Problem 7

In [1]:
```
import numpy as np
import scipy as sp
```

In [2]:
```
# 7(a) calculate Wilkinson's polynomial coefficients

N = 20 #number of terms
p = np.array([1.,-1.]) #k =1 term
for n in range(2, N+1): #construct iteratively: p(N+1)=x*P(N)-(N+1)P
(N)
    p = np.append(p,[0]) - np.append([0], p*n)

print('the coefficients are: \n {} \n'.format(p))
```

```
the coefficients are:
 [ 1.00000000e+00 -2.10000000e+02  2.06150000e+04 -1.25685000e+06
  5.33279460e+07 -1.67228082e+09  4.01717716e+10 -7.56111184e+11
  1.13102770e+13 -1.35585183e+14  1.30753501e+15 -1.01422999e+16
  6.30308121e+16 -3.11333643e+17  1.20664780e+18 -3.59997952e+18
  8.03781182e+18 -1.28709312e+19  1.38037598e+19 -8.75294804e+18
  2.43290201e+18]
```

In [3]:
```
#7(b)  evaluate and find roots

def Wpoly(x):#write a function to evaluate w(x) given x
    return np.polyval(p, x)

ppoly=np.polynomial.Polynomial(p[::-1])#,domain=[0, 22], window=[0,2
2]) #use as a polynomial object explicity

def Wpoly2(x):#same as above, uses polynomial object expilitly, doesn
t change anything
    return np.polynomial.polynomial.polyval(x, ppoly.coef)


root20N=sp.optimize.newton(Wpoly, 21.)
# root20N2=sp.optimize.newton(Wpoly2, 21.)

roots_polyroot=ppoly.roots()

print('Newtowns method from x_0=21 finds root: {}\n'.format(root20N))

print('Numpy polynomial built-in method finds roots:\n {}\n'.format(r
oots_polyroot))

print('Note that when evaluate at roots W(1.:20.)=:\n {}\n'.format(Wp
oly(np.linspace(1., 20., 20))))
```

```
Newtowns method from x_0=21 finds root: 19.999995981138753

Numpy polynomial built-in method finds roots:
 [ 1.        +0.j          2.        +0.j          2.99999999+0.j
  4.0000002 +0.j          4.9999963 +0.j          6.0000482 +0.j
  6.99956009+0.j          8.00287806+0.j          8.98673525+0.j
 10.04985911+0.j         10.88618474+0.j         12.35783538+0.j
 12.56199912+0.j         14.51893762-0.21308755j 14.51893762+0.213087
55j
 16.20675028+0.j         16.8857393 +0.j         18.03009401+0.j
 18.99390267+0.j         20.00054206+0.j        ]

Note that when evaluate at roots W(1.:20.)=:
 [ 1.02400000e+03  8.19200000e+03  5.52960000e+04 -3.60448000e+05
 -1.53600000e+06 -9.51091200e+06 -2.10739200e+07 -5.47880960e+07
 -1.23918336e+08 -8.90880000e+07 -5.45177600e+08 -8.53770240e+08
 -1.35883571e+09 -1.93599078e+09 -3.76012800e+09 -5.96220314e+09
 -9.53860915e+09 -2.35116380e+10 -1.60700334e+10 -2.70295040e+10]
```

As can be seen both method is able to find the root at 20, and the scipy newtons method does slightly better. This did not work if stored values of polynomial coefficients as integers, in that case only found root at $x = 1$. Note however that when directly evaluating $w(x)$ at the 20 roots we get junk.

In [4]:
```
# 7(c)
#note that a20==
deltas=[1.e-8, 1.e-6, 1.e-4, 1.e-2]
root20deltas=[]
for delta in deltas:
    ptemp=p.copy()
    ptemp[0]=p[0]+delta
#     print(ptemp[0])
    root20deltas.append(sp.optimize.newton(lambda x:np.polyval(ptemp,
x), 21., maxiter=500))
    print('\n delta={} results in largest root:  {}'.format(delta, ro
ot20deltas[-1]))
    temproots=np.polynomial.Polynomial(ptemp[::-1]).roots()
    print('and full set of roots from .roots() :\n {}\n'.format(tempr
oots))
```

```
 delta=1e-08 results in largest root:  9.585097719672847
and full set of roots from .roots() :
 [ 1.        +0.j          2.        +0.j          3.        +0.j
  3.99999993+0.j          5.00000167+0.j          5.99997626+0.j
  7.00033047+0.j          7.99455793+0.j          9.10180119+0.j
  9.58140435+0.j         10.88054718-1.10962138j 10.88054718+1.109621
38j
 12.75247322-2.1276999j  12.75247322+2.1276999j  15.20890593-2.879942
7j
 15.20890593+2.8799427j  18.17150122-2.76904678j 18.17150122+2.769046
78j
 20.6475355 -1.18691242j 20.6475355 +1.18691242j]


 delta=1e-06 results in largest root:  7.752698116492445
and full set of roots from .roots() :
 [ 1.        +0.j          2.        +0.j          3.00000001+0.j
  3.99999979+0.j          5.0000061 +0.j          5.99962248+0.j
  7.01928406+0.j          7.75217851+0.j          8.6417379 -1.007536
9j
  8.6417379 +1.0075369j   9.99509892-2.28806974j  9.99509892+2.288069
74j
 11.86552541-3.7475366j  11.86552541+3.7475366j  14.65897489-5.150236
4j
 14.65897489+5.1502364j  18.8039955 -5.53148348j 18.8039955 +5.531483
48j
 23.1490169 -2.7409845j  23.1490169 +2.7409845j ]


 delta=0.0001 results in largest root:  5.969334704221419
and full set of roots from .roots() :
 [ 1.         +0.j          2.         +0.j
  3.         +0.j          3.9999992  +0.j
  5.00030297 +0.j          5.96934721 +0.j
  6.81227031 -0.59000689j  6.81227031 +0.59000689j
  7.70572566 -1.80906046j  7.70572566 +1.80906046j
  8.84585096 -3.35922838j  8.84585096 +3.35922838j
 10.49578215 -5.35388913j 10.49578215 +5.35388913j
 13.31450056 -7.85161055j 13.31450056 +7.85161055j
 18.93033431-10.07295983j 18.93033431+10.07295983j
 28.40021241 -6.51043422j 28.40021241 +6.51043422j]


 delta=0.01 results in largest root:  5.469592828289074
and full set of roots from .roots() :
 [ 1.         +0.j          2.         +0.j
  3.00000004 +0.j          3.99991269 +0.j
  5.03691842 +0.j          5.46959692 +0.j
  5.98137621 -1.11660899j  5.98137621 +1.11660899j
  6.64729367 -2.43830385j  6.64729367 +2.43830385j
  7.44596883 -4.19140464j  7.44596883 +4.19140464j
  8.56227534 -6.65957602j  8.56227534 +6.65957602j
 10.5819106 -10.4503631j  10.5819106 +10.4503631j
 16.01017374-16.82880772j 16.01017374+16.82880772j
 38.47818362-20.83432359j 38.47818362+20.83432359j]
```

As shown above, even for $\delta = 10^{-8}$ the location of the roots has changed dramatically, with the largest real root being at 9.6, and decreasing with increased δ. Note that as increase δ, the number of complex roots also increases, and only the lowest roots remain largely unchanged.

In [5]:
```
# 7(d)
p19t=p.copy()
p19t[1]=p[1]-2.**(-23)
root19t=sp.optimize.newton(lambda x:np.polyval(p19t, x), 21., maxiter
=500)
print('\n a_19-2^-23 results in largest root:  {}'.format(root19t))
roots19t=np.polynomial.Polynomial(p19t[::-1]).roots()
print('and full set of roots from .roots() :\n {}\n'.format(roots19t
))
print('Root 16 is {} and Root 17 is {}'.format(roots19t[15], roots19t
[16]))
```

```
 a_19-2^-23 results in largest root:  20.846906953184412
and full set of roots from .roots() :
 [ 1.        +0.j          2.        +0.j          3.        +0.j
  4.        +0.j          5.00000001+0.j          6.00000747+0.j
  6.99966203+0.j          8.0077659 +0.j          8.91462159+0.j
 10.09442575-0.64843329j 10.09442575+0.64843329j 11.79460219-1.654264
81j
 11.79460219+1.65426481j 13.99301795-2.51922167j 13.99301795+2.519221
67j
 16.73096403-2.81265595j 16.73096403+2.81265595j 19.50249798-1.940329
51j
 19.50249798+1.94032951j 20.8469273 +0.j        ]

Root 16 is (16.730964025746452-2.8126559537194207j) and Root 17 is (1
6.730964025746452+2.8126559537194207j)
```

Roots 16 and 17 are now complex conjugates and also not at 16 and 17 anymore. They have formed a double root.

7e(i) is on the attatched paper, (ii) and (iii) below

In [6]:
```
# 7 e ii


ppoly_prime=ppoly.deriv()


conds=[]
roots=[14, 16, 17, 20]
for root in roots:
    tempcond = 0
    for i in range(20):
        tempcond += np.abs(ppoly.coef[i]*root**i) #do sum

    tempcond=tempcond/np.abs(root * ppoly_prime(root)) #divide by pre
factor
    conds.append(tempcond)
    print('Condition number for root {} is {:e}'.format(root, tempcon
d))
```

```
Condition number for root 14 is 5.401038e+13
Condition number for root 16 is 3.543800e+13
Condition number for root 17 is 1.812051e+13
Condition number for root 20 is 1.378037e+11
```

(ii) as shown above, the condition numbers are very high for all the calculated roots. It is smallest for 20 since this will give the largest derivative and thus the largest denominator. This confirms are intuition seen earlier, since infintesimally small changes to the coefficients resulted in dramatic changes to the roots, which is what a large condition number indicates

(iii) This problem is poorly conditioned, so no algorithm, no matter how clever, can help us