stats_ch10_hypotheses

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 10 (hypothesis testing)

---

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.

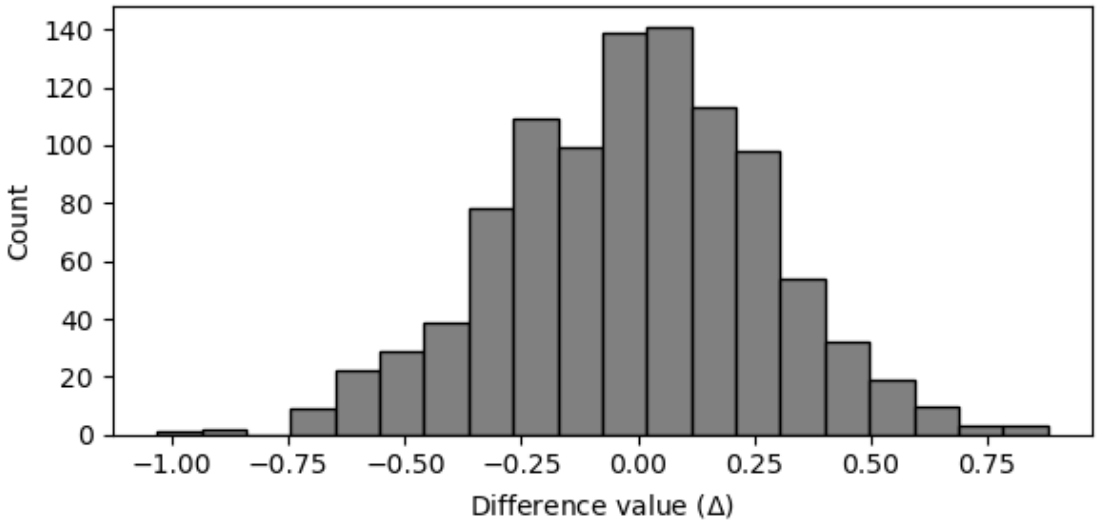
This code was written in google-colab. The notebook may require some modifications if you use a different IDE.

[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 10.2: Empirical distribution under H0

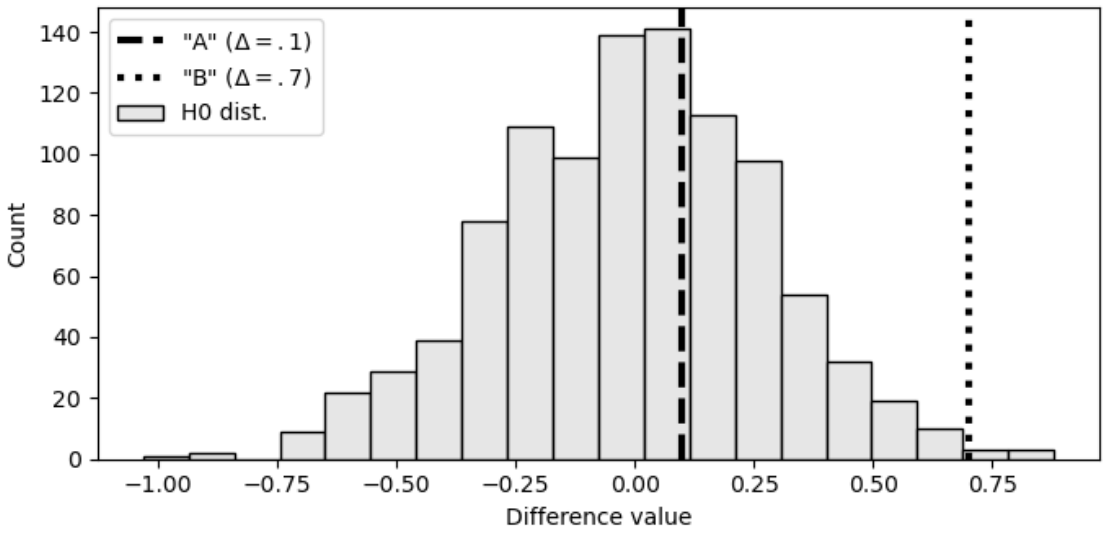
[3]:
```
N = 100 # per group per sample
numExps = 1000

meandiff = np.zeros(numExps)

# run the experiment
for i in range(numExps):
```

```
  pre = stats.truncnorm(a=-5,b=10,loc=6,scale=2).rvs(N)
  pst = stats.truncnorm(a=-5,b=10,loc=6,scale=2).rvs(N)

  meandiff[i] = np.mean(pst) - np.mean(pre)

plt.figure(figsize=(6,3))
plt.hist(meandiff,bins=20,edgecolor='k',color='gray')
plt.xlabel(r'Difference value ($\Delta$)')
plt.ylabel('Count')

plt.tight_layout()
#plt.savefig('hyp_empH0.png')
plt.show()
```



4 Figure 10.3: Distribution assuming H0 is true

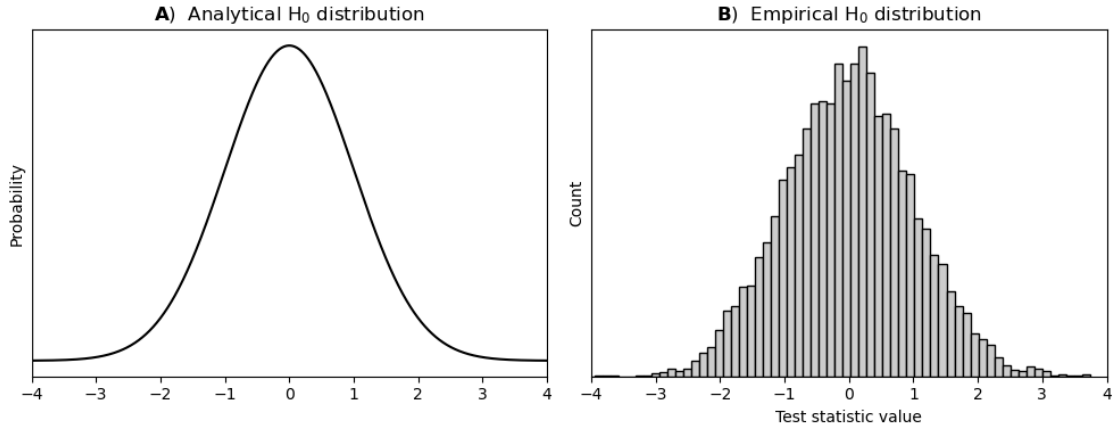
[5]:
```
plt.figure(figsize=(7,3.5))

plt.hist(meandiff,bins=20,edgecolor='k',color=(.9,.9,.9))
plt.axvline(.1,color='k',linestyle='--',linewidth=3)
plt.axvline(.7,color='k',linestyle=':',linewidth=3)
plt.xlabel('Difference value')
plt.ylabel('Count')
plt.legend([r'"A" ($\Delta=.1$)',r'"B" ($\Delta=.7$)','H0 dist.'])

plt.tight_layout()
#plt.savefig('hyp_empH0_withAs.png')
plt.show()
```



5 Figure 10.4: Analytical vs. empirical H0 distribution

[7]:
```
empirical = np.random.normal(loc=0,scale=1,size=10000)
x = np.linspace(-4,4,1001)
analytical = stats.norm.pdf(x) * np.diff(x[:2])

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

axs[0].plot(x,analytical,'k')
axs[0].set_title(r'$\bf{A}$)  Analytical H$_0$ distribution')
axs[0].set(xlim=[-4,4],yticks=[],ylabel='Probability')

axs[1].hist(empirical,bins='fd',color=(.8,.8,.8),edgecolor='k')
axs[1].set_title(r'$\bf{B}$)  Empirical H$_0$ distribution')
axs[1].set(xlim=[-4,4],yticks=[],ylabel='Count',xlabel='Test statistic value')
```

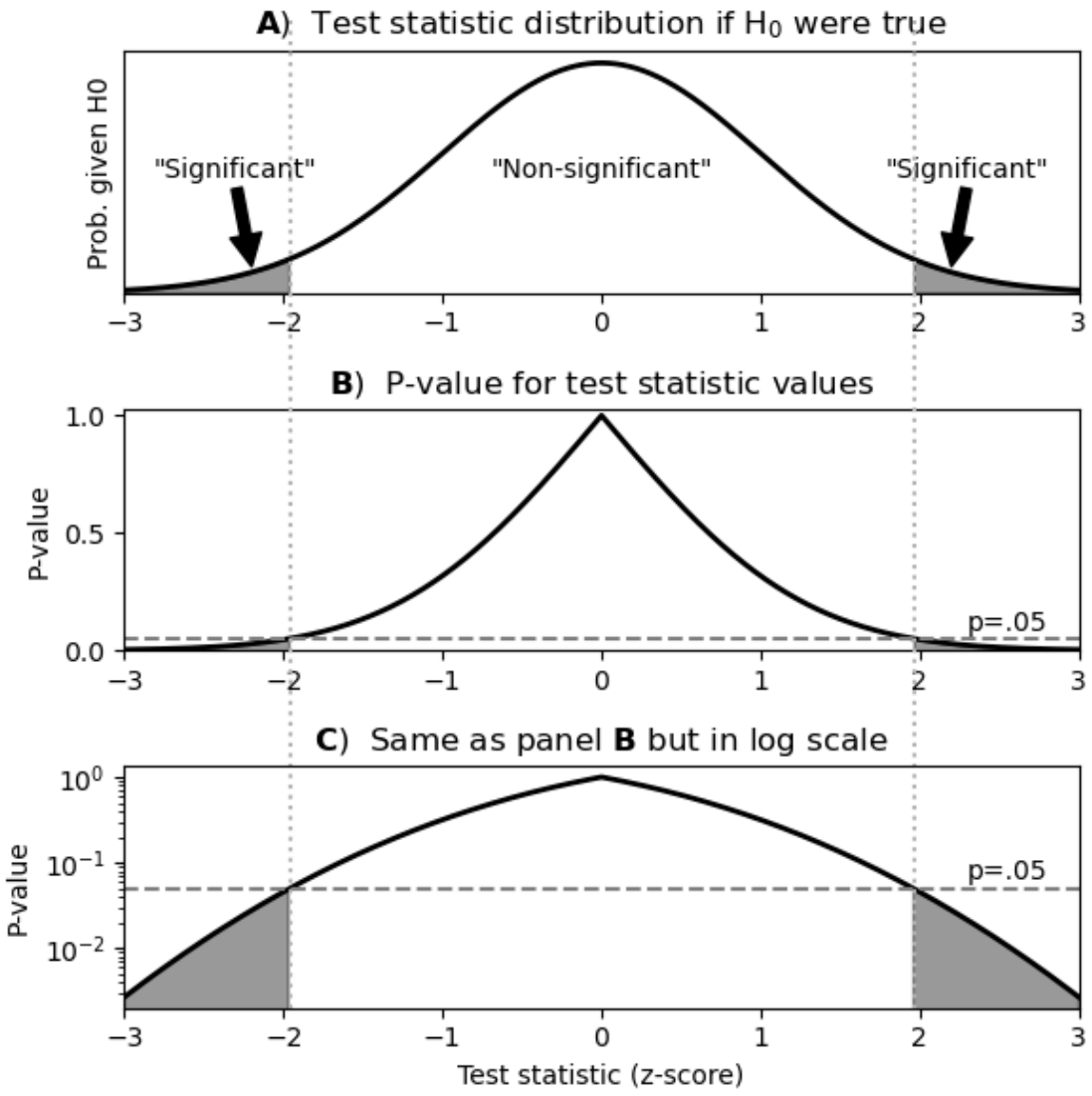
```
plt.tight_layout()
#plt.savefig('hyp_empVanalyH0.png')
plt.show()
```



6 Figure 10.5: p-values and thresholds

[9]:
```
# create a Gaussian probability curve
zvals = np.linspace(-3,3,1001)
zpdf  = stats.norm.pdf(zvals)

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

# plot the probability function and the vertical lines
axs[0].plot(zvals,zpdf,'k',linewidth=2)
axs[0].set(xlim=zvals[[0,-1]],ylim=[0,.42],yticks=[],ylabel='Prob. given H0')

# two-tailed p-values
pvalsL = stats.norm.cdf(zvals[:np.argmin(zvals**2)])
pvalsR = 1-stats.norm.cdf(zvals[np.argmin(zvals**2):])
pvals2 = 2*np.concatenate((pvalsL,pvalsR),axis=0) # doubled for a two-tailed test

# plot the probability function and the vertical lines
for i in range(1,3):
  axs[i].plot(zvals,pvals2,'k',linewidth=2)
  axs[i].set(xlim=zvals[[0,-1]],ylabel='P-value')
  axs[i].axhline(.05,color=(.5,.5,.5),linestyle='--')

# draw patches for significant regions
zidx = np.arange(np.argmin((zvals-stats.norm.ppf(.025))**2))
axs[0].fill_between(zvals[zidx],zpdf[zidx],color='k',alpha=.4)
```

```
axs[1].fill_between(zvals[zidx],pvals2[zidx],color='k',alpha=.4)
axs[2].fill_between(zvals[zidx],pvals2[zidx],color='k',alpha=.4)

zidx = np.arange(np.argmin((zvals-stats.norm.ppf(.975))**2),len(zvals))
axs[0].fill_between(zvals[zidx],zpdf[zidx],color='k',alpha=.4)
axs[1].fill_between(zvals[zidx],pvals2[zidx],color='k',alpha=.4)
axs[2].fill_between(zvals[zidx],pvals2[zidx],color='k',alpha=.4)

axs[2].axvline(x=zvals[np.argmin((zvals-stats.norm.ppf(.025))**2)],ymin=0,ymax=4.
 ↪1,c=(.7,.7,.7),linestyle=':',clip_on=False)
axs[2].axvline(x=zvals[np.argmin((zvals-stats.norm.ppf(.975))**2)],ymin=0,ymax=4.
 ↪1,c=(.7,.7,.7),linestyle=':',clip_on=False)

# indicators
axs[0].annotate('"Significant"',xy=(-2.2,stats.norm.pdf(-2.2)+.01),xytext=(-2.3,.
 ↪2),ha='center',arrowprops={'color':'k'})
axs[0].annotate('"Significant"',xy=( 2.2,stats.norm.pdf( 2.2)+.01),xytext=( 2.3,.
 ↪2),ha='center',arrowprops={'color':'k'})
axs[0].text(0,.2,'"Non-significant"',ha='center')
axs[1].text(2.3,.08,'p=.05')
axs[2].text(2.3,.065,'p=.05')

# panel titles
axs[0].set_title(r'$\bf{A}$)  Test statistic distribution if H$_0$ were true')
axs[1].set_title(r'$\bf{B}$)  P-value for test statistic values')
axs[2].set_title(r'$\bf{C}$)  Same as panel $\bf{B}$ but in log scale')

axs[2].set(yscale='log',xlabel='Test statistic (z-score)')
axs[1].set_ylim([0,1.03])

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

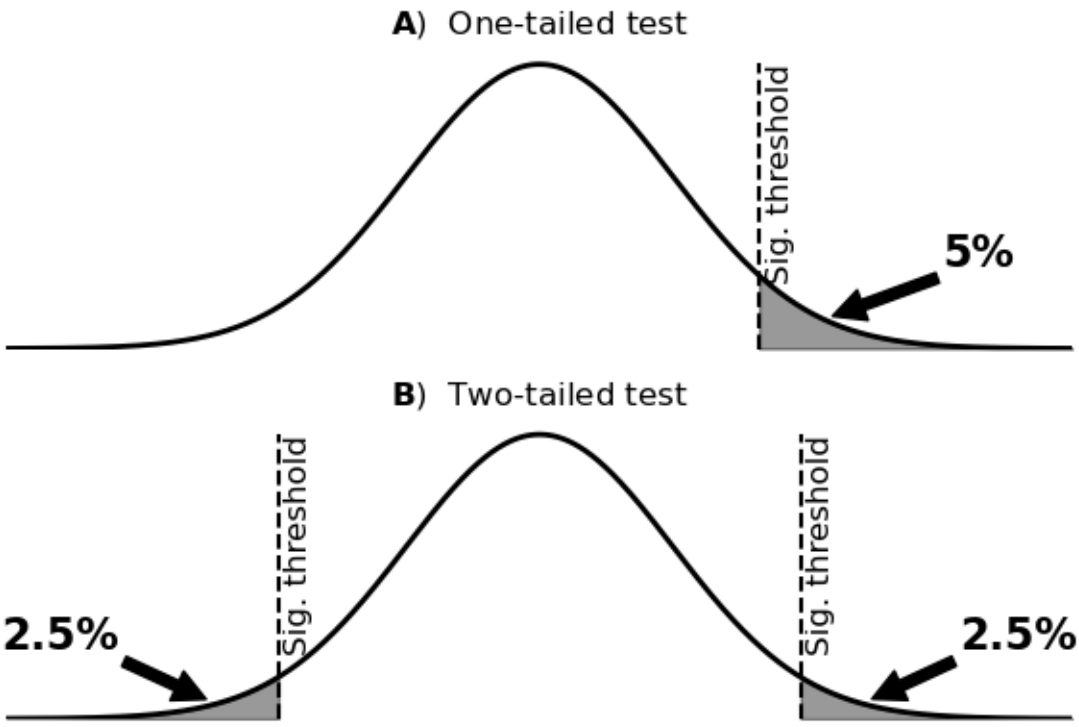


7 Figure 10.6: H0 distribution with critical value

[11]:
```
# create a Gaussian probability curve
x = np.linspace(-4,4,1001)
gpdf = stats.norm.pdf(x)

# create the figure and axis objects
_,axs = plt.subplots(2,1,figsize=(6,4))

# the find the indices of the 95% of the distribution
ubndi = np.argmin(np.abs(x-stats.norm.ppf(.95)))
```

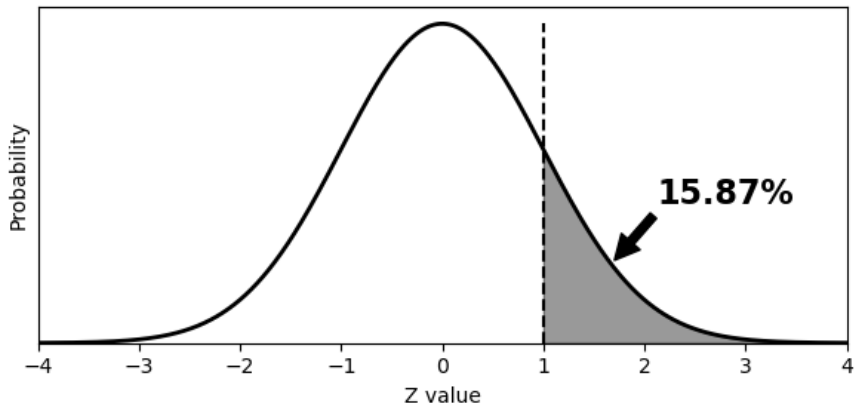
```
# plot the probability function and the vertical lines
axs[0].plot(x,gpdf,'k',linewidth=2)
axs[0].set(xlim=x[[0,-1]],ylim=[0,.42],xticks=[],yticks=[],
       xlabel='Test statistic',ylabel='Probability')

# create patches for the significant area
axs[0].fill_between(x[ubndi:],gpdf[ubndi:],color='k',alpha=.4)

# annotations
tailx = np.argmin(np.abs(x-2.2))
axs[0].annotate('5%',xy=(x[tailx],gpdf[tailx]+.01),
            xytext=(x[tailx]+1.1,gpdf[tailx]+.08),ha='center',
            arrowprops={'color':'k'},weight='bold',size=16)

# significance threshold line
axs[0].plot([x[ubndi],x[ubndi]],[0,.4],'k--')
axs[0].annotate('Sig. threshold',xy=[x[ubndi]+.05,.
 ↪4],va='top',rotation=90,size=12)

# the find the indices of the 2.5% and 97.5%
lbndi = np.argmin(np.abs(x-stats.norm.ppf(.025)))
ubndi = np.argmin(np.abs(x-stats.norm.ppf(1-.025)))

# plot the probability function and the vertical lines
axs[1].plot(x,gpdf,'k',linewidth=2)
axs[1].set(xlim=x[[0,-1]],ylim=[0,.42],xticks=[],yticks=[],
       xlabel='Test statistic',ylabel='Probability')

# now create patches for the significant area
axs[1].fill_between(x[:lbndi+1],gpdf[:lbndi+1],color='k',alpha=.4)

# significance threshold line
axs[1].plot([x[lbndi],x[lbndi]],[0,.4],'k--')
axs[1].annotate('Sig. threshold',xy=[x[lbndi]+.05,.
 ↪4],va='top',rotation=90,size=12)

# repeat for the right lobe
axs[1].fill_between(x[ubndi:],gpdf[ubndi:],color='k',alpha=.4)

# annotations
tailx = np.argmin(np.abs(x--2.5))
axs[1].annotate('2.5%',xy=(x[tailx],gpdf[tailx]+.01),
            xytext=(x[tailx]-1.1,gpdf[tailx]+.08),ha='center',
            arrowprops={'color':'k'},weight='bold',size=16)
tailx = np.argmin(np.abs(x-2.5))
axs[1].annotate('2.5%',xy=(x[tailx],gpdf[tailx]+.01),
            xytext=(x[tailx]+1.1,gpdf[tailx]+.08),ha='center',
```

```
            arrowprops={'color':'k'},weight='bold',size=16)

# significance threshold line
axs[1].plot([x[ubndi],x[ubndi]],[0,.4],'k--')
axs[1].annotate('Sig. threshold',xy=[x[ubndi]+.05,.
 ↪4],va='top',rotation=90,size=12)

# a few other niceties
axs[0].axis('off')
axs[1].axis('off')
axs[0].set_title(r'$\bf{A}$)  One-tailed test')
axs[1].set_title(r'$\bf{B}$)  Two-tailed test')

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



8 Figure 10.7: Area of z>1

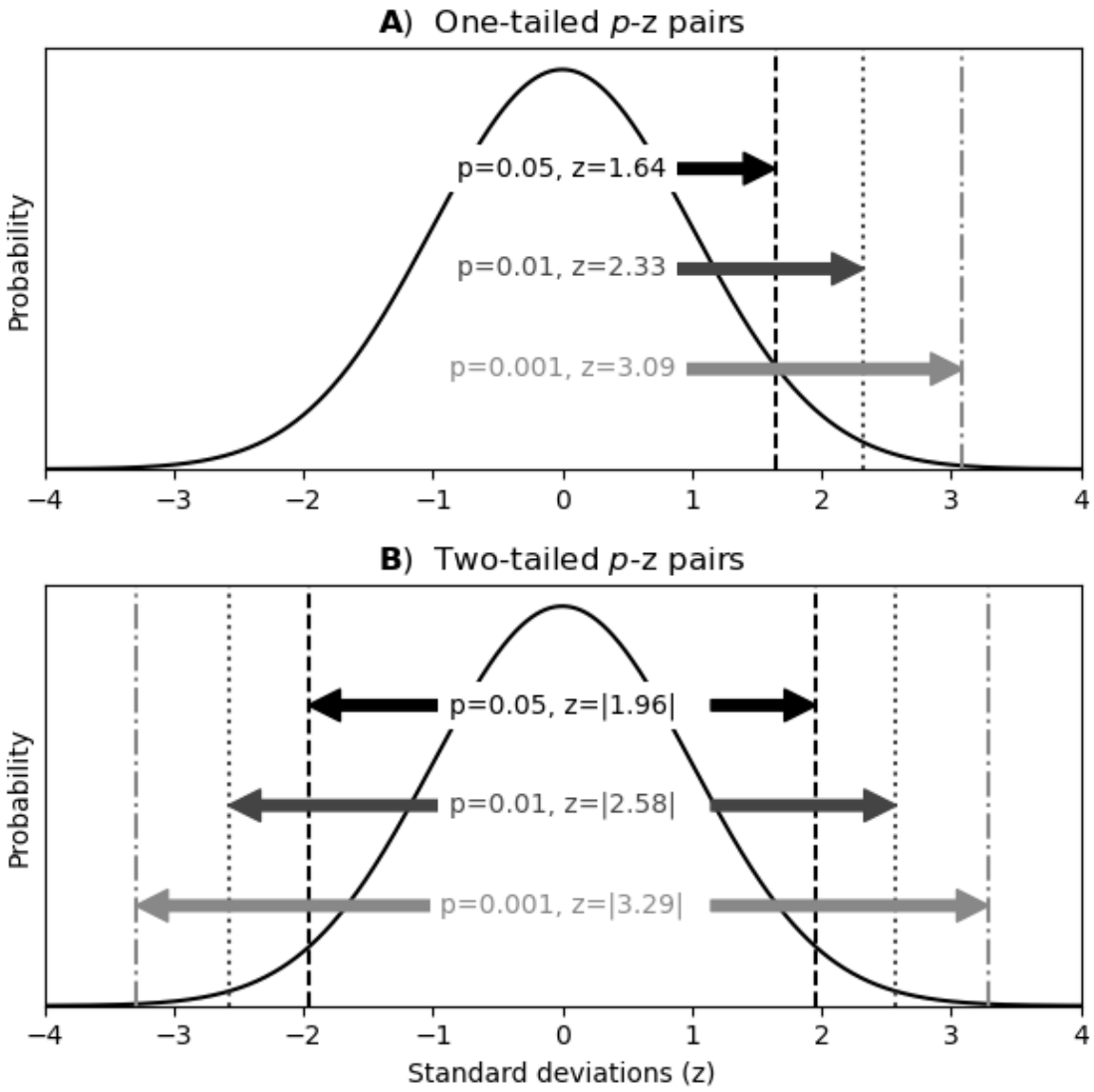
[12]:
```
# create a Gaussian probability curve
z = np.linspace(-4,4,1001)
gpdf = stats.norm.pdf(z)

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

# note that the cdf returns the area *left* of the input value,
# so we subtract 1 to get the area to the *right*.
areaZ1 = 1-stats.norm.cdf(1)

# plot the probability function and the vertical lines
ax.plot(z,gpdf,'k',linewidth=2)
ax.set(xlim=z[[0,-1]],ylim=[0,.42],yticks=[],
       xlabel='Z value',ylabel='Probability')

xidx = np.arange(np.argmin(np.abs(z-1)),len(z))
ax.fill_between(z[xidx],gpdf[xidx],color='k',alpha=.4)

# annotations
tailx = np.argmin(np.abs(x-1.7))
ax.annotate(f'{100*areaZ1:.2f}%',xy=(z[tailx],gpdf[tailx]+.01),
            xytext=(z[tailx]+1.1,gpdf[tailx]+.08),ha='center',
            arrowprops={'color':'k'},weight='bold',size=16)

# significance threshold line
ax.plot([1,1],[0,.4],'k--',linewidth=1.5)

plt.tight_layout()
#plt.savefig('hyp_zgt1.png')
plt.show()
```



9 Figure 10.8: p-z pairs

[14]:
```
# critical p-values to draw
ps2draw = [ .05,.01,.001 ]

stds = np.linspace(-4,4,1001)
prob = stats.norm.pdf(stds)

_,axs = plt.subplots(2,1,figsize=(6,6))

# draw the lines
for a in axs:
  a.plot(stds,prob,'k')
  a.set(ylabel='Probability',yticks=[],xlim=stds[[0,-1]],ylim=[0,.42])

## one-tailed
styles = ['--',':','-.']
for i,p in enumerate(ps2draw):
  # z-value for this p-value
  zval = stats.norm.ppf(1-p)

  # vertical line
  c = i/len(ps2draw)*.8 # line color
  axs[0].axvline(zval,color=(c,c,c),linestyle=styles[i])

  # arrow and text
  axs[0].annotate(f'p={p}, z={zval:.2f}',xy=(zval,.2-(i-1)/10),xytext=(0,.
 ↪2-(i-1)/10),color=[c,c,c],
                  bbox=dict(fc='w',edgecolor='none'),
                  ha='center',va='center',arrowprops={'color':(c,c,c)})

## two-tailed
for i,p in enumerate(ps2draw):
  # z-value for this p-value
  zval = stats.norm.ppf(p/2)

  # vertical line
  c = i/len(ps2draw)*.8 # line color
  axs[1].axvline(zval,color=(c,c,c),linestyle=styles[i])
  axs[1].axvline(-zval,color=(c,c,c),linestyle=styles[i])

  # arrow and text
  axs[1].annotate(f'p={p}, z=|{abs(zval):.2f}|',xy=(zval,.2-(i-1)/10),xytext=(0,.
 ↪2-(i-1)/10),color=[c,c,c],
                  bbox=dict(fc='w',edgecolor='none'),
                  ha='center',va='center',arrowprops={'color':(c,c,c)})
```

```
  axs[1].annotate('',xy=(-zval,.2-(i-1)/10),xytext=(1.15,.2-(i-1)/
↪10),arrowprops={'color':(c,c,c)})

# panel labels
axs[0].set_title(r'$\bf{A}$)  One-tailed $p$-z pairs')
axs[1].set_title(r'$\bf{B}$)  Two-tailed $p$-z pairs')
axs[1].set(xlabel='Standard deviations (z)')

plt.tight_layout()
#plt.savefig('hyp_pz_combos2know.png')
plt.show()
```



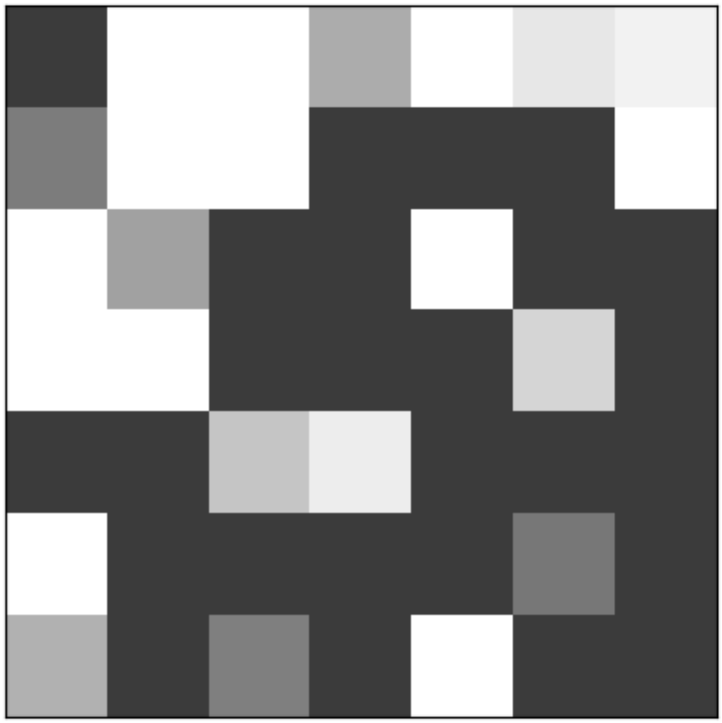
10 Getting p-values from z-values

[15]:
```
zval = 1
pval = stats.norm.cdf(zval)

print(f'The p-value for z = {zval:.2f} is p = {1-pval:.4f}')
```

```
The p-value for z = 1.00 is p = 0.1587
```

11 Figure 10:14 Spatial image for MCC

[18]:
```
# Code used for Figure 10.17
M = np.random.randn(7,7)
M[M<.2] = 0

plt.imshow(M,cmap='gray',vmin=-.3,vmax=1)
plt.xticks([])
plt.yticks([])
plt.show()
```



12 Exercise 1

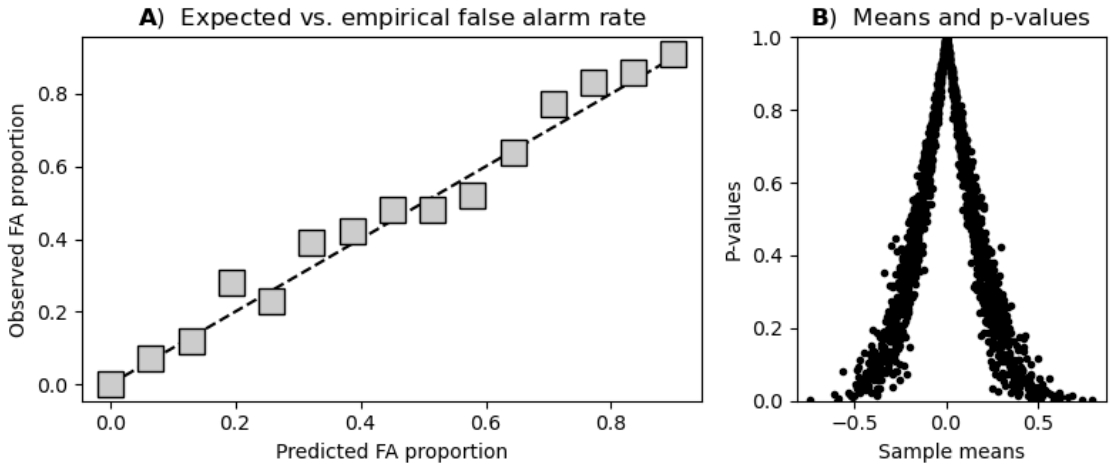
[ ]:
```
# No extra code here; find the relevant code cell, copy/paste that code here,
#  and modify it to have the p-value be soft-coded. Make sure you also update
#  the text in the figures!
```

13 Exercise 2

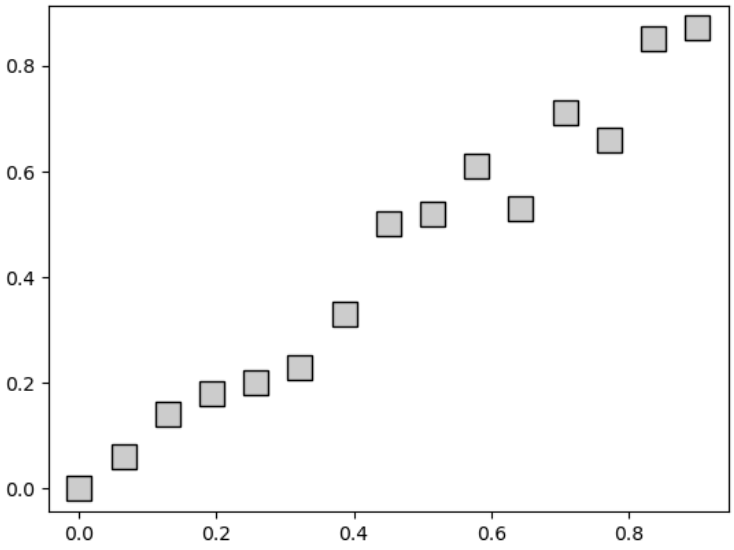
[19]:
```
X = np.random.randn(20)
p = stats.ttest_1samp(X,0)[1]
print(f'The mean is {np.mean(X):.2f} and the p-value from a t-test is {p:.4f}')
```

```
The mean is 0.44 and the p-value from a t-test is 0.0185
```

[21]:
```
# simulate data and t-test
alphaRange = np.linspace(.001,.9,15)

M = 100
type1errors = np.zeros((M,len(alphaRange)))
meansAndPvals = np.zeros((M*len(alphaRange),2))

# loop over alpha's
for ai,alpha in enumerate(alphaRange):
  # loop over experiments
  for expi in range(M):
    # generate the data and compute the p-value from a t-test
    X = np.random.randn(20)
    p = stats.ttest_1samp(X,0)[1]
    # store (as Boolean) whether this test was subthreshold
    type1errors[expi,ai] = p<alpha
    # gather the mean and p-value from this test
    meansAndPvals[expi*len(alphaRange) + ai,:] = [np.mean(X),p]

_,axs = plt.subplots(1,2,width_ratios=[2,1],figsize=(8,3.5))
axs[0].plot(alphaRange,np.mean(type1errors,axis=0),'ks',
         markersize=13,markerfacecolor=(.8,.8,.8))
axs[0].plot(alphaRange,alphaRange,'k--',zorder=-1)
axs[0].set(xlabel='Predicted FA proportion',ylabel='Observed FA proportion')
axs[0].set_title(r'$\bf{A}$)  Expected vs. empirical false alarm rate')

axs[1].plot(meansAndPvals[:,0],meansAndPvals[:,1],'k.')
axs[1].set(xlabel='Sample means',ylabel='P-values',ylim=[0,1])
axs[1].set_title(r'$\bf{B}$)  Means and p-values')

plt.tight_layout()
#plt.savefig('hyp_ex2.png')
plt.show()
```

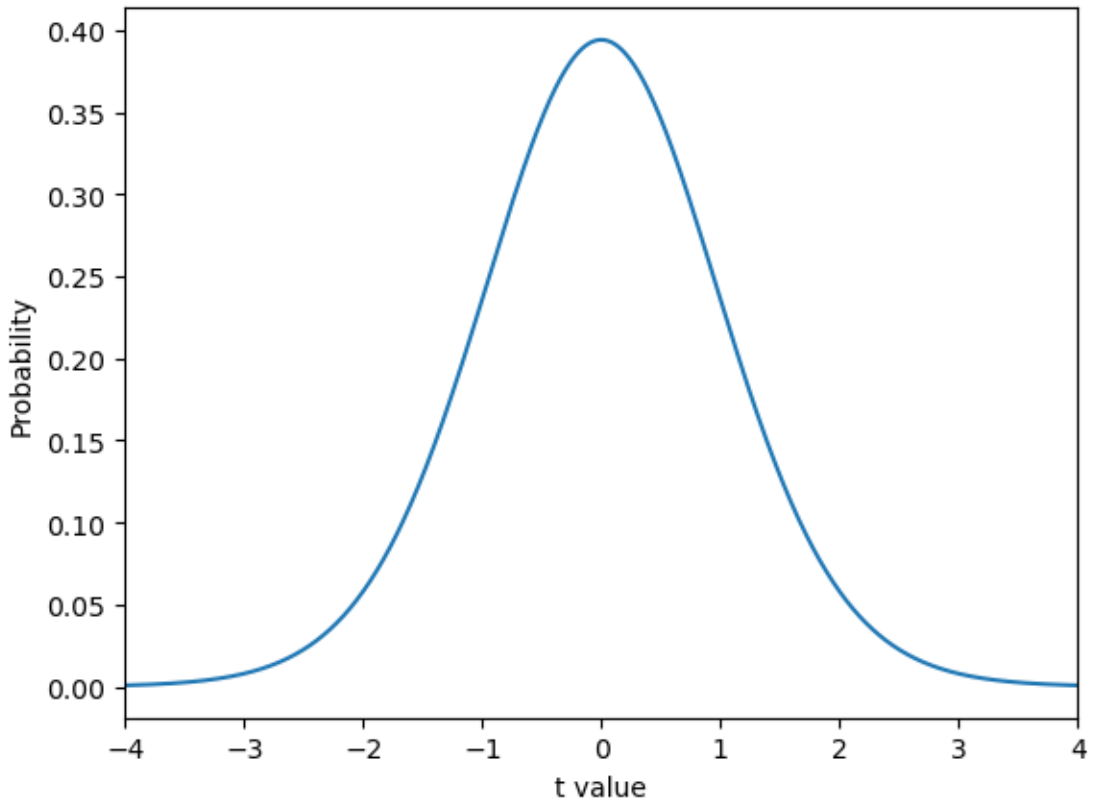


[22]:
```
# Note about the previous code cell:
#  You can implement many tests simultaneously in a matrix, with columns␣
 ↪containing experiments.
#  I'll introduce this in Chapter 12, but I show the code here FYI.

type1errors = np.zeros(len(alphaRange)) # note: vector not matrix!
for ai,alpha in enumerate(alphaRange):
  p = stats.ttest_1samp(np.random.randn(20,M),0)[1] # compare input to that in␣
 ↪previous cell
  type1errors[ai] = np.mean(p<alpha) # here is the averaging
# plot
plt.plot(alphaRange,type1errors,'ks',markersize=13,markerfacecolor=(.8,.8,.8))
plt.show()
```

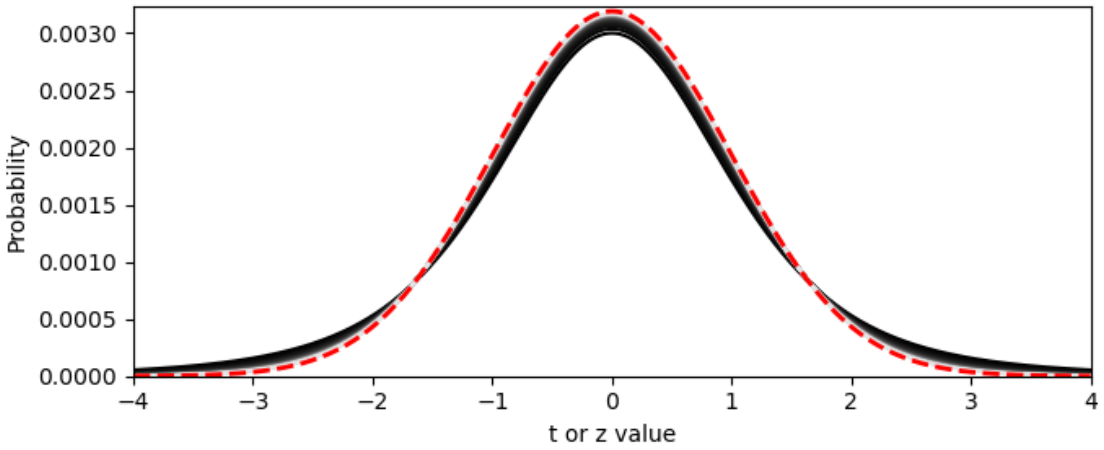


14 Exercise 3

[23]:
```
# one t-value distribution
tvals = np.linspace(-4,4,1001)

# compute the pdf
tpdf = stats.t.pdf(tvals,20)

plt.plot(tvals,tpdf)
plt.xlim(tvals[[0,-1]])
plt.xlabel('t value')
plt.ylabel('Probability')
plt.show()
```



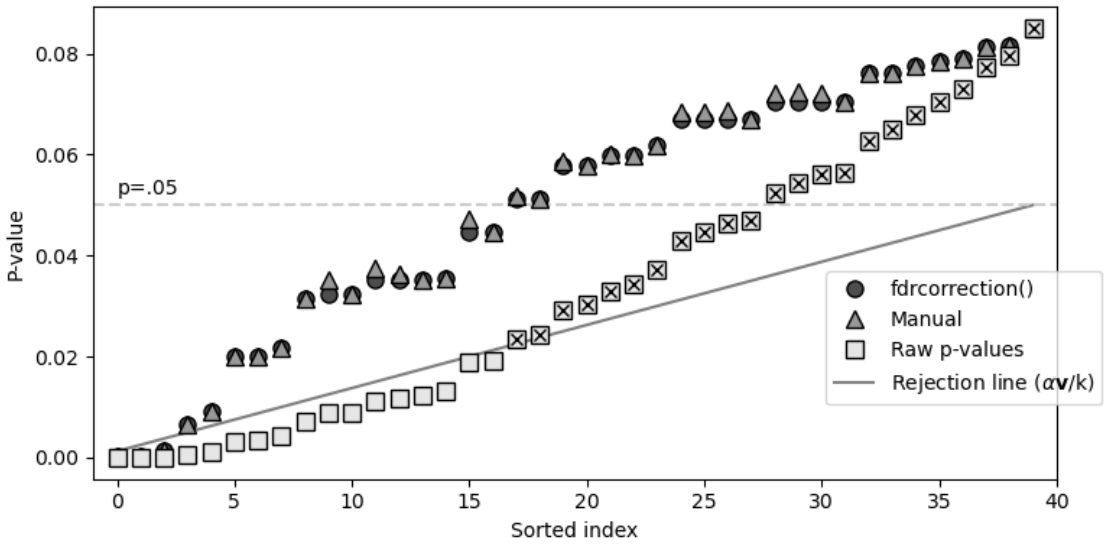
[25]:
```
# t-distributions with different df's
tvals = np.linspace(-4,4,1001)
dfs = np.arange(4,41)

# initialize the figure
plt.figure(figsize=(7,3))

# create and show the pdf's
for df in dfs:
  # compute the pdf
  tpdf = stats.t.pdf(tvals,df) * np.diff(tvals[:2])

  # optional: for a nice visual effect, you can shift the pdf's:
  #tpdf += df/100000

  # plot
  c = (df-np.min(dfs))/np.max(dfs) # color value with scaling
  plt.plot(tvals,tpdf,color=(c,c,c))

# then plot zscores (using "tvals" as z-values here)
plt.plot(tvals,stats.norm.pdf(tvals)*np.diff(tvals[:2]),'r--',linewidth=2)

plt.ylim([0,np.max(tpdf)*1.02])
plt.xlim(tvals[[0,-1]])
plt.xlabel('t or z value')
plt.ylabel('Probability')
plt.tight_layout()
#plt.savefig('hyp_ex3.png')
plt.show()
```



15 Exercise 4

[26]:
```
from statsmodels.stats.multitest import fdrcorrection

# p-value threshold (corrected)
pThresh = .05

# set of p-values
k = 40
pvals = np.random.uniform(low=.001,high=.3,size=k)**2

# step 1: sort the p-values
pvalsSort = np.sort(pvals)

# step 2: linear interpolated distribution
pvalsInterp = np.arange(1,k+1) / k

# step 3: adjusted p-values
pvals_adjusted = pvalsSort / pvalsInterp

# final step implemented in fdrcorrection() is to take the running-minimum of␣
 ↪sorted adjusted p-values.
#pvals_adjusted = np.minimum.accumulate(pvals_adjusted[::-1])[::-1]

# Using the statsmodel function.
# This function returns a tuple with (0) Boolean rejections and (1) adjusted␣
 ↪p-values.
# Here we need only the second output.
qq = fdrcorrection(pvalsSort,pThresh)[1]
# Also note that I'm inputting pvalsSort instead of pvals. That's done to␣
 ↪facilitate the
# plotting; you would normally input the vector of p-values without sorting.

## visualization!
plt.figure(figsize=(8,4))
plt.plot(qq,'ko',markerfacecolor=(.3,.3,.3),markersize=8,label='fdrcorrection()')
plt.plot(pvals_adjusted,'k^',markerfacecolor=(.6,.6,.
 ↪6),markersize=8,label='Manual')
plt.plot(pvalsSort,'ks',markerfacecolor=(.9,.9,.9),markersize=8,label='Raw␣
 ↪p-values')
plt.plot(pThresh*pvalsInterp,color='gray',label=r'Rejection line ($\alpha\bf{v}$/
 ↪k)',zorder=-10)
plt.axhline(pThresh,linestyle='--',color=(.8,.8,.8),zorder=-10)
plt.text(0,.052,'p=.05')

# cross out non-significant p-values
```

```
plt.plot(np.
 ↪where(pvals_adjusted>pThresh)[0],pvalsSort[pvals_adjusted>pThresh],'kx')

# final niceties
plt.legend(bbox_to_anchor=[.75,.46])
plt.xlabel('Sorted index')
plt.ylabel('P-value')
plt.xlim([-1,k])

plt.tight_layout()
#plt.savefig('hyp_ex4.png')
plt.show()
```



16 Exercise 5

[27]:
```
# find the p-value threshold in the non-adjusted p-values

# the p-values that are significant according to FDR correction
H0rejected = pvalsSort <= pvalsInterp*pThresh

# find the largest signficant (raw) pvalue
H0rejected_pvals = np.where(H0rejected)[0]
FDR_threshold = pvalsSort[H0rejected_pvals[-1]]

print(f'Uncorrected p-value threshold based on FDR: q={FDR_threshold:.4f}')
```

```
Uncorrected p-value threshold based on FDR: q=0.0190
```

17 Exercise 6

[28]:
```
N = 100

# p-values
pvals = np.random.uniform(.001,.25,N)**2

# thresholds
bon_thresh = .05/N
q = fdrcorrection(pvals,.05)

# print messages
print(f'       FDR led to {100*np.mean(q[0]):2.0f}% significant tests.')
print(f'Bonferroni led to {100*np.mean(pvals<bon_thresh):2.0f}% significant␣
 ↪tests.')
```

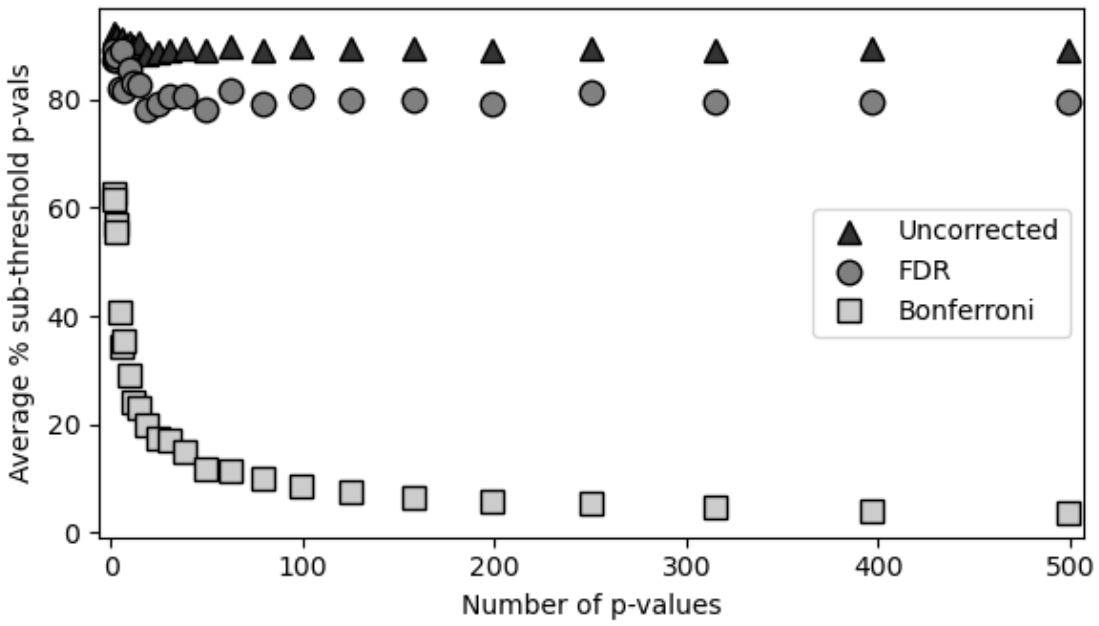
```
       FDR led to 86% significant tests.
Bonferroni led to 11% significant tests.
```

[29]:
```
# Experiment repetitions with the same N=100

sigTests = np.zeros((100,2))

for expi in range(100):

  pvals = np.random.uniform(.001,.25,N)**2
  bon_thresh = .05/N
  q = fdrcorrection(pvals,.05)

  # record the results
  sigTests[expi,0] = 100*np.mean(q[0])
  sigTests[expi,1] = 100*np.mean(pvals<bon_thresh)

# report the average and std
print(f'       FDR: mean of {np.mean(sigTests[:,0]):5.2f}%  (std: {np.
 ↪std(sigTests[:,0],ddof=1):.2f}) significant tests.')
print(f'Bonferroni: mean of {np.mean(sigTests[:,1]):5.2f}%  (std: {np.
 ↪std(sigTests[:,1],ddof=1):.2f}) significant tests.')
```

```
       FDR: mean of 80.45%  (std: 7.39) significant tests.
Bonferroni: mean of  8.89%  (std: 3.01) significant tests.
```

18 Exercise 7

[31]:
```
# You need to have run the code for exercise 6 before this code.

# the p-value set size
Ns = np.logspace(np.log10(2),np.log10(500),25,dtype=int)

# number of experiment repetitions
nRepetitions = 100

# results matrix (note: not storing the result of each repetition)
sigTests = np.zeros((len(Ns),3))

# and away we go!
for ni,n in enumerate(Ns):
  # loop over experiment repetitions
  for _ in range(nRepetitions): # we don't need the counting variable here...
    # p-values and corrections
    pvals = np.random.uniform(.001,.25,n)**2
    bon_thresh = .05/n
    q = fdrcorrection(pvals,.05)

    # record the results (note the summation)
    sigTests[ni,0] += 100*np.mean(pvals<.05)        # uncorrected
    sigTests[ni,1] += 100*np.mean(q[0])             # FDR-corrected
    sigTests[ni,2] += 100*np.mean(pvals<bon_thresh) # Bonferroni-corrected

# the code above keeps summing, so now divide by M repetitions to average
sigTests /= nRepetitions

# now for the visualization
plt.figure(figsize=(6,3.5))
plt.plot(Ns,sigTests[:,0],'k^',markersize=9,markerfacecolor=(.2,.2,.
 ↪2),label='Uncorrected')
plt.plot(Ns,sigTests[:,1],'ko',markersize=9,markerfacecolor=(.5,.5,.
 ↪5),label='FDR')
plt.plot(Ns,sigTests[:,2],'ks',markersize=9,markerfacecolor=(.8,.8,.
 ↪8),label='Bonferroni')

plt.legend()
plt.xlabel('Number of p-values')
plt.ylabel('Average % sub-threshold p-vals')
plt.xlim([Ns[0]-8,Ns[-1]+8])

plt.tight_layout()
#plt.savefig('hyp_ex7.png')
plt.show()
```



[ ]:
```
# In case you were wondering: the motivation for logarithmic scaling of set size
# is the most of the interesting action happens with small samples. You can try
# using a linear increase, or try setting the x-axis scale to be logarithmic.
```

19 Exercise 8

[32]:
```
# parameters
sampleSizeSkip = 3
sampleSizeMax  = 201

# initialize data variable and output vector
data = np.random.randn(5)
pvals = []
ssizes = []

while len(data)<sampleSizeMax:
  # compute the p-value and sample sizes
  pvals.append( stats.ttest_1samp(data,0).pvalue )
  ssizes.append( len(data) )

  # add more data!
  data = np.append(data,np.random.randn(sampleSizeSkip))

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

```
plt.plot(ssizes,pvals,'ko',markersize=10,markerfacecolor=[.7,.7,.7])
plt.axhline(y=.05,color='k',linestyle='--')
plt.ylim([0,1.05])
plt.xlabel('Sample sizes')
plt.ylabel('P-value')
plt.title('P-values in random Gaussian numbers',loc='center')

plt.tight_layout()
#plt.savefig('hyp_ex8.png')
plt.show()
```