stats_ch09_sampling

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 9 (sampling)

---

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.

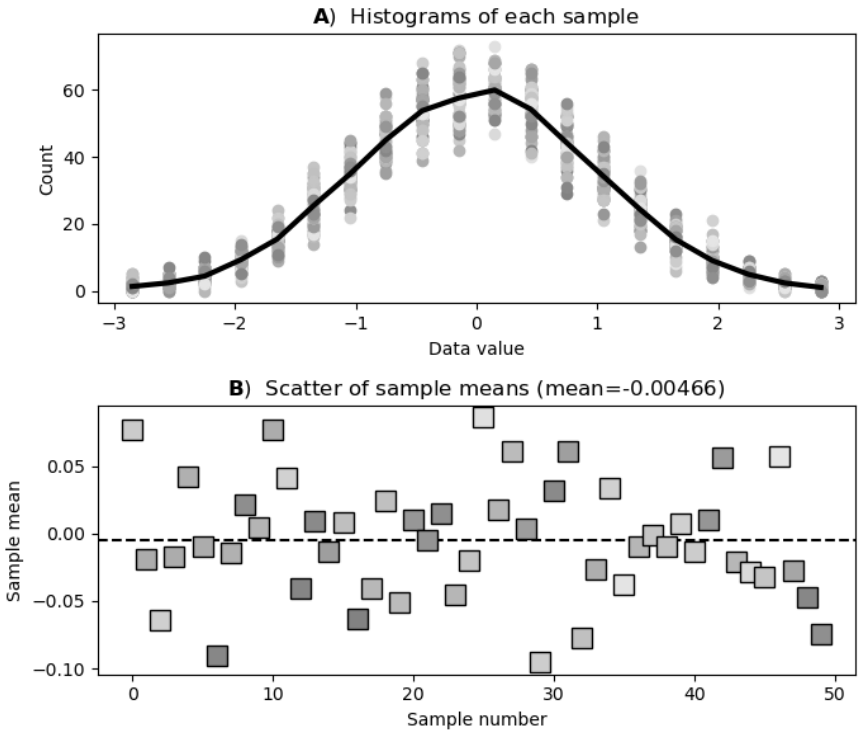
This code was written in google-colab. The notebook may require some modifications if you use a different IDE.

[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 9.1: Sampling variability in random data

[2]:
```
N = 500
nSamples = 50
kHistBins = 20

# bins for histograms
edges = np.linspace(-3,3,kHistBins+1)
```

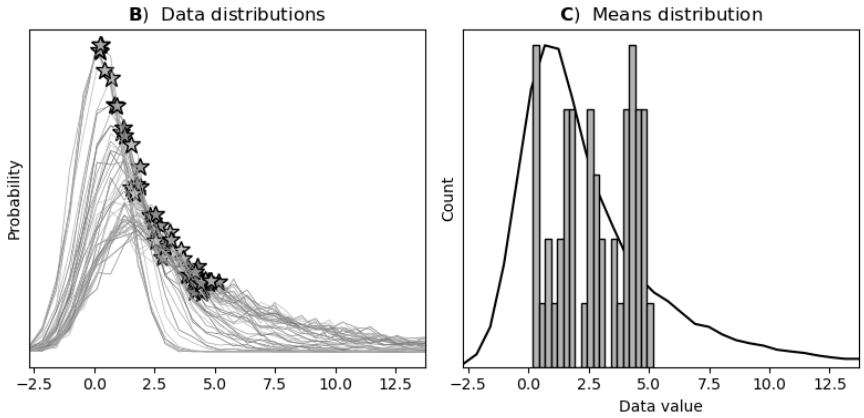
```
# declare matrices
allHistY = np.zeros((nSamples,kHistBins))
allMeans = np.zeros(nSamples)

# setup figure
f,axs = plt.subplots(2,1,figsize=(7,6))

for sampi in range(nSamples):
  # create data (parameters don't chage!)
  data = np.random.normal(loc=0,scale=1,size=N)

  # histograms
  y,x = np.histogram(data,bins=edges)
  allHistY[sampi,:] = y

  # get means
  allMeans[sampi] = np.mean(data)

  # plot
  c = np.random.uniform(low=.5,high=.9)
  axs[0].plot((x[:-1]+x[1:])/2,y,'o',color=(c,c,c))

  axs[1].plot(sampi,np.mean(data),'ks',markersize=12,markerfacecolor=(c,c,c))

# plot the average histogram
axs[0].plot((x[:-1]+x[1:])/2,np.mean(allHistY,axis=0),'k',linewidth=3)

# plot the means, and the mean of the means
axs[1].axhline(np.mean(allMeans),linestyle='--',color='k',zorder=-1)

# make the plots look nicer
axs[0].set(xlabel='Data value',ylabel='Count')
axs[0].set_title(r'$\bf{A}$)  Histograms of each sample')
axs[1].set(xlabel='Sample number',ylabel='Sample mean')
axs[1].set_title(r'$\bf{B}$)  Scatter of sample means (mean=%.3g)' %np.
 ↪mean(allMeans))

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



4 Figure 9.2: Samples and variability of sample means

[4]:
```
# number of samples
nSamples = 50

# histogram resolution
k = 30
edges = np.linspace(-3,14,31)
xx = (edges[:-1]+edges[1:])/2

# initialize output matrices
meenz  = np.zeros(nSamples) # sample averages
allYYs = np.zeros(k)        # average of histograms

_,axs = plt.subplots(1,2,figsize=(8,4))

# loop over samples
for i in range(nSamples):
  # generate random data from an exGaussian distribution
  randomX = stats.exponnorm.rvs(np.random.uniform(low=.1,high=5),size=2000)

  # get its histogram and normalize
```

```
  yy,_ = np.histogram(randomX,bins=edges)
  yy = yy/np.sum(yy)

  # average the distributions
  allYYs += yy

  # store the average of the distribution
  meenz[i] = np.mean(randomX)

  # plot the line
  rc = np.random.uniform(low=.4,high=.8) # random color
  axs[0].plot(xx,yy,linewidth=.5,color=(rc,rc,rc))
  axs[0].plot(meenz[i],yy[np.argmin(np.abs(xx-meenz[i]))],'k*',linewidth=.2,
              markerfacecolor=(rc,rc,rc),markersize=12)

# some plotting adjustments
axs[0].set(xlim=xx[[0,-1]],ylabel='Probability',yticks=[])
axs[0].set_title(r'$\bf{B}$)  Data distributions')

## the distribution of sample means
axs[1].hist(meenz,20,facecolor=(.7,.7,.7),edgecolor='k')
axs[1].plot(xx,allYYs/np.max(allYYs)*5,'k',zorder=-10)
axs[1].set(xlim=xx[[0,-1]],xlabel='Data value',ylabel='Count',yticks=[])
axs[1].set_title(r'$\bf{C}$)  Means distribution')

# output
plt.tight_layout()
#plt.savefig('sample_distOfExGausMeans.png')
plt.show()
```

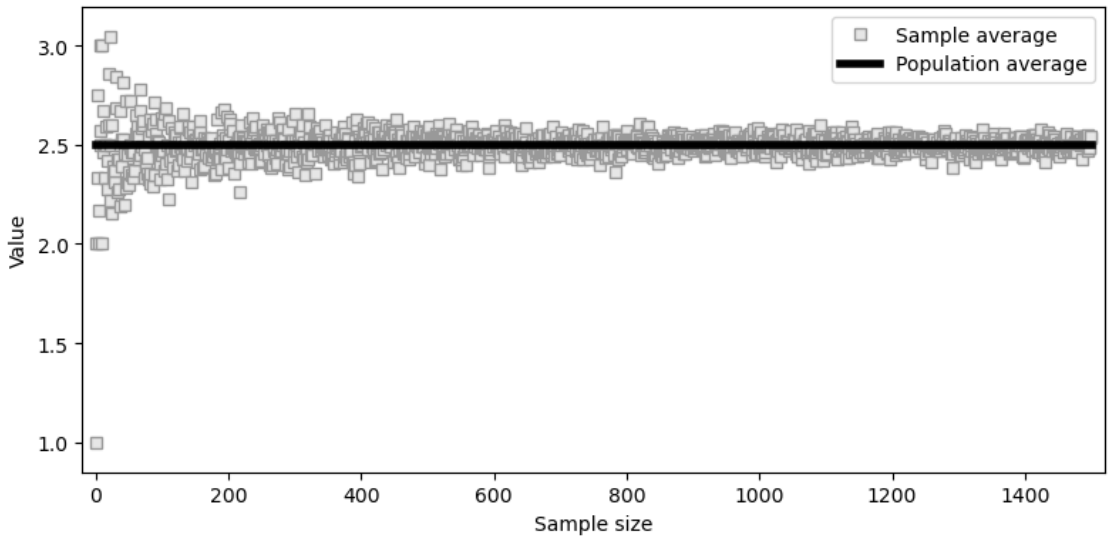


5 Figure 9.3: Law of Large Numbers (demo 1)

[5]:
```
# generate "population"
population = [ 1, 2, 3, 4 ]
for i in range(20):
    population = np.hstack((population,population))

nPop = len(population)
expval = np.mean(population)
print(f'Expected value (population mean): {expval}')
print(f'Population size: {nPop}')
```

```
Expected value (population mean): 2.5
Population size: 4194304
```

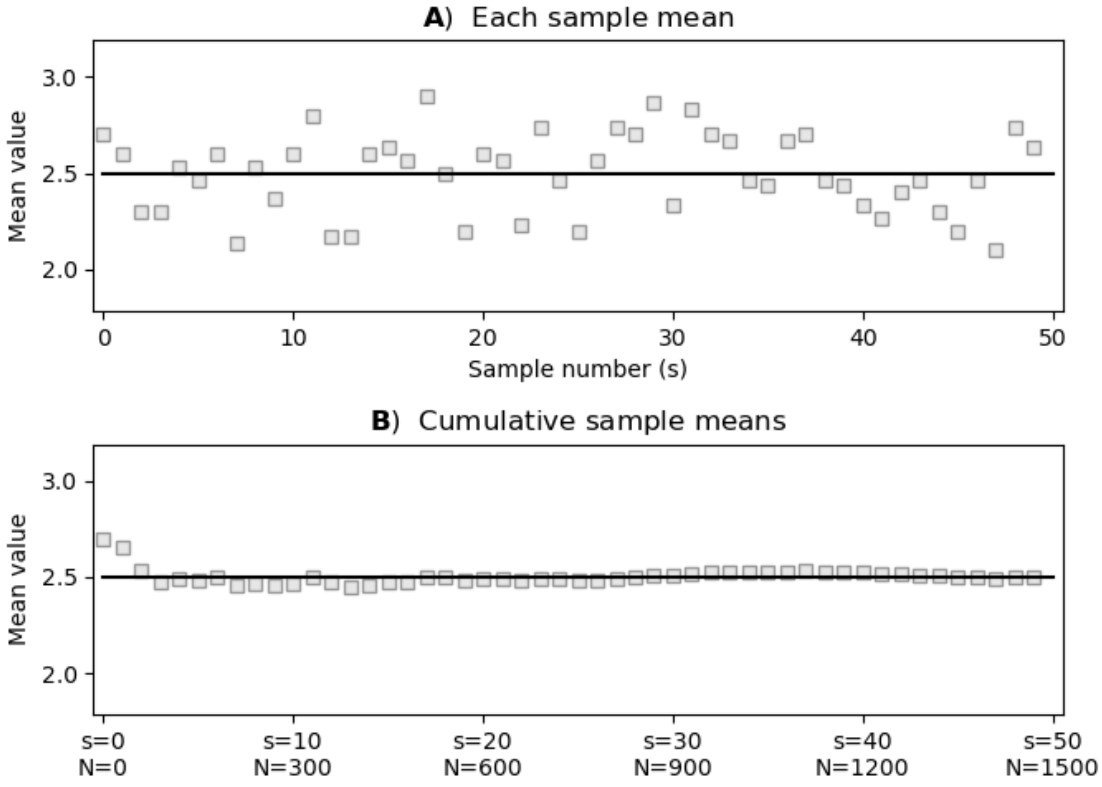
[6]:
```
## experiment: draw larger and larger samples

k = 1500  # maximum number of samples
sampleAves = np.zeros(k)

for i in range(k):
  # get a sample
  sample = np.random.choice(population,size=i+1)

  # compute and store its mean
  sampleAves[i] = np.mean( sample )

# visualize!
plt.figure(figsize=(8,4))
plt.plot(sampleAves,'s',markerfacecolor=(.9,.9,.9),color=(.6,.6,.6))
plt.plot([1,k],[expval,expval],'k',linewidth=4)
plt.xlabel('Sample size')
plt.ylabel('Value')
plt.xlim([-20,k+20])
plt.ylim([np.min(sampleAves)*.85,1.05*np.max(sampleAves)])
plt.legend(('Sample average','Population average'))

plt.tight_layout()
#plt.savefig('sample_LLNdemo1.png')
plt.show()
```



6 Figure 9.4: Law of Large Numbers (demo 2)

[7]:
```
# parameters and initializations
samplesize   = 30
numberOfExps = 50
samplemeans  = np.zeros(numberOfExps)

# run the experiment!
for expi in range(numberOfExps):
  # compute and store its mean
  samplemeans[expi] = np.mean( np.random.choice(population,size=samplesize) )

# show the results
fig,ax = plt.subplots(2,1,figsize=(7,5))

# each individual sample mean
ax[0].plot(samplemeans,'s',markerfacecolor=(.9,.9,.9),color=(.6,.6,.6))
ax[0].set_title(r'$\bf{A}$)  Each sample mean')
ax[0].set_xlabel('Sample number (s)')

# cumulative average over the samples
ax[1].plot(np.cumsum(samplemeans) / np.arange(1,numberOfExps+1),
           's',markerfacecolor=(.9,.9,.9),color=(.6,.6,.6))
ax[1].set_title(r'$\bf{B}$)  Cumulative sample means')

# multiline xtick labels
xticks = np.arange(0,51,10)
ax[1].set_xticks(xticks,labels=[f's={i}\nN={i*samplesize}' for i in xticks])
```

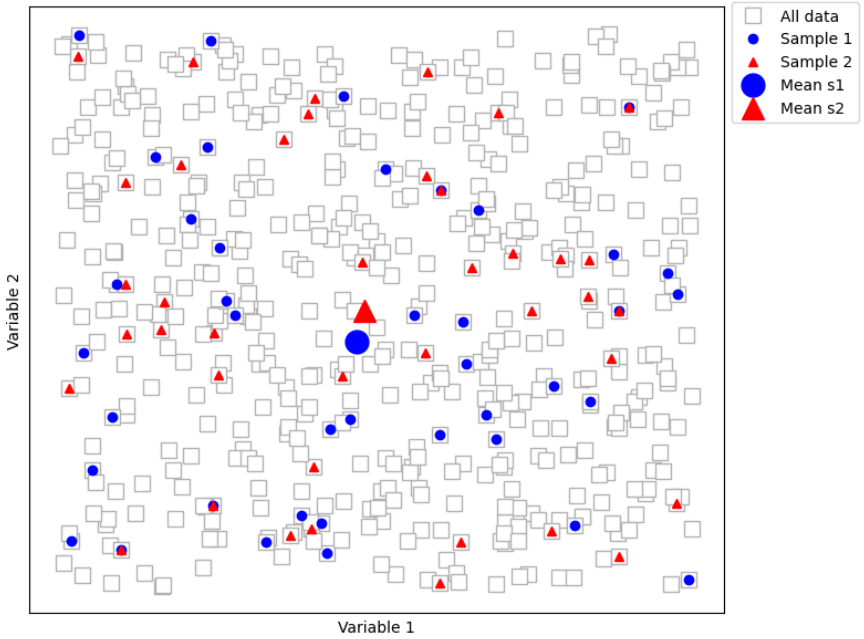
```
# common axis modifications
for a in ax:
  a.plot([0,numberOfExps],[np.mean(population),np.mean(population)],'k')
  a.set(ylabel='Mean value',xlim=[-.5,numberOfExps+.5],
        ylim=[np.min(samplemeans)*.85,1.1*np.max(samplemeans)])

totSS = np.arange(1,numberOfExps+1)*samplesize

plt.tight_layout()
#plt.savefig('sample_LLNdemo2.png')
plt.show()
```



7 Figure 9.5: Visualization of sample mean variability

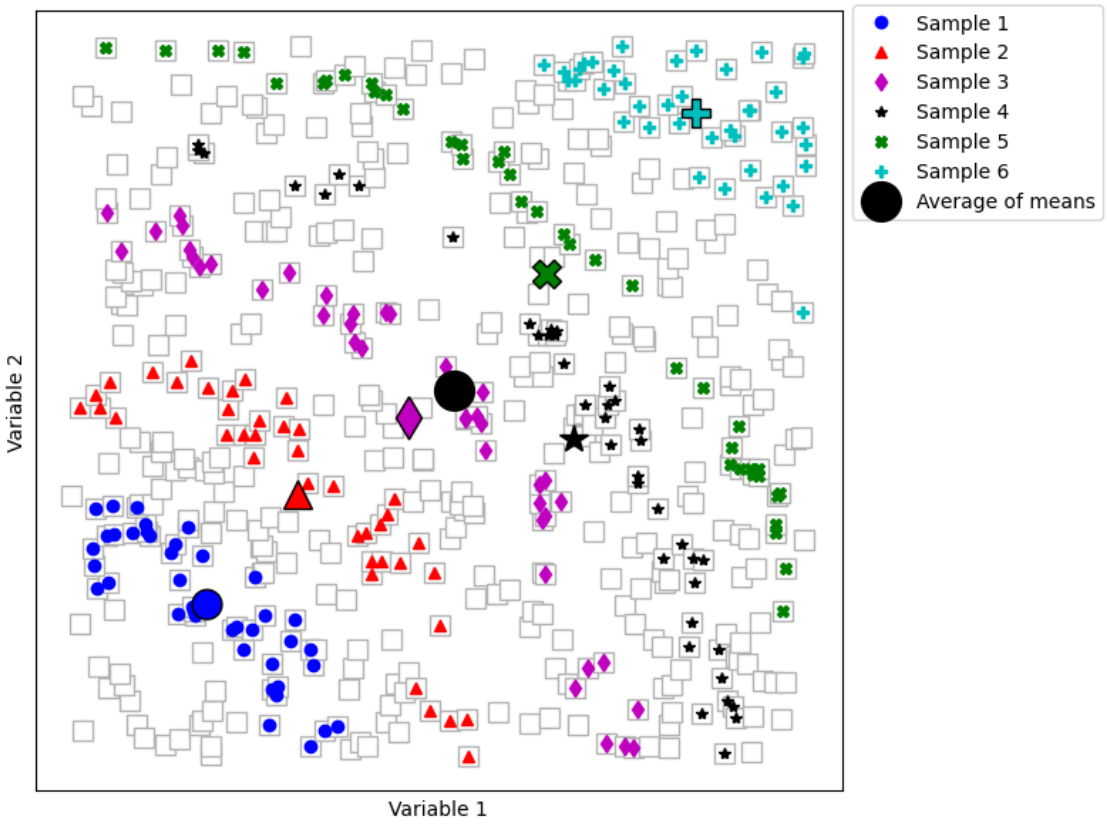
[8]:
```
N = 512
X = np.random.rand(N,2)

sample1 = X[np.random.choice(N,size=40),:]
sample2 = X[np.random.choice(N,size=40),:]
```

```
plt.figure(figsize=(8,6))

# plot all data points
plt.plot(X[:,0],X[:,1],'s',color=(.7,.7,.7),markerfacecolor='w',markersize=10)

# plot sample data
plt.plot(sample1[:,0],sample1[:,1],'bo')
plt.plot(sample2[:,0],sample2[:,1],'r^')

# plot sample means
plt.plot(np.mean(sample1[:,0]),np.mean(sample1[:,1]),'bo',markersize=15)
plt.plot(np.mean(sample2[:,0]),np.mean(sample2[:,1]),'r^',markersize=15)

plt.xticks([])
plt.yticks([])
plt.xlabel('Variable 1')
plt.ylabel('Variable 2')
plt.legend(['All data','Sample 1','Sample 2','Mean s1','Mean s2'],
           bbox_to_anchor=[1,1.02])

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



8 Figure 9.6: Sample biases can be overcome by LLN (given some assumptions)

[9]:
```
N = 512
X = np.random.rand(N,2)

# nonrandom sorting
X = X[np.argsort(np.sum(X**2,axis=1)),:]

# plot all data points
plt.figure(figsize=(8,6))
plt.plot(X[:,0],X[:,1],'s',color=(.7,.7,.7),markerfacecolor='w',markersize=10)

# nonrandom sampling to simulate a bias
sampmeans = np.zeros((6,2)) # hard-coded to 6 samples...
sampbias = np.linspace(20,N-40,6,dtype=int)
shapes = 'o^d*XP'
colors = 'brmkgc'
for si in range(6):
  # biased sample and its mean
  sample = X[sampbias[si]:sampbias[si]+40,:]
  sampmeans[si,:] = np.mean(sample,axis=0)

  # plot samples
  plt.plot(sample[:,0],sample[:,1],shapes[si],color=colors[si],
           markerfacecolor=colors[si],label=f'Sample {si+1}')

  # plot sample mean
  plt.plot(sampmeans[si,0],sampmeans[si,1],shapes[si],color='k',
           markerfacecolor=colors[si],markersize=15)

# plot the average of sample means
plt.plot(np.mean(sampmeans[:,0]),np.mean(sampmeans[:,1]),'ko',
         markerfacecolor='k',markersize=20,label='Average of means')

plt.xticks([])
plt.yticks([])
plt.xlabel('Variable 1')
plt.ylabel('Variable 2')
plt.legend(bbox_to_anchor=[1,1.02])

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



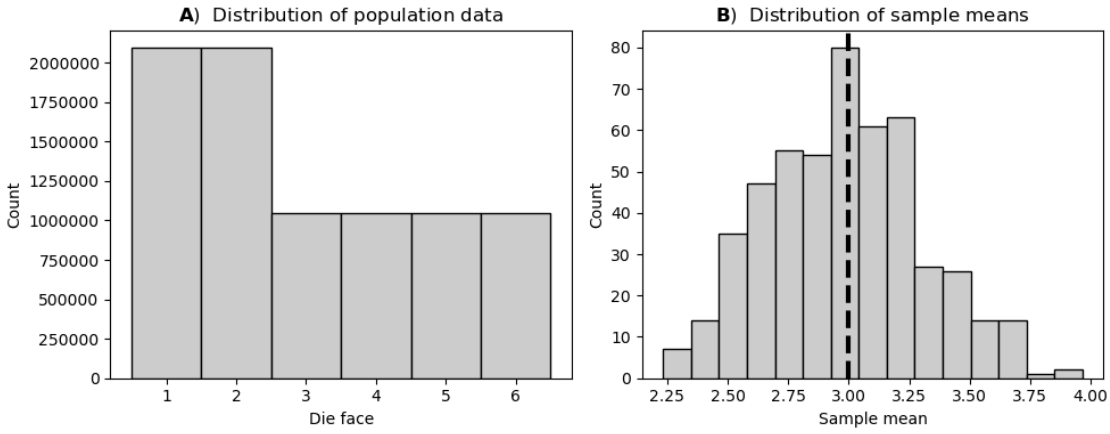
9 Figure 9.7: CLT, demo 1

[10]:
```
# generate "population" (simulating a weighted die)
population = [ 1, 1, 2, 2, 3, 4, 5, 6 ]
for i in range(20):
    population = np.hstack((population,population))

nPop = len(population)
expval = np.mean(population)
print(f'Expected value (population mean): {expval}')
print(f'Population size: {nPop}')
```

```
Expected value (population mean): 3.0
Population size: 8388608
```

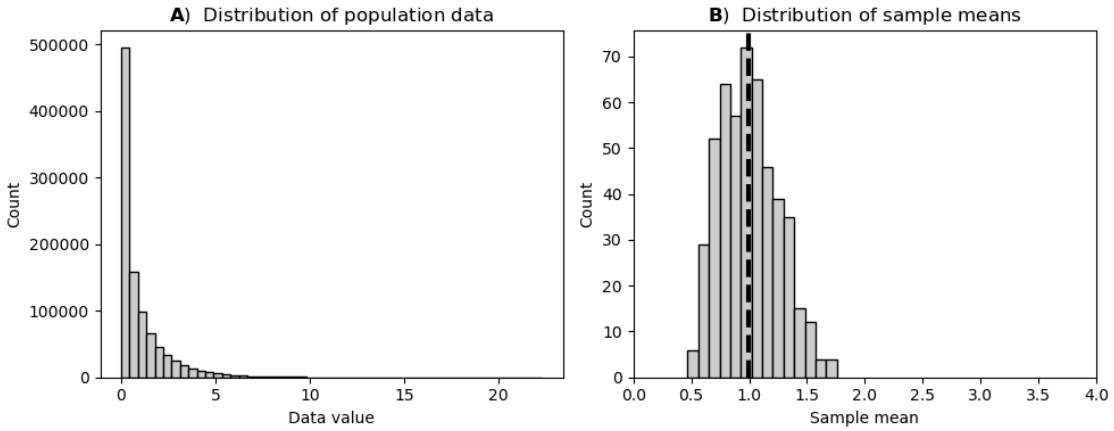
[11]:
```
# parameters and initializations
samplesize   = 30
numberOfExps = 500
samplemeans  = np.zeros(numberOfExps)
```

```
# run the experiment!
for expi in range(numberOfExps):
  # compute and store its mean
  samplemeans[expi] = np.mean( np.random.choice(population,size=samplesize) )

# show the results
fig,axs = plt.subplots(1,2,figsize=(10,4))

# histogram of the data
axs[0].hist(population,bins=np.arange(.5,7.5,step=1),color=[.8,.8,.
 ↪8],edgecolor='k')
axs[0].set(xticks=range(1,7),xlabel='Die face',ylabel='Count')
axs[0].set_title(r'$\bf{A}$)  Distribution of population data')
axs[0].ticklabel_format(style='plain')

# histogram of the sample means
axs[1].hist(samplemeans,bins='fd',color=[.8,.8,.8],edgecolor='k')
axs[1].axvline(expval,linewidth=3,color='k',linestyle='--')
axs[1].set(xlabel='Sample mean',ylabel='Count')
axs[1].set_title(r'$\bf{B}$)  Distribution of sample means')

plt.tight_layout()
#plt.savefig('sample_CLTdemo1.png')
plt.show()
```



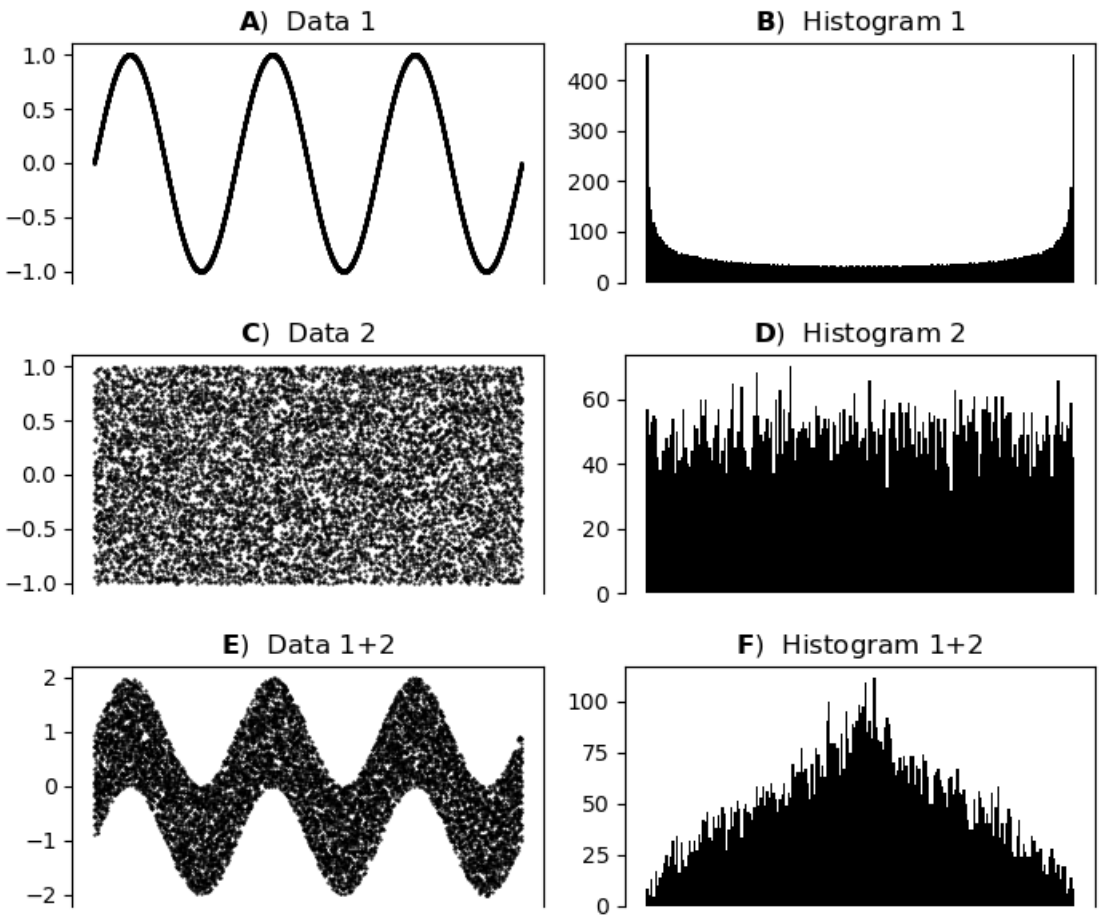
10 Figure 9.8: CLT, demo 2

[12]:
```
# new population!
Npop = 1000000
population = np.random.randn(Npop)**2

# parameters and initializations
samplesize   =  30
numberOfExps = 500
samplemeans  = np.zeros(numberOfExps)

# run the experiment!
for expi in range(numberOfExps):
  # compute and store its mean
  samplemeans[expi] = np.mean( np.random.choice(population,size=samplesize) )

# show the results
fig,axs = plt.subplots(1,2,figsize=(10,4))

# histogram of the data
axs[0].hist(population,bins=50,color=[.8,.8,.8],edgecolor='k')
axs[0].set(xlabel='Data value',ylabel='Count')
axs[0].set_title(r'$\bf{A}$)  Distribution of population data')
axs[0].ticklabel_format(style='plain')

# histogram of the sample means
axs[1].hist(samplemeans,bins='fd',color=[.8,.8,.8],edgecolor='k')
axs[1].set(xlabel='Sample mean',ylabel='Count',xlim=[0,4])
axs[1].axvline(np.mean(population),linewidth=3,color='k',linestyle='--')
axs[1].set_title(r'$\bf{B}$)  Distribution of sample means')

plt.tight_layout()
#plt.savefig('sample_CLTdemo2.png')
plt.show()
```



11 Figure 9.9: CLT, demo 3

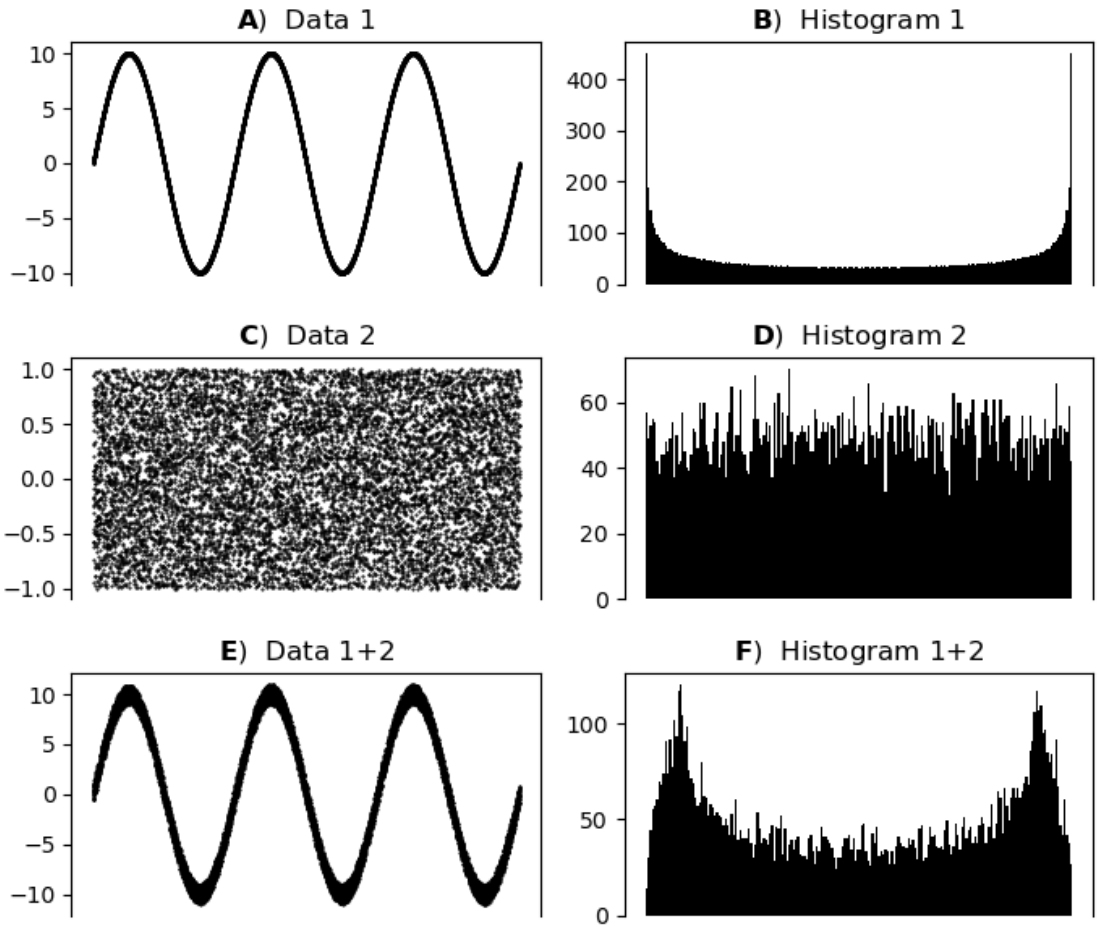
[13]:
```
# create two data variables
x = np.linspace(0,6*np.pi,10000)
s = np.sin(x)
u = 2*np.random.rand(len(x))-1

# combine them into a list for convenience
datasets = [ s,u,s+u ]
axislets = iter('ABCDEF') # axis labels

# plot!
_,axs = plt.subplots(3,2,figsize=(7,6))

for i in range(3):
  # axis variable label
  dlab = str(i+1) if i<2 else '1+2'

  # plot the data
  axs[i,0].plot(x,datasets[i],'k.',markersize=1)
  axs[i,0].set_title(r'$\bf{%s}$)  Data %s' %(next(axislets),dlab))

  # plot the histogram
  axs[i,1].hist(datasets[i],bins=200,color='k',edgecolor=None)
  axs[i,1].set_title(r'$\bf{%s}$)  Histogram %s' %(next(axislets),dlab))

# adjust the axis properties
for a in axs.flatten():
  a.xaxis.set_visible(False)
  a.spines['bottom'].set_visible(False)

plt.tight_layout()
#plt.savefig('sample_CLTdemo3a.png')
plt.show()
```



12 Figure 9.10: CLT requires comparable scaling

[14]:
```
# only difference from the previous figure is the amplitude-scaling!
s = 10*np.sin(x)

# combine them into a list for convenience
datasets = [ s,u,s+u ]
axislets = iter('ABCDEF') # axis labels

# plot!
_,axs = plt.subplots(3,2,figsize=(7,6))

for i in range(3):
  # axis variable label
  dlab = str(i+1) if i<2 else '1+2'

  # plot the data
```

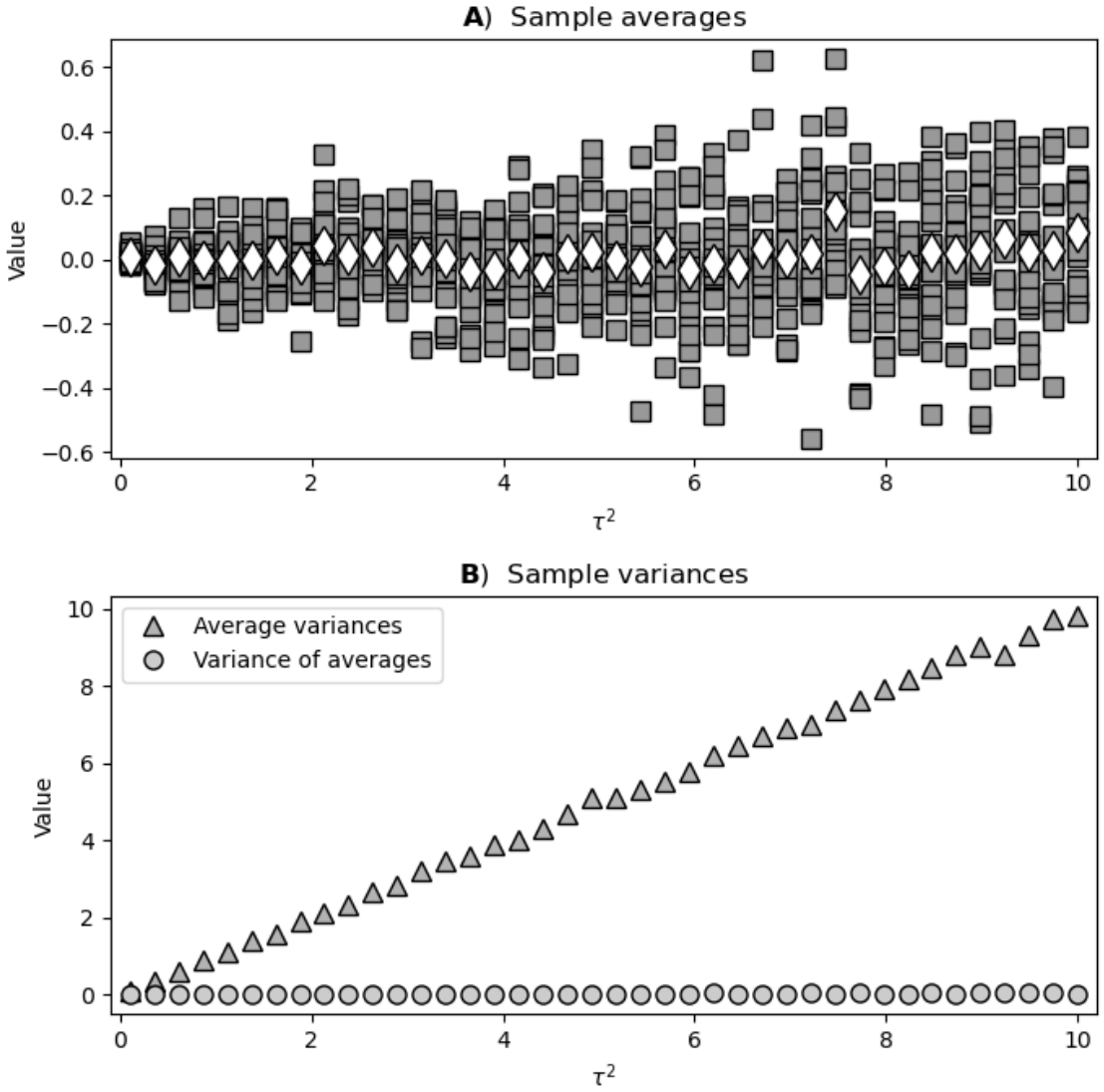
```
  axs[i,0].plot(x,datasets[i],'k.',markersize=1)
  axs[i,0].set_title(r'$\bf{%s}$)  Data %s' %(next(axislets),dlab))

  # plot the histogram
  axs[i,1].hist(datasets[i],bins=200,color='k',edgecolor=None)
  axs[i,1].set_title(r'$\bf{%s}$)  Histogram %s' %(next(axislets),dlab))


# adjust the axis properties
for a in axs.flatten():
  a.xaxis.set_visible(False)
  a.spines['bottom'].set_visible(False)

plt.tight_layout()
#plt.savefig('sample_CLTdemo3b.png')
plt.show()
```



13 Exercise 1

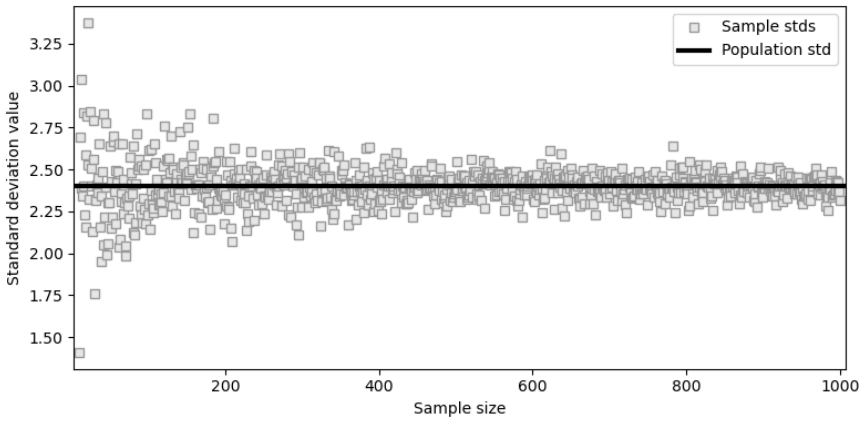
[15]:
```
# variance levels (tau^2)
tau2levels = np.linspace(.1,10,40)

# simulation parameters
samplesize = 200
numsamples =  20

# initialize results matrix
results = np.zeros((numsamples,len(tau2levels),2))

### run the experiment!
# loop over tau levels
for ni,tau2 in enumerate(tau2levels):
  # repeat for multiple samples
  for sampi in range(numsamples):

    # generate sample data with tau modulation
    data = np.random.normal(0,np.sqrt(tau2),size=samplesize)

    # store sample mean and variance
    results[sampi,ni,0] = np.mean(data)
    results[sampi,ni,1] = np.var(data,ddof=1)

### plotting
_,axs = plt.subplots(2,1,figsize=(7,7))

# plot the average of the sample means
axs[0].plot(np.tile(tau2levels,(20,1)),results[:,:,0],'ks',
            markerfacecolor=(.6,.6,.6),markersize=8)
axs[0].plot(tau2levels,np.mean(results[:,:,0],axis=0),'kd',
            markerfacecolor='w',markersize=12)
axs[0].set_title(r'$\bf{A}$)  Sample averages')

# plot the average within-sample variances
axs[1].plot(tau2levels,np.mean(results[:,:,1],axis=0),'k^',
            markerfacecolor=(.7,.7,.7),markersize=8,label='Average variances')

# plot the average across-sample variance of the sample means
axs[1].plot(tau2levels,np.var(results[:,:,0],axis=0,ddof=1),'ko',
            markerfacecolor=(.8,.8,.8),markersize=8,label='Variance of averages')
axs[1].set_title(r'$\bf{B}$)  Sample variances')
axs[1].legend()

for a in axs:
  a.set(xlabel=r'$\tau^2$',ylabel='Value',
```

```
        xlim=[tau2levels[0]-.2,tau2levels[-1]+.2])

plt.tight_layout()
#plt.savefig('sample_ex1.png')
plt.show()
```



14 Exercise 2

[16]:
```
# the sample sizes
samplesizes = np.arange(10,1001)

# generate population data with known std
pop_std     = 2.4
populationN = 1000000
population  = np.random.randn(populationN)
population  = population / np.std(population,ddof=1) # force std=1
population  = population * pop_std # force std

# initialize results matrix
samplestds = np.zeros(len(samplesizes))

# run the experiment!
for sampi in range(len(samplesizes)):

  # pick a random sample
  sample = np.random.choice(population,size=samplesizes[sampi])
  samplestds[sampi] = np.std(sample,ddof=1)

# show the results!
plt.figure(figsize=(8,4))
plt.plot(samplesizes,samplestds,'s',markerfacecolor=(.9,.9,.9),color=(.6,.6,.6))
plt.axhline(pop_std,color='k',linewidth=3)
plt.xlabel('Sample size')
plt.ylabel('Standard deviation value')
plt.xlim([samplesizes[0]-7,samplesizes[-1]+7])
plt.legend(('Sample stds','Population std'))

plt.tight_layout()
#plt.savefig('sample_ex2.png')
plt.show()
```



[17]:
```
# Note about the data-generation method:
# It is not sufficient to use np.random.normal(0,2.4,N), because that does
# not guarantee a *population* standard deviation of 2.4. Instead, it is
# necessary to force the std by first scaling to std=1 and then multiplying.

# Here's a demonstration:
print(np.std(np.random.normal(0,pop_std,populationN),ddof=1))
print(np.std(population,ddof=1))
```

```
2.3987793031234905
2.400000000000001
```

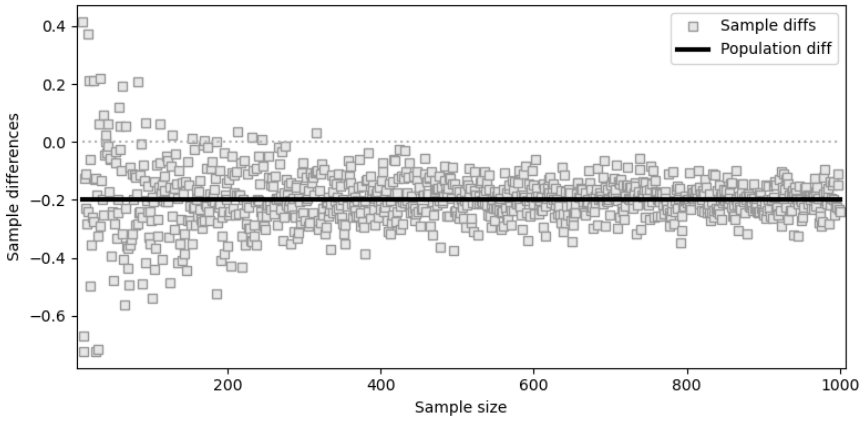
15 Exercise 3

[18]:
```
# parameters
popMean1 = 3
popMean2 = 3.2

# generate the populations
population1 = np.random.randn(populationN)
population1 = population1 - np.mean(population1) + popMean1

population2 = np.random.randn(populationN)
population2 = population2 - np.mean(population2) + popMean2

# one sample
s1 = np.mean( np.random.choice(population1,size=30) )
s2 = np.mean( np.random.choice(population2,size=30) )

print(f'Population difference: {popMean1-popMean2:.3f}')
print(f'Sample difference:     {s1-s2:.3f}')
```

```
Population difference: -0.200
Sample difference:     -0.595
```

[19]:
```
# initialize results matrix
samplediffs = np.zeros(len(samplesizes))

# run the experiment!
for sampi in range(len(samplesizes)):

  # pick a random sample
  s1 = np.random.choice(population1,size=samplesizes[sampi])
  s2 = np.random.choice(population2,size=samplesizes[sampi])
  samplediffs[sampi] = np.mean(s1) - np.mean(s2)
```

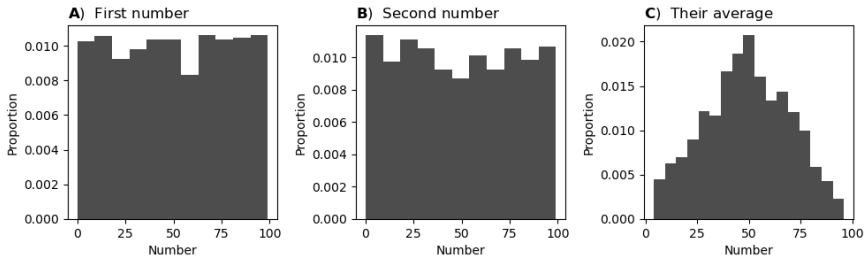
```
# show the results!
plt.figure(figsize=(8,4))
plt.plot(samplesizes,samplediffs,'s',markerfacecolor=(.9,.9,.9),color=(.6,.6,.6))
plt.
 ↪plot(samplesizes[[0,-1]],[popMean1-popMean2,popMean1-popMean2],'k',linewidth=3)
plt.plot(samplesizes[[0,-1]],[0,0],color=(.7,.7,.7),linestyle=':')

plt.xlabel('Sample size')
plt.ylabel('Sample differences')
plt.xlim([samplesizes[0]-7,samplesizes[-1]+7])
plt.legend(('Sample diffs','Population diff'))

plt.tight_layout()
#plt.savefig('sample_ex3.png')
plt.show()
```



16 Exercise 4

[20]:
```
N = 1200
numbers = np.zeros((N,3))

for i in range(N):
  nums = np.random.choice(range(100),2)
  numbers[i,0] = nums[0]
  numbers[i,1] = nums[1]
  numbers[i,2] = np.mean(nums)

# show the histograms
_,axs = plt.subplots(1,3,figsize=(10,np.pi))
for i in range(3):
```

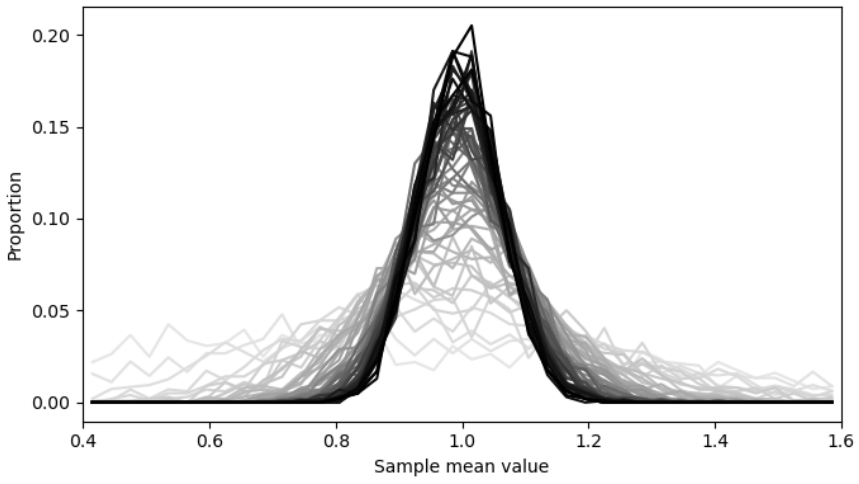
```
  axs[i].hist(numbers[:,i],bins='fd',color=(.3,.3,.3),density=True)
  axs[i].set(xlabel='Number',ylabel='Proportion')

axs[0].set_title(r'$\bf{A}$)  First number',loc='left')
axs[1].set_title(r'$\bf{B}$)  Second number',loc='left')
axs[2].set_title(r'$\bf{C}$)  Their average',loc='left')

plt.tight_layout()
#plt.savefig('sample_ex4.png')
plt.show()
```



17 Exercise 6

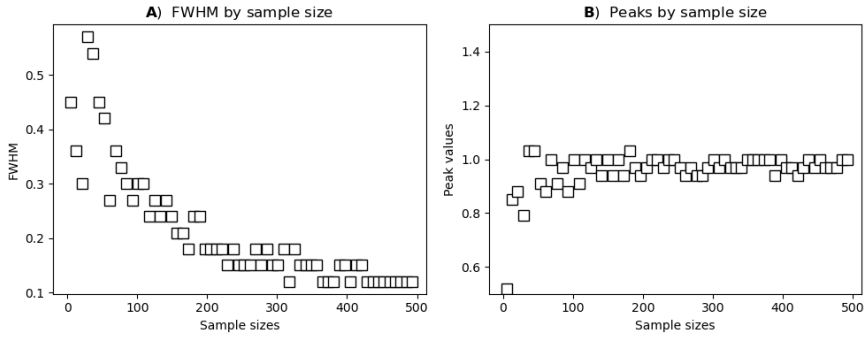
[21]:
```
# a population of random numbers
Npop = 1000000
population = np.random.randn(Npop)**2

# parameters and initializations
samplesizes   = np.arange(5,500,8)
numberOfsamps = 1000
samplemeans   = np.zeros(numberOfsamps)
fwhms         = np.zeros(len(samplesizes))
peakvals      = np.zeros(len(samplesizes))

# line colors
c = np.linspace((.9,.9,.9),(0,0,0),len(samplesizes))

## run the experiment!
plt.figure(figsize=(7,4))
for Ns in range(len(samplesizes)):
  # compute the means of lots of samples
  for expi in range(numberOfsamps):
    samplemeans[expi] = np.mean( np.random.
↪choice(population,size=samplesizes[Ns]) )

  # make a histogram of those means
```

```
  yy,xx = np.histogram(samplemeans,np.linspace(.4,1.6,41))
  yy = yy/np.sum(yy)

  ### compute FWHM
  # step 1: normalize
  yn = yy/np.max(yy)

  # step 2: find peak index
  idx = np.argmax(yn)

  # step 3: compute FWHM
  fwhms[Ns] = xx[idx-1+np.argmin(np.abs(yn[idx:]-.5))] - xx[np.argmin(np.abs(yn[:
↪idx]-.5))]

  # also store mean value
  peakvals[Ns] = xx[idx]

  # plot
  plt.plot((xx[:-1]+xx[1:])/2,yy,color=c[Ns])

plt.xlim([.4,1.6])
plt.xlabel('Sample mean value')
plt.ylabel('Proportion')
plt.tight_layout()
#plt.savefig('sample_ex6a.png')
plt.show()
```



[22]:
```
_,axs = plt.subplots(1,2,figsize=(10,4))

axs[0].plot(samplesizes,fwhms,'ks',markerfacecolor='w',markersize=8)
```

```
axs[0].set(xlabel='Sample sizes',ylabel='FWHM',xlim=[-20,samplesizes[-1]+20])
axs[0].set_title(r'$\bf{A}$)  FWHM by sample size')

axs[1].plot(samplesizes,peakvals,'ks',markerfacecolor='w',markersize=8)
axs[1].set(xlabel='Sample sizes',ylabel='Peak values',ylim=[.5,1.
 ↪5],xlim=[-20,samplesizes[-1]+20])
axs[1].set_title(r'$\bf{B}$)  Peaks by sample size')

plt.tight_layout()
#plt.savefig('sample_ex6b.png')
plt.show()
```



18 Exercise 7

[23]:
```
# a population of random numbers
Npop = 1000000
population = np.random.randn(Npop)**2

# experiment parameters
samplesizes = np.logspace(np.log10(10),np.log10(Npop/10),25,dtype=int)
numExps = 50

# theoretical standard error based on population standard deviation
theory = np.std(population) / np.sqrt(samplesizes)

# initialize the empirical estimates
standerror = np.zeros((numExps,len(samplesizes)))
samplemeans = np.zeros((numExps,len(samplesizes)))

# Run the experiment!
for expi in range(numExps):
  for idx,ssize in enumerate(samplesizes):
    # generate a random sample
```

```
    rsample = np.random.choice(population,size=ssize)

    # compute its standard error (estimate) and the sample mean
    standerror[expi,idx] = np.std(rsample,ddof=1) / np.sqrt(ssize)
    samplemeans[expi,idx] = np.mean(rsample)

## plotting
_,axs = plt.subplots(1,2,figsize=(9,4))
axs[0].plot(samplesizes,theory,'ks-',markersize=10,markerfacecolor=(.7,.7,.
 ↪7),label='Analytical SEM')
axs[0].plot(samplesizes,np.mean(standerror,axis=0),'p-',color=(.3,.3,.3),
            markersize=10,markerfacecolor=(0,0,0),label='Empirical SEM')
axs[0].plot(samplesizes,np.std(samplemeans,axis=0,ddof=1),'o-',color=(.7,.7,.7),
            markersize=10,markerfacecolor=(.9,.9,.9),label='Sample means STD')
axs[0].set(xlabel='Sample size',ylabel='Sample means variabilities')
axs[0].legend()
axs[0].set_title(r'$\bf{A}$)  Estimates by sample size')

axs[1].plot(theory,np.mean(standerror,axis=0),'ko',markerfacecolor=(.8,.8,.
 ↪8),markersize=12,alpha=.5,label='Empirical SEM')
axs[1].plot(np.std(samplemeans,axis=0,ddof=1),np.
 ↪mean(standerror,axis=0),'ks',markerfacecolor=(.3,.3,.3),markersize=12,alpha=.
 ↪5,label='Sample mean std')
axs[1].set(xlabel='Analytical SEM',ylabel=r'Emp. SEM $or$ means STD')
axs[1].legend()
axs[1].set_title(r'$\bf{B}$)  Consistency of estimates')

plt.tight_layout()
#plt.savefig('sample_ex7.png')
plt.show()
```