbs, blue/
 ↪totalMarbs],'k',linewidth=3,label='Probability')
plt.plot([1.5, 2.5],[yellow/totalMarbs,yellow/totalMarbs],'k',linewidth=3)
plt.plot([2.5, 3.5],[orange/totalMarbs,orange/totalMarbs],'k',linewidth=3)

plt.xticks([1,2,3],labels=('Blue','Yellow','Orange'))
plt.xlabel('Marble color')
plt.ylabel('Proportion/probability')
plt.legend()

plt.tight_layout()
#plt.savefig('prob_ex8.png')
plt.show()
```



18 Exercise 9

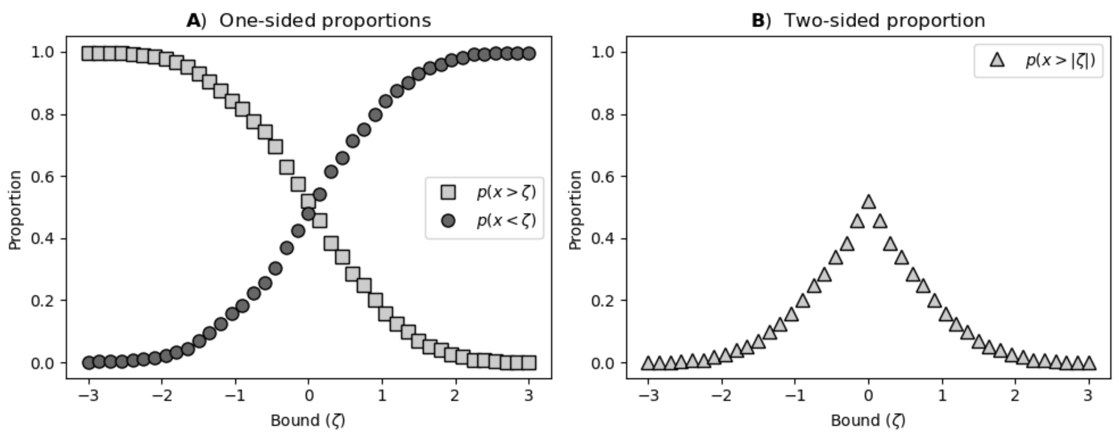
[27]:
```python
# colored marble counts
blue   = 40
yellow = 30
orange = 20
totalMarbs = blue + yellow + orange

# range of sample sizes
sampleSizes = np.arange(20,2001,step=10)

# some initializations to simplify the code in the experiment
empiProbs = np.zeros(3)
trueProbs = np.array([ blue/totalMarbs,yellow/totalMarbs,orange/totalMarbs ])

# initialize
rms = np.zeros(len(sampleSizes))

# run the experiment
for idx,thisN in enumerate(sampleSizes):
  # draw N marbles
  drawColors = np.random.choice(jar,size=thisN,replace=True)

  # compute proportion
  for ei in range(3):
    empiProbs[ei] = np.sum(drawColors==(ei+1)) / thisN

  # compute the sum of squared errors
  rms[idx] = np.sqrt( np.mean( (empiProbs-trueProbs)**2 ) )

plt.plot(sampleSizes,rms,'ks',markerfacecolor=[.7,.7,.7])
plt.xlim([sampleSizes[0]-30,sampleSizes[-1]+30])
plt.xlabel('Sample size')
plt.ylabel('RMS')
plt.title('Empirical proportion errors',loc='center')

plt.tight_layout()
#plt.savefig('prob_ex9.png')
plt.show()
```



19 Exercise 10

[28]:
```
# generate some data, compute proportions of data above some value

# simulation parameters
N = 1000 # sample size
k = 41   # number of data bins

# generage the data
data = np.random.randn(N)

# determine boundaries (based on a priori knowledge of the distribution)
bounds = np.linspace(-3,3,k) # what I call 'zeta' in the text

# initialize the results (empirical proportions)
emppropsGT = np.zeros(k)
emppropsLT = np.zeros(k)
empprops2tail = np.zeros(k)

# loop over the boundaries
```

```
for idx,bi in enumerate(bounds):

  # empirical proportions for each side separately
  emppropsGT[idx] = np.sum(data>bi) / N
  emppropsLT[idx] = np.sum(data<bi) / N

  # and for the two-sided
  empprops2tail[idx] = np.sum(data>np.abs(bi)) / N

# visualize
_,axs = plt.subplots(1,2,figsize=(10,4))

axs[0].plot(bounds,emppropsGT,'ks',markerfacecolor=(.8,.8,.
 ↪8),markersize=8,label=r'$p(x>\zeta)$')
axs[0].plot(bounds,emppropsLT,'ko',markerfacecolor=(.4,.4,.
 ↪4),markersize=8,label=r'$p(x<\zeta)$')
axs[0].legend()
axs[0].set(xlabel=r'Bound ($\zeta$)',ylim=[-.05,1.
 ↪05],ylabel='Proportion',title=r'$\bf{A}$)  One-sided proportions')

axs[1].plot(bounds,empprops2tail,'k^',markerfacecolor=(.8,.8,.
 ↪8),markersize=8,label=r'$p(x>|\zeta|)$')
axs[1].set(xlabel=r'Bound ($\zeta$)',ylim=[-.05,1.
 ↪05],ylabel='Proportion',title=r'$\bf{B}$)  Two-sided proportion')
axs[1].legend()

plt.tight_layout()
#plt.savefig('prob_ex10.png')
plt.show()
```



```
[ ]: ## Note about the above exercise:
     # I encourage you to try different distributions for generating data, but
     # you will need to adjust the proportion bounds; -3 to +3 is hard-coded to
     # be appropriate for a normal distribution.
     #
     # Consider this to be an additional way to challenge yourself :)
```

20 Exercise 11

```
[ ]: ### Here's the url:
     # https://docs.scipy.org/doc/scipy/reference/stats.html#continuous-distributions

     # All you need to do is copy the code for Figure 8.7 but replace "norm" with, e.
      ↪g., "crystalball."
     # I recommend removing the code to create the gray patches.
```