CA5_mz8454

March 16, 2019

In [13]:
```
from scipy.stats import norm, t
def power_in_hypo_test_for_mean(mu_b, mu_h, n, alpha, sd, pop = True, tail = -1):
    assert n > 0
    assert sd > 0
    assert alpha > 0 and alpha < 1
    try:
        if tail not in (-1, 0, 1):
            raise ValueError
    except ValueError:
        print ('Test type indicator not found')
    else:
        if pop:
            if tail == -1:
                if mu_b < mu_h:
                    z = norm.ppf(alpha)
                    m = z*sd/n**0.5 + mu_h # critical population mean
                    score = (m - mu_b)/(sd/n**0.5)
                    p = norm.cdf(score) #probability of rejecting H0 when it is false
                    return (p)
                else:
                    return print ('Cannot find the power because H0 is true')
            elif tail == 1:
                if mu_b > mu_h:
                    z = norm.ppf(1-alpha)
                    m = z*sd/n**0.5 + mu_h # critical population mean
                    score = (m - mu_b)/(sd/n**0.5)
                    p = 1 - norm.cdf(score) #probability of rejecting H0 when it is f
                    return (p)
                else:
                    return print('Cannot find the power because H0 is true')
            else:
                if mu_b != mu_h:
                # power for lower tail
                    z1 = norm.ppf(alpha/2)
                    m1 = z1*sd/n**0.5 + mu_h # critical population mean
                    score1 = (m1 - mu_b)/(sd/n**0.5)
                    p1 = norm.cdf(score1) # probability of rejecting H0 when it is fa
                # power for upper tail
```

```
                        z2 = norm.ppf(1-alpha/2)
                        m2 = z2*sd/n**0.5 + mu_h # critical population mean
                        score2 = (m2 - mu_b)/(sd/n**0.5)
                        p2 = 1 - norm.cdf(score2) # probability of rejecting H0 when it i
                    # calculate power
                        p = p1 + p2 # probability of rejecting H0 when it is false
                        return (p)
                    else:
                        return print('Cannot find the power because H0 is true')
            else:
                if tail == -1:
                    if mu_b < mu_h:
                        t_score = t.ppf(alpha, n-1)
                        m = t_score*sd/n**0.5 + mu_h # critical sample mean
                        score = (m - mu_b)/(sd/n**0.5)
                        p = t.cdf(score, n-1) # probability of rejecting H0 when it is fa
                        return (p)
                    else:
                        return print('Cannot find the power because H0 is true')
                elif tail == 1:
                    if mu_b > mu_h:
                        t_score = t.ppf(1-alpha, n-1)
                        m = t_score*sd/n**0.5 + mu_h # critical sample mean
                        score = (m - mu_b)/(sd/n**0.5)
                        p = 1 - t.cdf(score, n-1) # probability of rejecting H0 when it i
                        return print (p)
                    else:
                        return print('Cannot find the power because H0 is true')
                else:
                    if mu_b != mu_h:
                    # test for lower tail
                        t1 = t.ppf(alpha/2, n-1)
                        m1 = t1*sd/n**0.5 + mu_h # critical sample mean
                        score1 = (m1 - mu_b)/(sd/n**0.5)
                        p1 = t.cdf(score1, n-1) # probability of rejecting H0 when it is
                    # test for upper tail
                        t2 = t.ppf(1-alpha/2, n-1)
                        m2 = t2*sd/n**0.5 + mu_h # critical sample mean
                        score2 = (m2 - mu_b)/(sd/n**0.5)
                        p2 = 1 - t.cdf(score2, n-1) # probability of rejecting H0 when it
                    # caluculate power
                        p = p1 + p2 # probability of rejecting H0 when it is false
                        return print (p)
                    else:
                        return print('Cannot find the power because H0 is true')
```

In [14]:
```
power_in_hypo_test_for_mean(23, 25, 30, 0.02, 3, True, -1)
power_in_hypo_test_for_mean(25, 25, 30, 0.02, 3, True, -1)
```

```
        power_in_hypo_test_for_mean(23, 25, 30, 0.02, 3, True, 2)
```

```
0.944948995447
Cannot find the power because H0 is true
Test type indicator not found
```

```
In [15]: power_in_hypo_test_for_mean(17, 15, 35, 0.01, 4, True, 1)
         power_in_hypo_test_for_mean(15, 15, 35, 0.01, 4, True, 1)
```

```
0.736205927435
Cannot find the power because H0 is true
```

```
In [16]: power_in_hypo_test_for_mean(76, 75, 16, 0.05, 8, True, 0)
         power_in_hypo_test_for_mean(77, 75, 16, 0.05, 8, True, 0)
         power_in_hypo_test_for_mean(75, 75, 16, 0.05, 8, True, 0)
```

```
0.0790975341606
0.170075045753
Cannot find the power because H0 is true
```

```
In [17]: power_in_hypo_test_for_mean(9950, 10000, 30, 0.05, 125, False, -1)
         power_in_hypo_test_for_mean(2.09, 2.0, 35, 0.05, 0.3, False, 1)
         power_in_hypo_test_for_mean(15.1, 15.4, 35, 0.05, 2.5, False, 0)
```

```
0.68670570785
0.533185876981
0.102277050319
```

```
In [18]: power_in_hypo_test_for_mean(119.999, 120, 36, 0.05, 12, pop=False, tail=-1)
         power_in_hypo_test_for_mean(120.001, 120, 36, 0.05, 12, pop=False, tail=1)
         power_in_hypo_test_for_mean(16.5, 16, 30, 0.05, 0.8, pop=False, tail=0)
```

```
0.0500483096257
0.0500483096257
0.910637011996
```

```
In [19]: power_in_hypo_test_for_mean(119.999, 120, 36, 0.05, 12, pop=True, tail=4)
         power_in_hypo_test_for_mean(119.999, 120, 36, 0.05, 12, pop=False, tail=4)
```

```
Test type indicator not found
Test type indicator not found
```

In [20]:
```
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline

# plot power curve for lower-tailed test with population sd
x = np.arange(110, 120, 0.02)
plt.plot(x, power_in_hypo_test_for_mean(x, 120, 36, 0.05, 12, pop=True, tail=-1), 'r--
plt.show()

# plot power curve for upper-tailed test with population sd
x = np.arange(120, 130, 0.02)
plt.plot(x, power_in_hypo_test_for_mean(x, 120, 36, 0.05, 12, pop=True, tail=1), 'r--
plt.show()

# plot power curve for two-tailed test with population sd
x = np.arange(15, 17, 0.02)
plt.plot(x, power_in_hypo_test_for_mean(x, 16, 30, 0.05, 0.8, pop=True, tail=0), 'r--
plt.show()
```

```
---------------------------------------------------------------------------

ValueError                                Traceback (most recent call last)

<ipython-input-20-acfbc2e227ca> in <module>()
  6 # plot power curve for lower-tailed test with population sd
  7 x = np.arange(110, 120, 0.02)
----> 8 plt.plot(x, power_in_hypo_test_for_mean(x, 120, 36, 0.05, 12, pop=True, tail=-1),
  9 plt.show()
 10


<ipython-input-13-a87c87d14dae> in power_in_hypo_test_for_mean(mu_b, mu_h, n, alpha, sc
 12         if pop:
 13             if tail == -1:
---> 14                 if mu_b < mu_h:
 15                     z = norm.ppf(alpha)
 16                     m = z*sd/n**0.5 + mu_h # critical population mean


ValueError: The truth value of an array with more than one element is ambiguous. Use a
```

In [21]:
```
list_x =[x for x in np.arange(110, 120, 0.02)]
list_y=[]
for x in np.arange(110, 120, 0.02):
    y = power_in_hypo_test_for_mean(x, 120, 36, 0.05, 12, pop=True, tail=-1)
```

```
            list_y.append(y)
        plt.plot(list_x,list_y)
        plt.show()
```

```
0.999603384995
0.999588804218
0.999573727446
0.999558139367
0.999542024252
0.999525365949
0.999508147868
0.999490352981
0.999471963802
0.999452962384
0.999433330308
0.99941304867
0.999392098073
0.999370458616
0.999348109885
0.999325030939
0.999301200304
0.999276595957
0.999251195318
0.999224975238
0.999197911988
0.999169981246
0.999141158089
0.999111416977
0.999080731743
0.999049075584
0.999016421042
0.998982739999
0.998948003661
0.998912182544
0.998875246467
0.998837164533
0.998797905122
0.998757435872
0.998715723674
0.998672734649
0.998628434146
0.998582786718
0.998535756117
0.998487305276
0.998437396298
0.998385990441
0.998333048104
0.998278528815
```

0.998222391217
0.998164593053
0.998105091155
0.998043841424
0.997980798825
0.997915917367
0.997849150088
0.997780449047
0.997709765307
0.997637048917
0.997562248907
0.997485313264
0.997406188928
0.997324821771
0.997241156586
0.997155137071
0.997066705821
0.996975804308
0.99688237287
0.996786350698
0.996687675823
0.9965862851
0.996482114199
0.996375097588
0.996265168521
0.996152259027
0.996036299898
0.995917220672
0.995794949625
0.995669413756
0.995540538779
0.995408249105
0.995272467838
0.995133116758
0.994990116311
0.9948433856
0.994692842374
0.994538403018
0.994379982538
0.994217494562
0.994050851317
0.993879963634
0.993704740928
0.993525091194
0.993340921002
0.993152135485
0.992958638331
0.992760331782

0.992557116622
0.992348892173
0.992135556289
0.991917005353
0.991693134269
0.99146383646
0.991229003865
0.990988526934
0.990742294627
0.990490194409
0.990232112253
0.989967932637
0.989697538543
0.989420811455
0.989137631368
0.988847876778
0.988551424696
0.988248150642
0.98793792865
0.987620631278
0.987296129604
0.986964293238
0.986624990327
0.98627808756
0.985923450178
0.985560941979
0.985190425333
0.984811761187
0.984424809077
0.984029427142
0.983625472133
0.98321279943
0.982791263052
0.982360715676
0.981921008649
0.981471992011
0.981013514503
0.980545423595
0.980067565499
0.979579785194
0.979081926444
0.978573831821
0.978055342727
0.977526299424
0.97698654105
0.976435905651
0.975874230208
0.975301350661

0.974717101943
0.974121318006
0.973513831853
0.972894475571
0.972263080362
0.97161947658
0.970963493762
0.970294960667
0.96961370531
0.968919555004
0.968212336395
0.967491875504
0.966757997767
0.966010528078
0.965249290831
0.964474109964
0.963684809005
0.962881211116
0.962063139141
0.961230415655
0.960382863012
0.959520303392
0.958642558857
0.957749451398
0.956840802991
0.955916435648
0.954976171475
0.954019832723
0.953047241848
0.952058221564
0.951052594906
0.950030185288
0.948990816557
0.947934313061
0.946860499708
0.945769202023
0.944660246221
0.94353345926
0.942388668914
0.941225703832
0.940044393607
0.938844568843
0.937626061218
0.936388703558
0.935132329898
0.933856775557
0.932561877204
0.931247472931

0.929913402319
0.928559506512
0.927185628288
0.92579161213
0.924377304297
0.922942552901
0.921487207974
0.920011121543
0.918514147706
0.916996142701
0.915456964983
0.913896475297
0.91231453675
0.910711014888
0.909085777767
0.907438696029
0.905769642976
0.904078494642
0.902365129869
0.900629430379
0.898871280848
0.897090568979
0.895287185576
0.893461024615
0.891611983316
0.889739962217
0.887844865244
0.885926599781
0.883985076743
0.882020210643
0.880031919664
0.878020125725
0.87598475455
0.873925735737
0.87184300282
0.86973649334
0.867606148907
0.865451915261
0.863273742342
0.861071584344
0.858845399783
0.856595151553
0.854320806982
0.852022337898
0.849699720677
0.847352936303
0.844981970419
0.842586813382

0.840167460314
0.837723911149
0.835256170686
0.832764248632
0.830248159648
0.827707923395
0.825143564575
0.822555112972
0.819942603489
0.81730607619
0.814645576329
0.81196115439
0.809252866117
0.806520772542
0.803764940016
0.800985440234
0.798182350262
0.795355752557
0.792505734987
0.789632390853
0.786735818903
0.783816123347
0.780873413871
0.777907805646
0.774919419334
0.771908381101
0.768874822613
0.765818881043
0.762740699068
0.75964042487
0.756518212124
0.753374219999
0.750208613146
0.747021561681
0.743813241179
0.740583832654
0.737333522542
0.734062502676
0.730770970269
0.727459127885
0.724127183412
0.720775350033
0.717403846192
0.714012895561
0.710602727002
0.707173574529
0.703725677264
0.700259279399

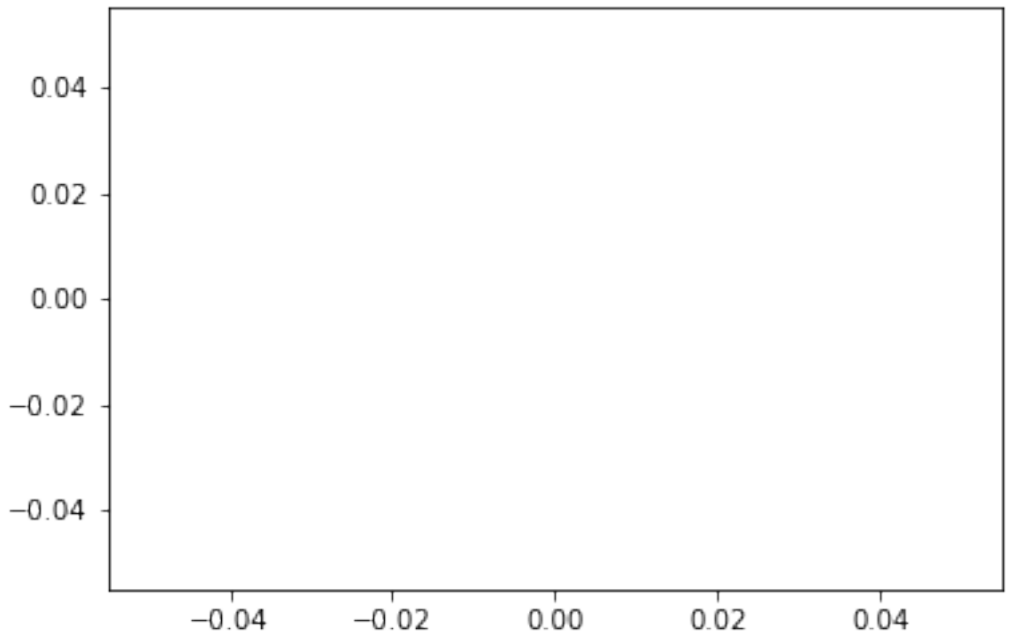
0.696774630141
0.693271983673
0.689751599097
0.686213740384
0.68265867632
0.679086680446
0.675498030999
0.671893010854
0.668271907456
0.664635012757
0.660982623146
0.657315039382
0.653632566518
0.649935513831
0.646224194743
0.642498926744
0.638760031313
0.635007833834
0.631242663515
0.6274648533
0.623674739784
0.619872663122
0.616058966938
0.612233998235
0.608398107299
0.604551647603
0.600694975711
0.596828451179
0.592952436453
0.589067296771
0.585173400057
0.581271116817
0.577360820033
0.573442885061
0.569517689516
0.565585613166
0.561647037824
0.557702347234
0.553751926959
0.549796164269
0.545835448028
0.541870168578
0.537900717622
0.533927488111
0.529950874128
0.525971270765
0.521989074011
0.518004680633

0.514018488055
0.510030894241
0.506042297573
0.502053096737
0.498063690599
0.494074478084
0.490085858064
0.486098229229
0.482111989973
0.478127538275
0.474145271575
0.470165586661
0.466188879546
0.462215545352
0.458245978191
0.454280571049
0.450319715669
0.446363802433
0.44241322025
0.438468356439
0.434529596614
0.430597324574
0.426671922189
0.42275376929
0.418843243557
0.414940720412
0.411046572909
0.407161171628
0.40328488457
0.399418077052
0.395561111605
0.391714347869
0.387878142499
0.384052849059
0.38023881793
0.376436396212
0.372645927629
0.36886775244
0.365102207344
0.361349625394
0.357610335909
0.353884664385
0.350172932417
0.346475457613
0.342792553516
0.339124529524
0.335471690816
0.331834338276

0.328212768424
0.324607273342
0.321018140607
0.31744565323
0.313890089584
0.310351723349
0.30683082345
0.303327654
0.299842474246
0.296375538512
0.292927096156
0.289497391514
0.286086663858
0.282695147352
0.279323071011
0.275970658657
0.27263812889
0.269325695045
0.266033565167
0.262761941976
0.259511022842
0.256280999756
0.253072059312
0.249884382683
0.246718145603
0.243573518349
0.240450665733
0.237349747081
0.234270916233
0.23121432153
0.22818010581
0.225168406407
0.22217935515
0.219213078364
0.216269696873
0.213349326008
0.210452075616
0.207578050067
0.204727348269
0.201900063681
0.199096284332
0.196316092835
0.193559566415
0.190826776922
0.188117790865
0.185432669431
0.182771468518
0.180134238762

0.177521025573
0.174931869164
0.172366804589
0.169825861782
0.167309065592
0.164816435828
0.162347987296
0.159903729849
0.157483668427
0.155087803107
0.15271612915
0.150368637052
0.148045312595
0.145746136897
0.143471086469
0.14122013327
0.13899324476
0.136790383963
0.134611509519
0.132456575751
0.130325532721
0.128218326292
0.126134898194
0.124075186084
0.122039123615
0.120026640498
0.11803766257
0.116072111861
0.114129906661
0.11221096159
0.110315187667
0.108442492379
0.106592779753
0.104765950423
0.102961901706
0.101180527674
0.0994217192196
0.0976853641373
0.0959713471905
0.0942795501867
0.0926098520507
0.0909621288983
0.0893362541098
0.0877320984041
0.0861495299125
0.0845884142527
0.0830486146028
0.0815299917753

```
0.0800324042909
0.0785557084523
0.0770997584181
0.0756644062757
0.074249502115
0.0728548941015
0.0714804285484
0.0701259499897
0.0687913012521
0.0674763235264
0.0661808564394
0.064904738124
0.0636478052905
0.0624098932959
0.0611908362135
0.0599904669019
0.0588086170732
0.0576451173608
0.0564997973865
0.0553724858268
0.054263010479
0.0531711983261
0.0520968756011
0.0510398678512
```



```
In [8]: from scipy.stats import norm, t
        import numpy as np
        import matplotlib.pyplot as plt

        %matplotlib inline

In [12]: for x in np.arange(110, 120, 0.02):
             plt.plot(x, power_in_hypo_test_for_mean(x, 120, 36, 0.05, 12, pop=True, tail=-1),
             plt.show()

0.999603384995


        ---------------------------------------------------------------------------

        ValueError                                Traceback (most recent call last)

        <ipython-input-12-d4731bf28c36> in <module>()
          1 for x in np.arange(110, 120, 0.02):
        ----> 2     plt.plot(x, power_in_hypo_test_for_mean(x, 120, 36, 0.05, 12, pop=True, tail=-
          3     plt.show()


        C:\Users\SEAN  PHAN\Anaconda3\lib\site-packages\matplotlib\pyplot.py in plot(*args, **
       3316                      mplDeprecation)
       3317     try:
    -> 3318         ret = ax.plot(*args, **kwargs)
       3319     finally:
       3320         ax._hold = washold


        C:\Users\SEAN  PHAN\Anaconda3\lib\site-packages\matplotlib\__init__.py in inner(ax, *a
       1889                     warnings.warn(msg % (label_namer, func.__name__),
       1890                                   RuntimeWarning, stacklevel=2)
    -> 1891             return func(ax, *args, **kwargs)
       1892         pre_doc = inner.__doc__
       1893         if pre_doc is None:


        C:\Users\SEAN  PHAN\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py in plot(self,
       1404         kwargs = cbook.normalize_kwargs(kwargs, _alias_map)
       1405
    -> 1406         for line in self._get_lines(*args, **kwargs):
       1407             self.add_line(line)
       1408             lines.append(line)
```

```
C:\Users\SEAN  PHAN\Anaconda3\lib\site-packages\matplotlib\axes\_base.py in _grab_next_
405                 return
406             if len(remaining) <= 3:
--> 407                 for seg in self._plot_args(remaining, kwargs):
408                     yield seg
409                 return


C:\Users\SEAN  PHAN\Anaconda3\lib\site-packages\matplotlib\axes\_base.py in _plot_args
366         # downstream.
367         if any(v is None for v in tup):
--> 368             raise ValueError("x and y must not be None")
369
370         kw = {}


ValueError: x and y must not be None
```

In [ ]: