# stats_ch17_power

August 6, 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 17**

[1]:
```
import numpy as np
import scipy.stats as stats
import pandas as pd
import matplotlib.pyplot as plt

# statsmodels library for computing power
import statsmodels.stats.power as smp
# define global figure properties used for publication
import matplotlib_inline.backend_inline
```

# 2 Power for a one-sample t-test

[2]:
```
# parameters
xBar = 1
h0   = 0
std  = 2
n    = 42 # sample size
alpha = .05  # significance level

# Compute the non-centrality parameter
tee = (xBar-h0) / (std/np.sqrt(n))

# Critical t-values (2-tailed)
df = n - 1  # df for one-sample t-test
t_critL = stats.t.ppf(alpha/2, df) # in Equation 17.2, this is -\tau_{\alpha/
↪2,df}
t_critR = stats.t.ppf(1-alpha/2, df)

# two one-sided power areas
powerL = stats.t.cdf(t_critL+tee, df) # note shiting the distribution
powerR = 1 - stats.t.cdf(t_critR+tee, df)
```

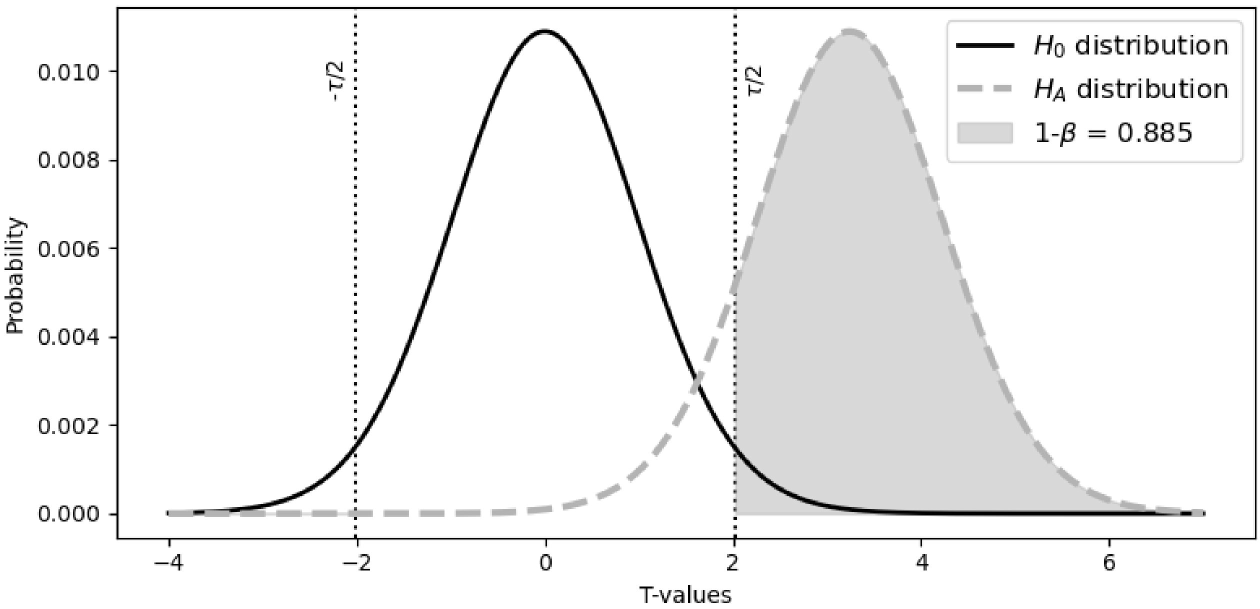
```
# note: can also use the loc input:
#powerL = stats.t.cdf(t_critL, df, loc=delta)

# total power
totalPower = powerL + powerR

# and report
print(f't = {tee:.3f}')
print(f'shifted tau-left = {t_critL+tee:.3f}')
print(f'shifted tau-right = {t_critR+tee:.3f}')
print('')
print(f'Statistical power: {totalPower:.4f}')
```

```
t = 3.240
shifted tau-left = 1.221
shifted tau-right = 5.260

Statistical power: 0.8854
```

## 3 Figure 17.2: Visualization of statistical power for this example

[3]:
```
#### simulation parameters ####
# this is the same code as above, but repeated here to make it easy for you to␣
↪modify and explore

# parameters
xBar = 1
h0   = 0
std  = 2
n    = 42 # sample size
alpha = .05  # significance level

# Compute the non-centrality parameter
tee = (xBar-h0) / (std/np.sqrt(n))

# Critical t-values (2-tailed)
df = n - 1  # df for one-sample t-test
t_critL = stats.t.ppf(alpha/2, df) # in Equation 17.2, this is -\tau_{\alpha/
↪2,df}
t_critR = stats.t.ppf(1-alpha/2, df)

# two one-sided power areas
powerL = stats.t.cdf(t_critL+tee, df) # note shiting the distribution
powerR = 1 - stats.t.cdf(t_critR+tee, df)
totalPower = powerL + powerR
```

```
########## now for the visualization ############
# t-values
tvals = np.linspace(-4,7,401)

# open the figure
plt.figure(figsize=(8,4))

# draw the distributions
plt.plot(tvals,stats.t.pdf(tvals,n-1)*np.diff(tvals[:2]),'k',linewidth=2,␣
↪label='$H_0$ distribution')
plt.plot(tvals,stats.t.pdf(tvals-tee,n-1)*np.diff(tvals[:
↪2]),'--',linewidth=3,color=(.7,.7,.7), label=r'$H_A$ distribution')

# critical t-values at alpha=.025 ("tau" in the equations)
plt.axvline(t_critL,color='k',linestyle=':',zorder=-3)
plt.axvline(t_critR,color='k',linestyle=':',zorder=-3)
plt.text(t_critL-.2,stats.t.pdf(0,n-1)*.9*np.diff(tvals[:2]),r'-$\tau$/
↪2',rotation=90,ha='center',va='center')
plt.text(t_critR+.22,stats.t.pdf(0,n-1)*.9*np.diff(tvals[:2]),r'$\tau$/
↪2',rotation=90,ha='center',va='center')

# fill in areas for computing 1-beta (note: basically invisible on the left side;
↪ try setting xBar=-1)
plt.fill_between(tvals,stats.t.pdf(tvals-tee,n-1)*np.diff(tvals[:
↪2]),where=(tvals<t_critL),color=(.7,.7,.7),alpha=.5)
plt.fill_between(tvals,stats.t.pdf(tvals-tee,n-1)*np.diff(tvals[:
↪2]),where=(tvals>t_critR),color=(.7,.7,.7),alpha=.5,label=fr'1-$\beta$ =␣
↪{totalPower:.3f}')

plt.xlabel('T-values')
plt.ylabel('Probability')
plt.legend(fontsize=12)

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




## 4 Using statsmodels

[4]:
```
# parameters
xBar = 1
h0   = 0
std  = 2 # sample standard deviation
sampsize = 42
alpha = .05  # significance level

effectSize = (xBar-h0) / std
power_sm = smp.TTestPower().power(
    effect_size=effectSize, nobs=sampsize, alpha=alpha, alternative='two-sided')

print(f'Statistical power using statsmodels: {power_sm:.4f}')
```

```
Statistical power using statsmodels: 0.8856
```

## 5 Sample size for a desired power

[5]:
```
# parameters
power = .8  # desired statistical power level (1-\beta)
h0    = 0   # mean if H0 is true
xBar  = 1   # sample mean
std   = 1.5 # sample standard deviation

# effect size
effectSize = (xBar-h0) / std
```

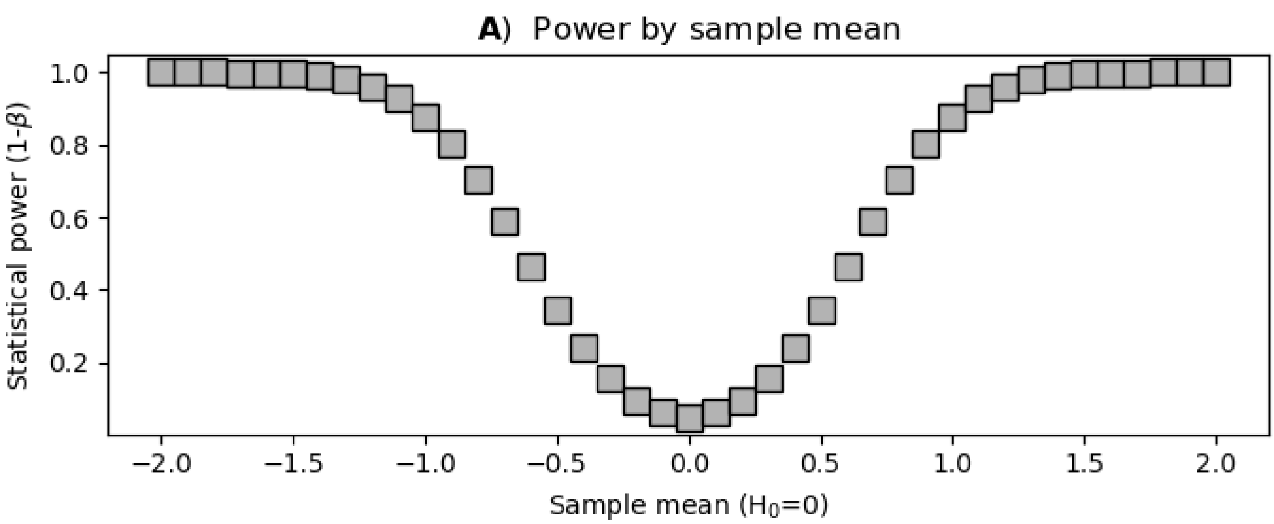
```
# compute sample size
sample_size = smp.TTestPower().solve_power(
    effect_size=effectSize, alpha=.05, power=power, alternative='two-sided')
# and report
print(f'Required sample size: {round(sample_size)}')
```

```
Required sample size: 20
```

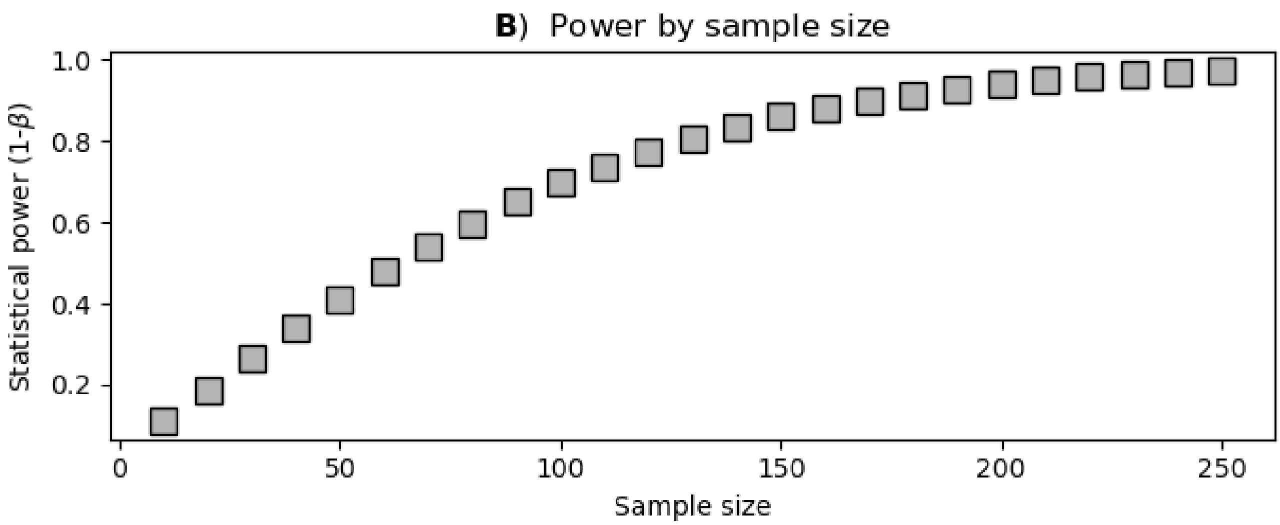
## 6 Exercise 1

[6]:

```
# parameters
std  = 2
sampsize = 41
xBars = np.linspace(-2,2,41)

# initialize results vector
powers = np.zeros(len(xBars))
# run the experiment!
for i,xm in enumerate(xBars):
  powers[i] = smp.TTestPower().power(effect_size=xm/std, nobs=sampsize, alpha=.
 ↪05)

# and plot the results
plt.figure(figsize=(7,3))
plt.plot(xBars,powers,'ks',markersize=10,markerfacecolor=(.7,.7,.7))
plt.xlabel(r'Sample mean (H$_0$=0)')
plt.ylabel(r'Statistical power (1-$\beta$)')
plt.title(r'$\bf{A}$)  Power by sample mean')
plt.tight_layout()
#plt.savefig('power_ex1a.png')
plt.show()
```

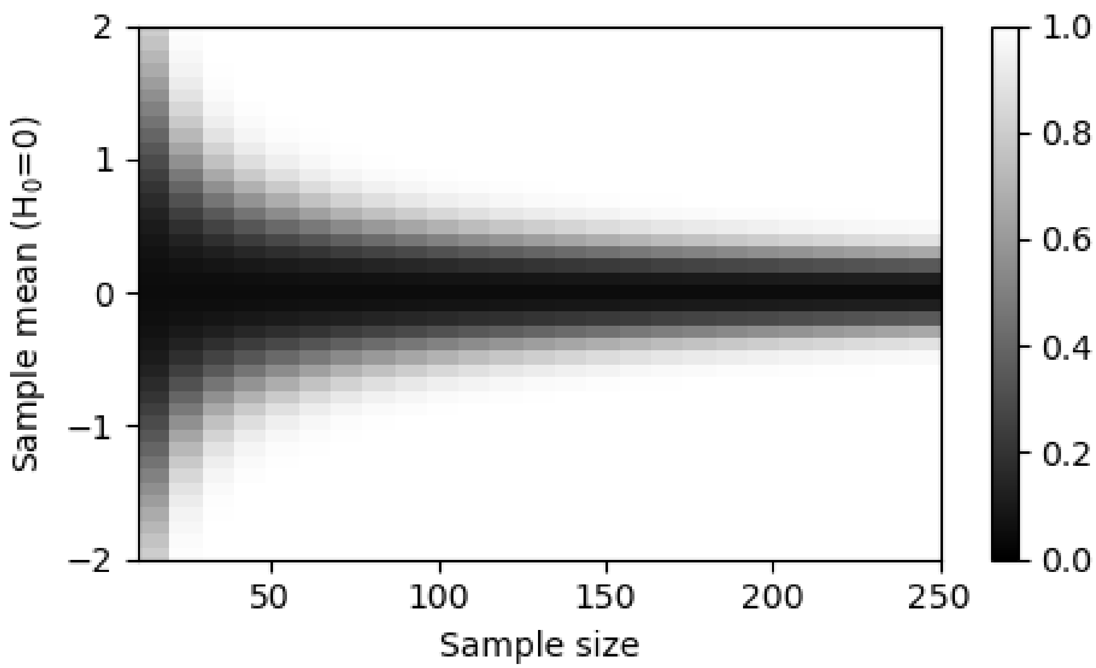


[7]:
```
# parameters
xBar = .5
sampleSizes = np.arange(10,251,step=10)

# initialize results vector
powers = np.zeros(len(sampleSizes))

# the experiment
for i,ss in enumerate(sampleSizes):
  powers[i] = smp.TTestPower().power(effect_size=xBar/std, nobs=ss, alpha=.05)

# plot the results
plt.figure(figsize=(7,3))
plt.plot(sampleSizes,powers,'ks',markersize=10,markerfacecolor=(.7,.7,.7))
plt.xlabel('Sample size')
plt.ylabel(r'Statistical power (1-$\beta$)')
plt.title(r'$\bf{B}$)  Power by sample size')

plt.tight_layout()
#plt.savefig('power_ex1b.png')
plt.show()
```




[8]:
```
# parameters
xBars = np.linspace(-2,2,41)
sampleSizes = np.arange(10,251,step=10)

# initialize the results matrix
powers = np.zeros((len(xBars),len(sampleSizes)))
```

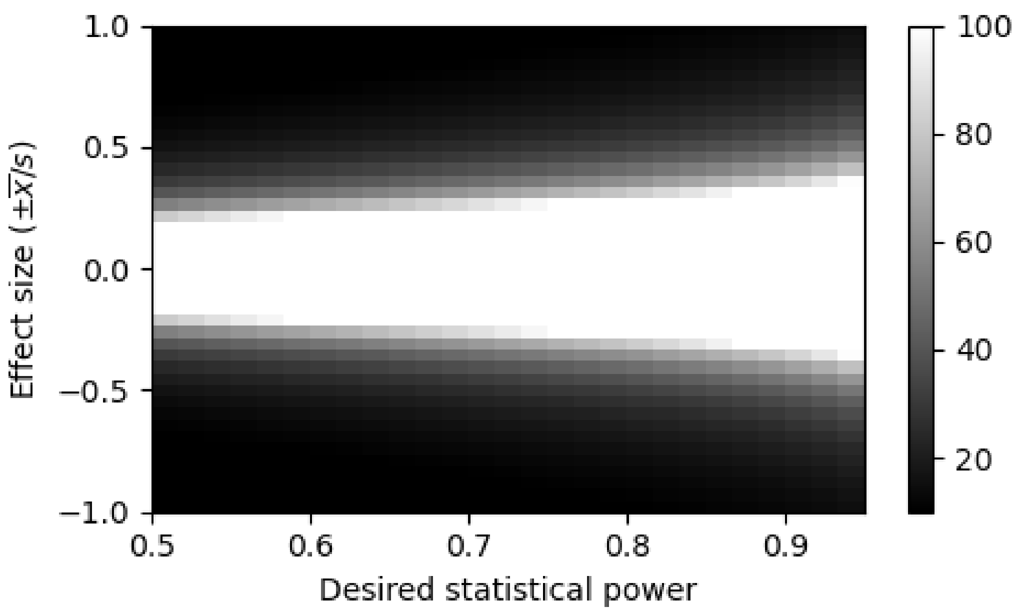
```
# run the experiment (manipulate mean and N independently)
for xi,xm in enumerate(xBars):
  for si,ss in enumerate(sampleSizes):
    powers[xi,si] = smp.TTestPower().power(effect_size=xm/std, nobs=ss, alpha=.
 ↪05)

# and the results...
plt.figure(figsize=(5,3))
plt.
 ↪imshow(powers,origin='lower',extent=[sampleSizes[0],sampleSizes[-1],xBars[0],xBars[-1]],
           aspect='auto',cmap='gray',vmin=0,vmax=1)
plt.colorbar()
plt.xlabel('Sample size')
plt.ylabel(r'Sample mean (H$_0$=0)')
plt.yticks(range(-2,3))

plt.tight_layout()
#plt.savefig('power_ex1c.png')
plt.show()
```




# 7 Exercise 2

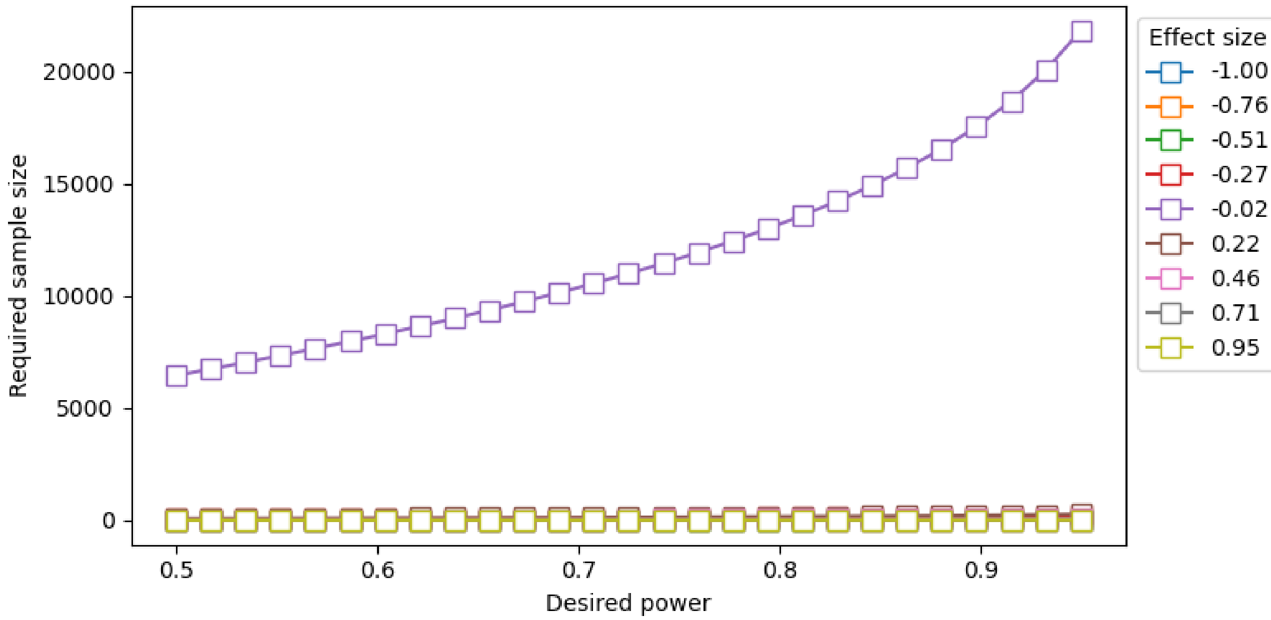
[9]:
```
# (Note: some variables in this exercise were defined in Exercise 1)
# parameters
xBars = np.linspace(-2,2,42)
power = np.linspace(.5,.95,27)
```

```
# initialize the results matrix
sampleSizes = np.zeros((len(xBars),len(power)))

# run the experiment (manipulate mean and N independently)
for xi,xm in enumerate(xBars):
  for pi,pwr in enumerate(power):
    sampleSizes[xi,pi] = smp.TTestPower().solve_power(effect_size=xm/std, alpha=.
↪05, power=pwr)

# and the results...
plt.figure(figsize=(5,3))
plt.imshow(sampleSizes,origin='lower',extent=[power[0],power[-1],xBars[0]/
↪std,xBars[-1]/std],
           aspect='auto',cmap='gray',vmin=10,vmax=100)
plt.colorbar()
plt.xlabel('Desired statistical power')
plt.ylabel(r'Effect size ($\pm\overline{x}/s$)')
plt.yticks(np.arange(-1,1.1,.5))

plt.tight_layout()
#plt.savefig('power_ex2a.png')
plt.show()
```




[10]:
```
plt.figure(figsize=(8,4))

# (un)comment one of these lines
rows2plot = slice(0,len(xBars),5)
# rows2plot = slice(0,5)
```

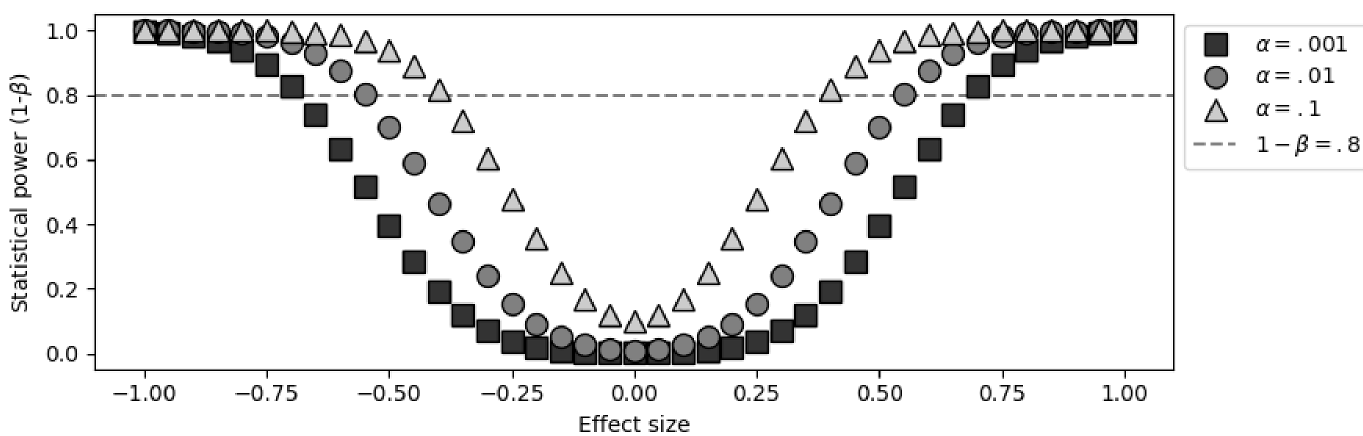
```
# for the plotting
plt.plot(power,sampleSizes[rows2plot,:].T,'s-',markerfacecolor='w',markersize=8)
plt.legend([f'{v/std:.2f}' for v in␣
 ↪xBars[rows2plot]],bbox_to_anchor=(1,1),title='Effect size')
plt.xlabel('Desired power')
plt.ylabel('Required sample size')
plt.tight_layout()
#plt.savefig('power_ex2b.png')
plt.show()
```




# 8 Exercise 3

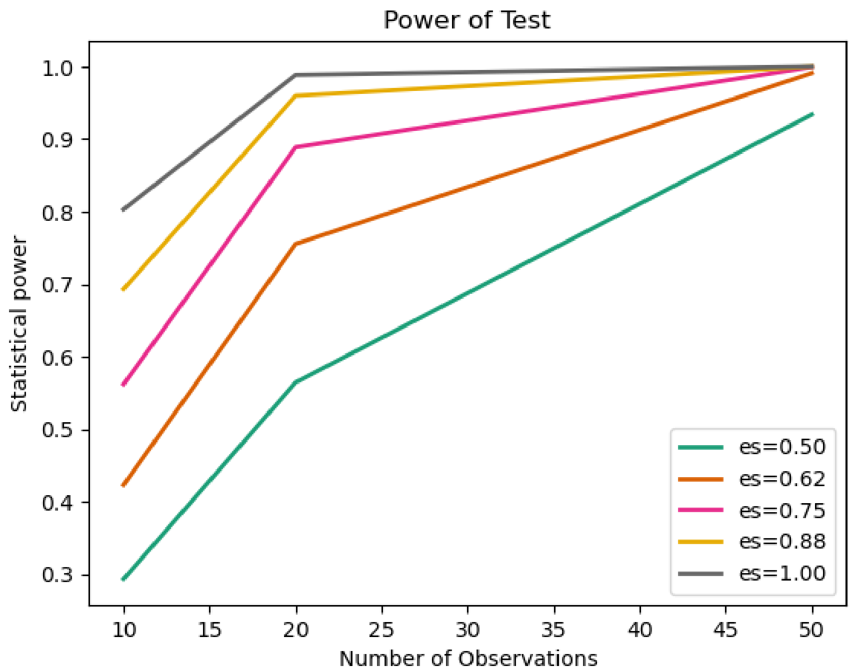
[11]:
```
# parameters
std  = 2
sampsize = 42
xBars = np.linspace(-2,2,41)

# initialize results vector
powers = np.zeros((len(xBars),3))
# run the experiment!
for i,xm in enumerate(xBars):
  powers[i,0] = smp.TTestPower().power(effect_size=xm/std, nobs=sampsize, alpha=.
 ↪001)
  powers[i,1] = smp.TTestPower().power(effect_size=xm/std, nobs=sampsize, alpha=.
 ↪01)
  powers[i,2] = smp.TTestPower().power(effect_size=xm/std, nobs=sampsize, alpha=.
 ↪1)
```

```
# for an extra challenge, put the alpha values into a for-loop by raising 10 to␣
↪higher negative powers!
#and plot the results
plt.figure(figsize=(9,3))
plt.plot(xBars/std,powers[:,0],'ks',markersize=10,markerfacecolor=(.2,.2,.
↪2),label=r'$\alpha=.001$')
plt.plot(xBars/std,powers[:,1],'ko',markersize=10,markerfacecolor=(.5,.5,.
↪5),label=r'$\alpha=.01$')
plt.plot(xBars/std,powers[:,2],'k^',markersize=10,markerfacecolor=(.8,.8,.
↪8),label=r'$\alpha=.1$')
plt.axhline(y=.8,linestyle='--',color='gray',zorder=-10,label=r'$1-\beta=.8$')
plt.xlabel(r'Effect size')
plt.ylabel(r'Statistical power (1-$\beta$)')
plt.legend(bbox_to_anchor=(1,1))

plt.tight_layout()
#plt.savefig('power_ex3.png')
plt.show()
```



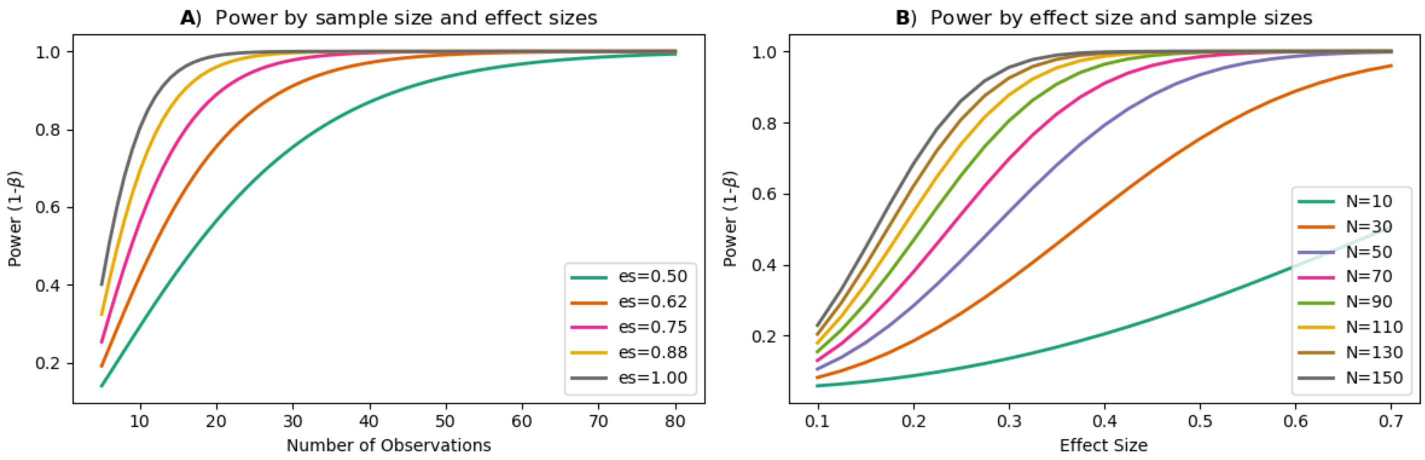

## 9 Exercise 4

[12]:
```
# basic usage (this is the code I showed in the book)
smp.TTestPower().plot_power(dep_var='nobs',nobs=np.
↪array([10,20,50]),effect_size=np.linspace(.5,1,5))
plt.ylabel('Statistical power')
plt.show()
```




[13]:

```
### here's the solution to the exercise

# setup a figure
_,axs = plt.subplots(1,2,figsize=(12,4))

# call the power plot calculations
smp.TTestPower().plot_power(dep_var='nobs',nobs=np.arange(5,81),effect_size=np.
 ↪linspace(.5,1,5),ax=axs[0])
smp.TTestPower().plot_power(dep_var='effect_size',nobs=np.
 ↪arange(10,151,20),effect_size=np.linspace(.1,.7,25),ax=axs[1])

# some plot adjustments
axs[0].set_title(r'$\bf{A}$)  Power by sample size and effect sizes')
axs[1].set_title(r'$\bf{B}$)  Power by effect size and sample sizes')
axs[0].set_ylabel(r'Power (1-$\beta$)')
axs[1].set_ylabel(r'Power (1-$\beta$)')

# fix strange issue of sample sizes printing as 10.00
axs[1].legend([l[:-3] for l in axs[1].get_legend_handles_labels()[1]])

plt.tight_layout()
#plt.savefig('power_ex4.png')
plt.show()
```



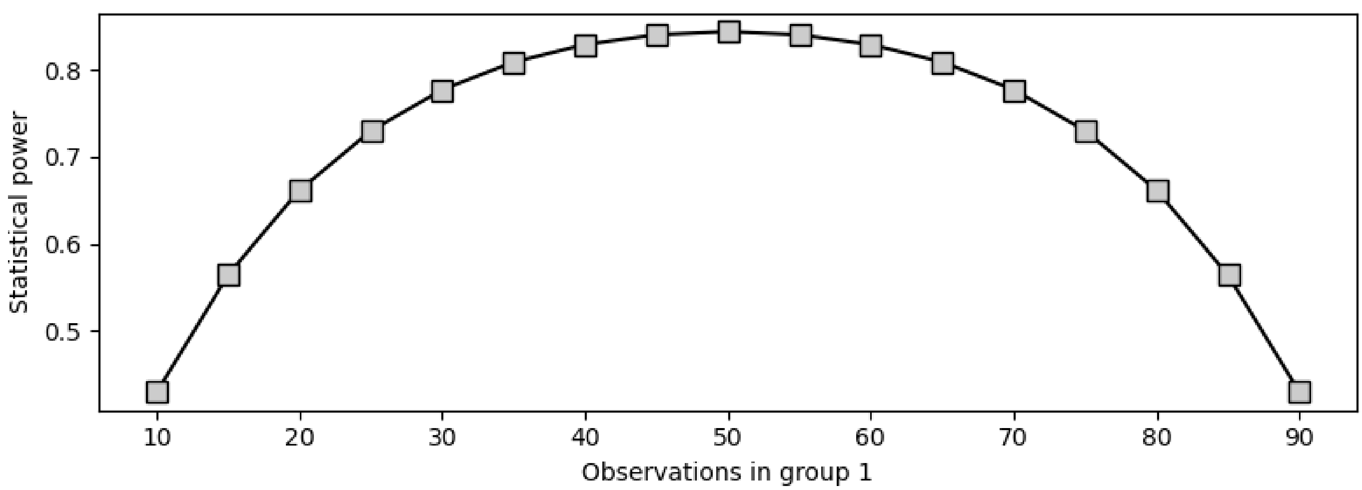

## 10 Exercise 5

[14]:
```
# simulation parameters
effectSize = .6  # group mean differences, divided by standard deviation.
n1 = 50          # sample size of group 1.
ssRatio = 2      # Ratio of sample size of group 2 to group 1. 1 means equal␣
 ↪sample sizes.

# Note about sample size parameters (text below is taken from https://www.
 ↪statsmodels.org/dev/generated/statsmodels.stats.power.TTestIndPower.power.
 ↪html#statsmodels.stats.power.TTestIndPower.power)
# n1: number of observations of sample 1. The number of observations of sample␣
 ↪two
# is ratio times the size of sample 1, i.e. nobs2 = nobs1 * ratio

# Compute power
power = smp.TTestIndPower().power(effect_size=effectSize, nobs1=n1, alpha=.05,␣
 ↪ratio=ssRatio)
print(f'Total sample size is {n1}+{n1*ssRatio}={n1+n1*ssRatio}, power is {power:.
 ↪2f}')
```

```
Total sample size is 50+100=150, power is 0.93
```

[15]:
```
# the total sample size
totalN = 100

# sample sizes in group 1
n1sampleSizes = np.arange(10,91,5)

# initialize results vector
powers = np.zeros(len(n1sampleSizes))
```

```
# run the simulation!
for i,n1 in enumerate(n1sampleSizes):
  # calculate the n2 sample size
  n2 = totalN-n1

  # the ratio
  sr = n2/n1

  # compute and store power
  powers[i] = smp.TTestIndPower().power(effect_size=.6, nobs1=n1, alpha=.05,␣
 ↪ratio=sr)

# plot the results
plt.figure(figsize=(8,3))
plt.plot(n1sampleSizes,powers,'ks-',markersize=9,markerfacecolor=(.8,.8,.8))
plt.xlabel('Observations in group 1')
plt.ylabel('Statistical power')

plt.tight_layout()
#plt.savefig('power_ex5.png')
plt.show()
```




## 11 Exercise 6

[16]:
```
# population and sample parameters
mu    = .8
sigma = 1.8
n     = 38
# critical t-values (2-tailed)
t_critL = stats.t.ppf(.05/2, n-1)
t_critR = stats.t.ppf(1-.05/2, n-1)
```

```
# simulation parameters
num_simulations = 10000
rejectH0 = 0 # initialize a counter for when H0 was rejected

# run the experiment!
for _ in range(num_simulations):
  # draw a sample from a population with known parameters
  sample = np.random.normal(mu,sigma,n)
  sample_mean = np.mean(sample)
  sample_se = np.std(sample,ddof=1) / np.sqrt(n)
  # Calculate the t-statistic
  tVal = sample_mean / sample_se
  # Check if t-stat falls into the rejection region
  if tVal<t_critL or tVal>t_critR:
    rejectH0 += 1

# Estimate empirical power (percent of simulations where H0 was rejected)
powerEm = 100 * rejectH0/num_simulations

## compute analytic power from formula
effectSize = mu / sigma # using population parameters
powerAn = 100 * smp.TTestPower().power(effect_size=effectSize, nobs=n, alpha=.05)

# print the results
print(f'Theoretical power from formula:   {powerAn:.3f}%')
print(f'Empirical power from simulations: {powerEm:.3f}%')
```

```
Theoretical power from formula:   76.055%
Empirical power from simulations: 76.690%
```

[18]:
```
### Note about the code in the previous cell: I wrote out the mechanics of the␣
↪t-test so you
#   could see the link to the formula for computing statistical power. In␣
↪practice, it's simpler
#   to use the ttest function in scipy. The code below produces the same result␣
↪using less code.

# re-initialize the counter!
rejectH0 = 0

# run the experiment!
for _ in range(num_simulations):
  # draw a sample from a population with known parameters
  sample = np.random.normal(mu,sigma,n)
  # up the counter if the t-value is significant
  if stats.ttest_1samp(sample,0).pvalue<.05:
    rejectH0 += 1
```

```
# Estimate empirical power (percent of simulations where H0 was rejected)
powerEm = 100 * rejectH0/num_simulations

## compute analytic power from formula
effectSize = mu / sigma # using population parameters
powerAn = 100 * smp.TTestPower().power(effect_size=effectSize, nobs=n, alpha=.05)

# print the results
print(f'Analytical power from formula:    {powerAn:.3f}%')
print(f'Empirical power from simulations: {powerEm:.3f}%')
```

```
Analytical power from formula:    76.055%
Empirical power from simulations: 76.490%
```

## 12 Exercise 7

[19]:
```
# population and sample parameters
mu    = .8
sigma = 1.8
n     = 38

# simulation parameters
num_simulations = 10000
rejectH0 = 0 # initialize a counter for when H0 was rejected

# run the experiment!
for _ in range(num_simulations):
  # draw a sample from a population with known parameters
  sample = np.exp( np.random.normal(mu,sigma,n) )
  # up the counter if the t-value is significant
  if stats.ttest_1samp(sample,0).pvalue<.05:
    rejectH0 += 1

# Estimate empirical power (percent of simulations where H0 was rejected)
powerEm = 100 * rejectH0/num_simulations

## compute analytic power from formula
popMean = np.exp(mu + sigma**2/2)
popStd  = np.exp(mu + sigma**2/2) * np.sqrt(np.exp(sigma**2)-1)
effectSize = popMean / popStd # using population parameters
powerAn = 100 * smp.TTestPower().power(effect_size=effectSize, nobs=n, alpha=.05)
# print the results
print(f'Analytical power from formula:    {powerAn:.3f}%')
print(f'Empirical power from simulations: {powerEm:.3f}%')
```

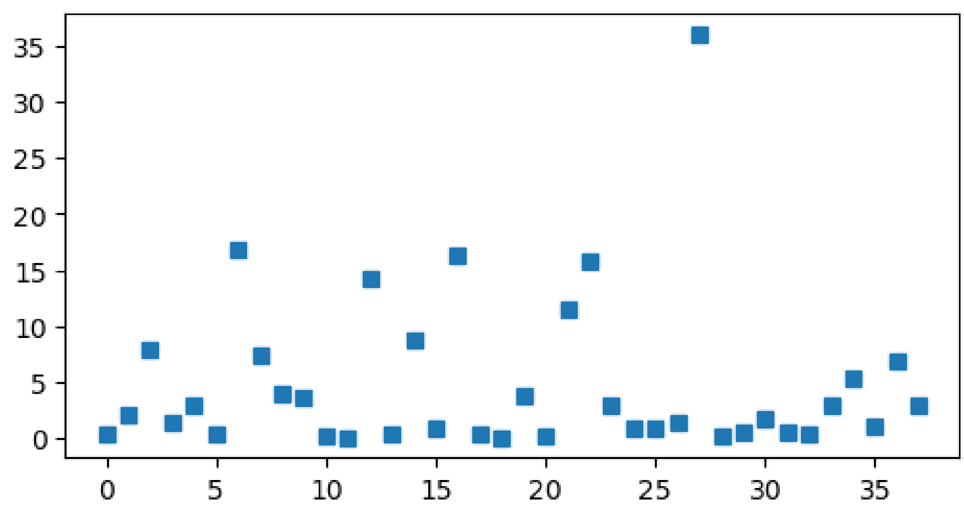
```
Analytical power from formula:    22.811%
Empirical power from simulations: 84.490%
```

## 13 Exercise 8

[20]:
```
## repeat using Wilcoxon test against H0=0

# initialize a counter for when H0 was rejected
rejectH0 = 0

# run the experiment!
for _ in range(num_simulations):
  # draw a sample from a population with known parameters
  sample = np.exp( np.random.normal(mu,sigma,n) )
  W = stats.wilcoxon(sample)
  if W.pvalue<.05:
    rejectH0 += 1

# Estimate empirical power (percent of simulations where H0 was rejected)
powerEm = 100 * rejectH0/num_simulations

# print the results
print(f'Analytical power from simulations: {powerEm:.3f}%')
```

```
Analytical power from simulations: 100.000%
```

[24]:
```
plt.figure(figsize=(6,3))
plt.plot(sample,'s');
```




[25]:
```
# now using a more reasonable H0 value
h0 = 10
# don't forget to keep resetting this counter!
rejectH0 = 0
```

```
# run the experiment!
for _ in range(num_simulations):
  # draw a sample from a population with known parameters
  sample = np.exp( np.random.normal(mu,sigma,n) )
  W = stats.wilcoxon(sample-h0)
  if W.pvalue<.05:
    rejectH0 += 1

# Estimate empirical power (percent of simulations where H0 was rejected)
powerEm = 100 * rejectH0/num_simulations

# print the results
print(f'Empirical power from simulations: {powerEm:.3f}%')
```

```
Empirical power from simulations: 80.680%
```

## 14 Exercise 9

[26]:
```
# import and process data (take from Exercise 11.12)

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/
 ↪winequality-red.csv"
data = pd.read_csv(url,sep=';')

# which columns to t-test
cols2test = data.keys()
cols2test = cols2test.drop('quality')

# create a new column for binarized (boolean) quality
data['boolQuality'] = False
data['boolQuality'][data['quality']>5] = True
```

[27]:
```
# statistical threshold (Bonferroni-corrected)
bonP = .05/len(cols2test)

# loop over column
for col in cols2test:
  # for convenience, extract the numerical variables
  Xh = data[col][data['boolQuality']==True].values  # high rating
  Xl = data[col][data['boolQuality']==False].values # low rating

  # sample size and ratio
  nh = len(Xh)
  nl = len(Xl)
  sr = nh/nl

  # effect size (es)
  es_num = np.mean(Xh)-np.mean(Xl)
  es_den = np.sqrt( ( (nh-1)*np.var(Xh,ddof=1)+(nl-1)*np.var(Xl,ddof=1) )/
 ↪(nh+nl-2) )

  # compute power
  power = smp.TTestIndPower().power(effect_size=es_num/es_den, nobs1=nh,␣
 ↪alpha=bonP, ratio=sr)

  # run the t-test
  tres = stats.ttest_ind(Xh,Xl,equal_var=False)

  # print the results
  print(f'{col:>20}: t({tres.df:.0f})={tres.statistic:6.2f}, p={tres.pvalue:.
 ↪3f}, power={power:.3f}')
```

```
      fixed acidity: t(1596)=  3.86, p=0.000, power=0.894
   volatile acidity: t(1515)=-13.48, p=0.000, power=1.000
        citric acid: t(1593)=  6.48, p=0.000, power=1.000
     residual sugar: t(1575)= -0.09, p=0.931, power=0.005
          chlorides: t(1266)= -4.29, p=0.000, power=0.970
free sulfur dioxide: t(1523)= -2.46, p=0.014, power=0.425
total sulfur dioxide: t(1355)= -9.34, p=0.000, power=1.000
            density: t(1576)= -6.55, p=0.000, power=1.000
                 pH: t(1567)= -0.13, p=0.896, power=0.005
          sulphates: t(1495)=  8.85, p=0.000, power=1.000
            alcohol: t(1517)= 19.78, p=0.000, power=1.000
```

## 15 Exercise 10

[28]:
```
# the examples I showed in the book
smp.TTestIndPower().solve_power(effect_size=1,alpha=.05,power=.
 ↪8,nobs1=50,ratio=None)
smp.TTestIndPower().solve_power(effect_size=1,alpha=.05,power=.
 ↪8,nobs1=None,ratio=1)
```

[28]:
```
16.71472257227619
```

[29]:
```
# statistical threshold (Bonferroni-corrected)
bonP = .05/len(cols2test)

# loop over column
for col in cols2test:
  # for convenience, extract the numerical variables
  Xh = data[col][data['boolQuality']==True].values  # high rating
  Xl = data[col][data['boolQuality']==False].values # low rating
  # sample size and ratio
  nh = len(Xh)
  nl = len(Xl)
  sr = nh/nl
  # effect size (es)
  es_num = np.mean(Xh)-np.mean(Xl)
  es_den = np.sqrt( ( (nh-1)*np.var(Xh,ddof=1)+(nl-1)*np.var(Xl,ddof=1) )/
 ↪(nh+nl-2) )
  # compute power
  try:
    nl_sr = smp.TTestIndPower().solve_power(
        effect_size=es_num/es_den, alpha=bonP, power=.8, nobs1=nh, ratio=None)

    # print the results
    print(f'{col:>20}: N-high: {nh}, N-low: {int(nh*nl_sr):>3}')

  except:
    print(f'{col:>20}: ** Does not compute! **')
```

```
       fixed acidity: N-high: 855, N-low: 653
    volatile acidity: N-high: 855, N-low:  30
         citric acid: N-high: 855, N-low: 153
      residual sugar: ** Does not compute! **
           chlorides: N-high: 855, N-low: 413
 free sulfur dioxide: ** Does not compute! **
total sulfur dioxide: N-high: 855, N-low:  64
             density: N-high: 855, N-low: 153
                  pH: ** Does not compute! **
           sulphates: N-high: 855, N-low:  73
             alcohol: N-high: 855, N-low:  14
```

[30]:
```
nl_sr
```

[30]: 0.017293176626794377

## 16 Exercise 11

[31]:
```
# statistical threshold (Bonferroni-corrected)
bonP = .05/len(cols2test)

# loop over column
for col in cols2test:
  # for convenience, extract the numerical variables
  Xh = data[col][data['boolQuality']==True].values  # high rating
  Xl = data[col][data['boolQuality']==False].values # low rating

  # sample size and ratio
  nh = len(Xh)
  nl = len(Xl)
  sr = nh/nl

  # effect size (es)
  es_num = np.mean(Xh)-np.mean(Xl)
  es_den = np.sqrt( ( (nh-1)*np.var(Xh,ddof=1)+(nl-1)*np.var(Xl,ddof=1) )/
↪(nh+nl-2) )

  # compute power
  try:
    nh_r = smp.TTestIndPower().solve_power(
        effect_size=es_num/es_den, alpha=bonP, power=.8, nobs1=None, ratio=sr)

    # print the results
    print(f'{col:>20}: N-high: {int(nh_r):>7}, N-low: {int(nh_r*sr)}')

  except:
    print(f'{col:>20}: ** Does not compute! **')
```

```
       fixed acidity: N-high:     692, N-low: 796
    volatile acidity: N-high:      56, N-low: 65
         citric acid: N-high:     244, N-low: 281
      residual sugar: N-high: 1351105, N-low: 1552681
           chlorides: N-high:     521, N-low: 599
 free sulfur dioxide: N-high:    1649, N-low: 1895
total sulfur dioxide: N-high:     112, N-low: 129
             density: N-high:     244, N-low: 281
                  pH: N-high:  591943, N-low: 680257
           sulphates: N-high:     128, N-low: 147
             alcohol: N-high:      28, N-low: 33
```