AlgorithmComparison

November 16, 2020

1 Comparison of algorithms

In [131]:
```
from __future__ import print_function
import numpy as np
import matplotlib.pyplot as plt
import math
import scipy.interpolate
import os
%matplotlib inline
plt.rcParams['figure.figsize'] = (15, 8)

### for Palatino and other serif fonts use:
#plt.rcParams.update({
#    "text.usetex": True,
#    "font.family": "serif",
#    "font.serif": ["Palatino"],
#})

# only with Python2.7 !!
import sys
sys.path.insert(0,'/usr/lib/python2.7/pyobs-master/')
from pyobs import *
```

In [340]:
```
def round_on_error(value, error):
    significant_digits = 1-int(math.floor(math.log(error, 10)))
    return round(value,significant_digits)

def get_2_significant(value):
    return round(value,1-int(math.floor(math.log(value, 10))))

def get_2_significant_0(value):
    return int(round(10.0/(10.0**int(math.floor(math.log(value, 10))))*value,0))

def get_position_sign(error):
    return 1-int(math.floor(math.log(error, 10)))

def printwe(value,error):
```

```
            e = get_2_significant(error)
            a = get_position_sign(e)
            e = get_2_significant_0(error)
            stri = '{:.' + str(a) + 'f}({})'
            print(stri.format(value,e))
            #parece qeu solo funciona si el error es menor que 1. Mirar la Q de doubaltw. Co

        #def printwe2(value,error):
        #    e = get_2_significant(error)
        #    v = round_on_error(value,e)
        #    e = get_2_significant_0(error)
        #    print(v,'(',e,')', sep='')
```

```
In [351]: -int(math.floor(math.log(0.5, 10)))
```

```
Out[351]: 1
```

1.1 Plaquette

```
In [352]: P_hmc = np.loadtxt("plaq-hmc.data", skiprows=500, usecols=2);
          P_altwinding = np.loadtxt("plaq-altwindinghmc.data", skiprows=500, usecols=2);
          P_altinstanton = np.loadtxt("plaq-altinstantonhmc.data", skiprows=500, usecols=2);
          P_doublewinding  = np.loadtxt("plaq-doublealtwinding.data", skiprows=500, usecols=2)
          #P_altw2  = np.loadtxt("plaq-altw3.data", skiprows=500, usecols=2);
```

```
In [8]: print("Configuraciones HMC: {}".format(len(P_hmc)))
        print("Configuraciones altwinding: {}".format(len(P_altwinding)))
        print("Configuraciones altinstanton: {}".format(len(P_altinstanton)))
        print("Configuraciones doublewinding: {}".format(len(P_doublewinding)))
```

```
Configuraciones HMC: 111738
Configuraciones altwinding: 32677
Configuraciones altinstanton: 15458
Configuraciones doublewinding: 28476
```

Algorithm	Statistics	P	τ_{int}	$\tau_{\text{int}}/\tau_{\text{int}}^{(\text{HMC})}$
HMC	{{len(P_hmc)}}	{{printwe(Phmc,ePhmc)}}	{{printwe(tauPhmc, etauPhmc)}}	{{printwe(tauR_Pdoublewinding, etauR_Pdoublewinding)}}
Alt Winding	{{len(P_altwinding)}}	{{printwe(Paltwinding,ePaltwinding)}}	{{printwe(tauPaltwinding, etauPaltwinding)}}	{{printwe(tauR_Paltwinding, etauR_Paltwinding)}}
Alt Instanton	{{len(P_altinstanton)}}	{{printwe(Paltinstanton,ePaltinstanton)}}	{{printwe(tauPaltinstanton, etauPaltinstanton)}}	{{printwe(tauR_Paltinstanton, etauR_Paltinstanton)}}
Alt Double Winding	{{len(P_doublewinding)}}	{{printwe(Pdoublewinding,ePdoublewinding)}}	{{printwe(tauPdoublewinding, etauPdoublewinding)}}	{{printwe(tauR_Pdoublewinding, etauR_Pdoublewinding)}}

```
In [366]: a = 2.5
```

1.1.1 HMC

$P = \{\{\text{printwe(Phmc, ePhmc)}\}\}$
$\tau_{int,P} = \{\{\text{printwe(tauPhmc, etauPhmc)}\}\}$

```
In [192]: P_hmc = P_hmc[:]
          MCtime_for_P_hmc = np.arange(1, len(P_hmc)+1, 1)
          corr_phmc = observa()
          einfo = errinfo()
          einfo.addEnsemble(0,Stau=1.0)
          corr_phmc.primary_observable(0,'Plaquette $P(t=0)$', [0], ['R0'], [MCtime_for_P_hmc.t

          [Phmc, ePhmc]= corr_phmc.vwerr(errinfo=einfo)
          [tauPhmc, etauPhmc] = corr_phmc.tauint()
          tauPhmc = tauPhmc[0][0][0]
          etauPhmc = etauPhmc[0][0][0]

          print(corr_phmc.vwerr(plot=False,errinfo=einfo))

          printwe(Phmc,ePhmc)
```

```
[0.6700214468636829, 4.946715688873867e-06]
0.6700214(49)
```

1.1.2 Alt winding

$P = \{\{\text{printwe(Paltwinding, ePaltwinding)}\}\}$
$\tau_{int,P} = \{\{\text{printwe(tauPaltwinding, etauPaltwinding)}\}\}$
$\tau_{int}/\tau_{int}^{(HMC)} = \{\{\text{printwe(tauR\_Paltwinding, etauR\_Paltwinding)}\}\}$

```
In [220]: P_altwinding = P_altwinding[:]
          MCtime_for_P_altwinding = np.arange(1, len(P_altwinding)+1, 1)
          corr_paltwinding = observa()
          einfo = errinfo()
          einfo.addEnsemble(0,Stau=1.0)
          corr_paltwinding.primary_observable(0,'Plaquette $P(t=0)$', [0], ['R0'], [MCtime_for_

          [Paltwinding, ePaltwinding]= corr_paltwinding.vwerr(errinfo=einfo)
          [tauPaltwinding, etauPaltwinding] = corr_paltwinding.tauint()
          tauPaltwinding = tauPaltwinding[0][0][0]
          etauPaltwinding = etauPaltwinding[0][0][0]

          print(corr_paltwinding.vwerr(plot=False,errinfo=einfo))

          #printwe(Paltwinding,ePaltwinding)

          tauR_Paltwinding = tauPaltwinding/tauPhmc
          etauR_Paltwinding = tauR_Paltwinding *  np.sqrt( (etauPhmc/tauPhmc)**2.0 + (etauPaltw
```

```
[0.670017460679774, 8.502901030588478e-06]
```

1.1.3 Alt instanton

$P =$ {{printwe(Paltinstanton, ePaltinstanton)}}
$\tau_{int,P} =$ {{printwe(tauPaltinstanton, etauPaltinstanton)}}
$\tau_{int}/\tau_{int}^{(HMC)} =$ {{printwe(tauR_Paltinstanton, etauR_Paltinstanton)}}

```
In [224]: P_altinstanton = P_altinstanton[:]
          MCtime_for_P_altinstanton = np.arange(1, len(P_altinstanton)+1, 1)
          corr_paltinstanton = observa()
          einfo = errinfo()
          einfo.addEnsemble(0,Stau=1.0)
          corr_paltinstanton.primary_observable(0,'Plaquette $P(t=0)$', [0], ['R0'], [MCtime_fo

          [Paltinstanton, ePaltinstanton]= corr_paltinstanton.vwerr(errinfo=einfo)
          [tauPaltinstanton, etauPaltinstanton] = corr_paltinstanton.tauint()
          tauPaltinstanton = tauPaltinstanton[0][0][0]
          etauPaltinstanton = etauPaltinstanton[0][0][0]

          print(corr_paltinstanton.vwerr(plot=False,errinfo=einfo))

          #printwe(Paltinstanton,ePaltinstanton)

          tauR_Paltinstanton = tauPaltinstanton/tauPhmc
          etauR_Paltinstanton = tauR_Paltinstanton *  np.sqrt( (etauPhmc/tauPhmc)**2.0 + (etaul
```

```
[0.6700294241738907, 1.2334126716496382e-05]
```

1.1.4 Double winding

$P =$ {{printwe(Pdoublewinding, ePdoublewinding)}}
$\tau_{int,P} =$ {{printwe(tauPdoublewinding, etauPdoublewinding)}}
$\tau_{int}/\tau_{int}^{(HMC)} =$ {{printwe(tauR_Pdoublewinding, etauR_Pdoublewinding)}}

```
In [225]: P_doublewinding = P_doublewinding[:]
          MCtime_for_P_doublewinding = np.arange(1, len(P_doublewinding)+1, 1)
          corr_pdoublewinding = observa()
          einfo = errinfo()
          einfo.addEnsemble(0,Stau=1.0)
          corr_pdoublewinding.primary_observable(0,'Plaquette $P(t=0)$', [0], ['R0'], [MCtime_f

          [Pdoublewinding, ePdoublewinding]= corr_pdoublewinding.vwerr(errinfo=einfo)
          [tauPdoublewinding, etauPdoublewinding] = corr_pdoublewinding.tauint()
          tauPdoublewinding = tauPdoublewinding[0][0][0]
          etauPdoublewinding = etauPdoublewinding[0][0][0]
```

```
print(corr_pdoublewinding.vwerr(plot=False,errinfo=einfo))

#printwe(Pdoublewinding,ePdoublewinding)

tauR_Pdoublewinding = tauPdoublewinding/tauPhmc
etauR_Pdoublewinding = tauR_Pdoublewinding *  np.sqrt( (etauPhmc/tauPhmc)**2.0 + (eta
```

```
[0.6700272023440166, 9.399314260470143e-06]
```

1.2 t_0

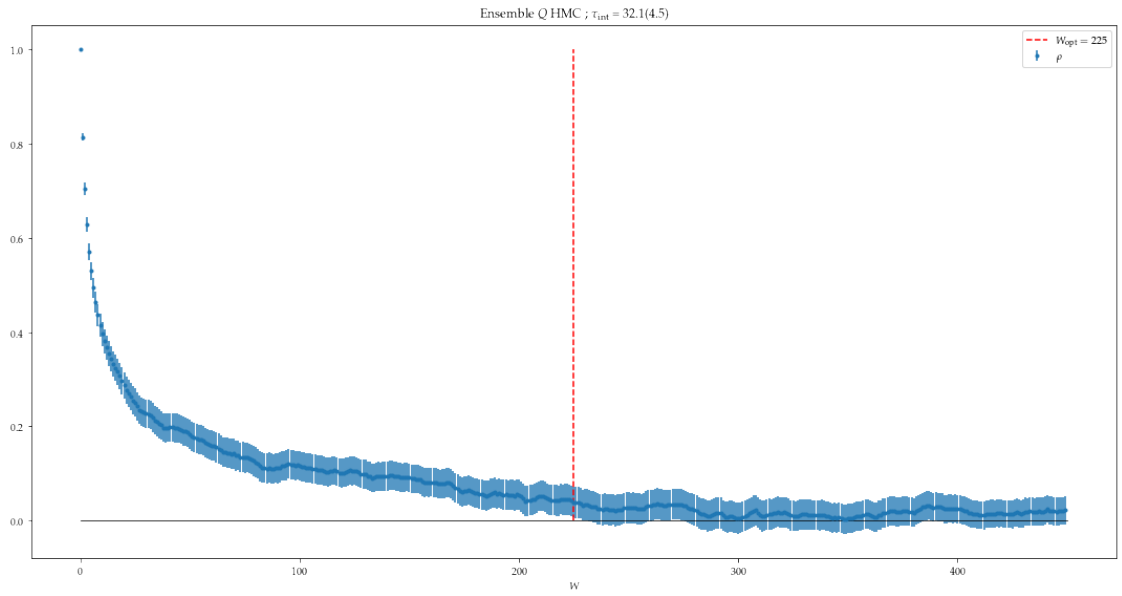
```
In [43]: t0hmc = np.loadtxt("t0hmc.txt")
         MCtimeHMC = np.arange(1, len(t0hmc)+1, 1)

         t0altw = np.loadtxt("t0altw.txt")
         MCtimealtw = np.arange(1, len(t0altw)+1, 1)

         t0alti = np.loadtxt("t0alti.txt")
         MCtimealti = np.arange(1, len(t0alti)+1, 1)

         t0doubaltw = np.loadtxt("t0doubaltw.txt")
         MCtimedoubaltw = np.arange(1, len(t0doubaltw)+1, 1)
```

1.2.1 HMC

```
In [42]: len(t0altw)
```

```
Out[42]: 12000
```

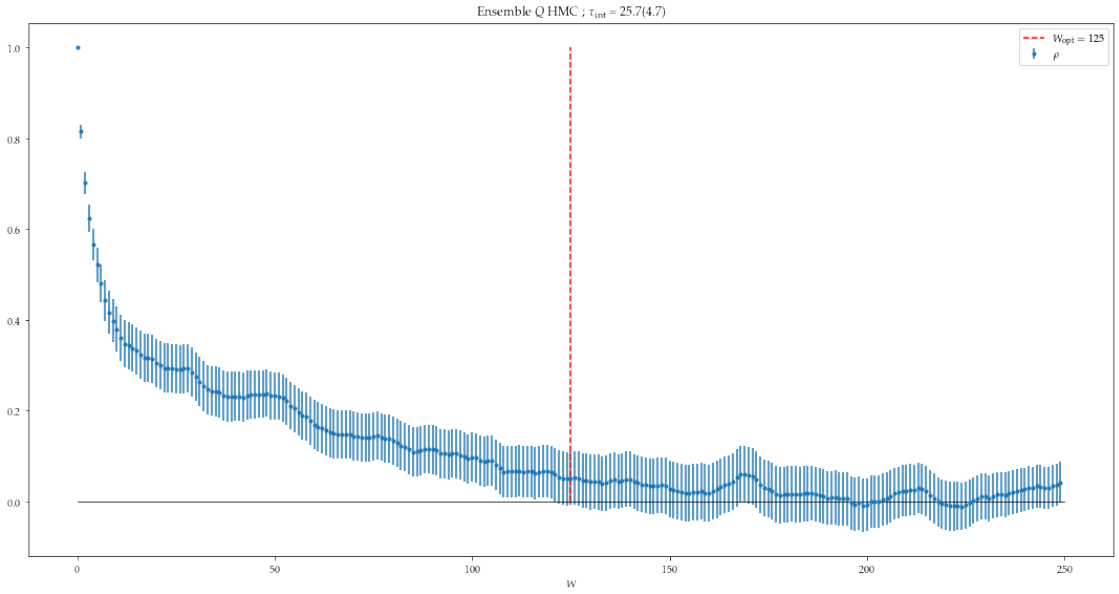
```
In [35]: corr = observa()
         einfo = errinfo()
         einfo.addEnsemble(0,Stau=1.0, W=225)
         corr.primary_observable(0,'$Q$ HMC', [0], ['R0'], [MCtimeHMC.tolist()], [(t0hmc).tolis
         [qhmc, eqhmc] = corr.vwerr(errinfo=einfo, )

         print(corr.vwerr(plot=True,errinfo=einfo))
```




```
[0.7320578378315383, 0.0021742537553067227]
```

1.2.2 Alt Winding

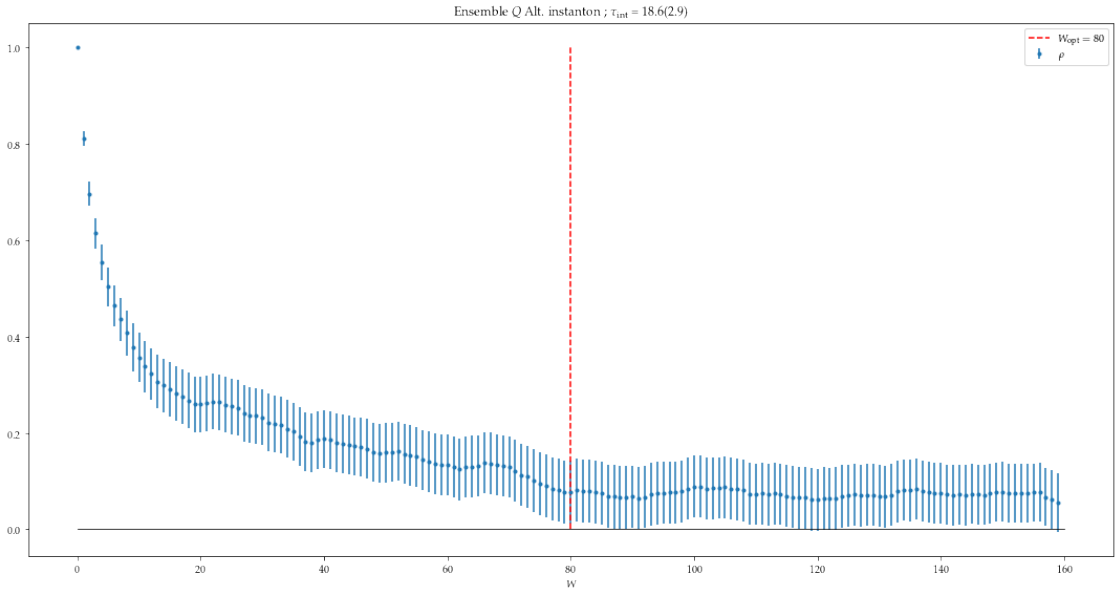
```
In [40]: corr = observa()
         einfo = errinfo()
         einfo.addEnsemble(0,Stau=1.0, W=125)
         corr.primary_observable(0,'$Q$ HMC', [0], ['R0'], [MCtimealtw.tolist()], [(t0altw).to
         [qaltw, eqaltw] = corr.vwerr(errinfo=einfo, )

         print(corr.vwerr(plot=True,errinfo=einfo))
```




```
[0.7416314018029203, 0.00376747528945022]
```

1.2.3 Alt instanton

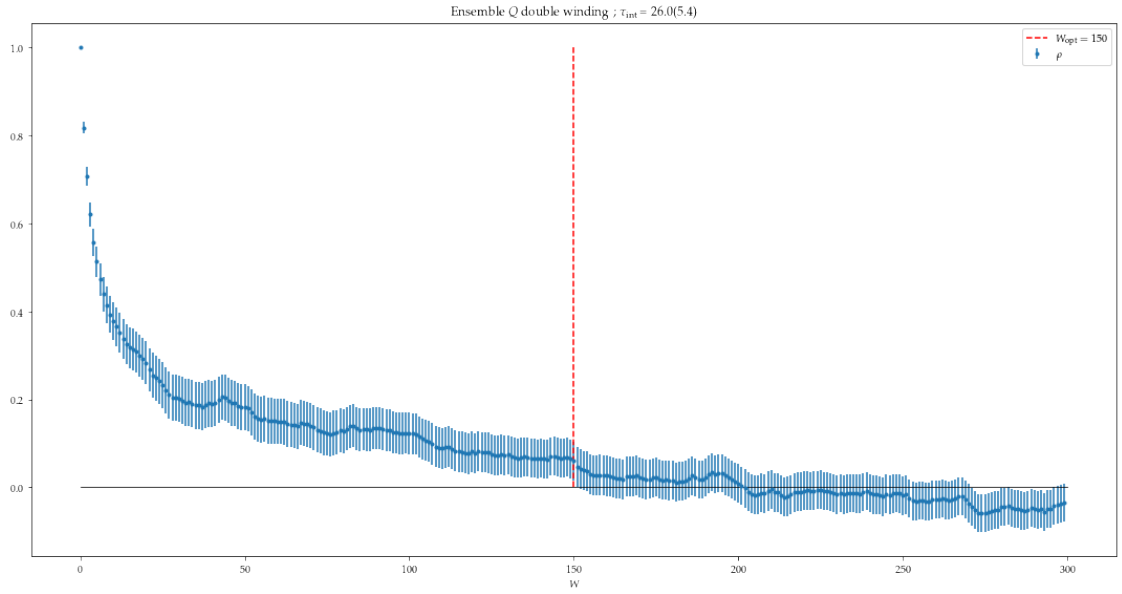
```
In [53]: corr = observa()
         einfo = errinfo()
         einfo.addEnsemble(0,Stau=1.0, W=80)
         corr.primary_observable(0,'$Q$ Alt. instanton', [0], ['R0'], [MCtimealti.tolist()], [
         [qalti, eqalti] = corr.vwerr(errinfo=einfo, )

         print(corr.vwerr(plot=True,errinfo=einfo))
```




```
[0.7384171182115985, 0.00343629500885052]
```

1.2.4 Double winding

```
In [50]: corr = observa()
         einfo = errinfo()
         einfo.addEnsemble(0,Stau=1.0, W=150)
         corr.primary_observable(0,'$Q$ double winding', [0], ['R0'], [MCtimedoubaltw.tolist()]
         [qdoubaltw, eqdoubaltw] = corr.vwerr(errinfo=einfo, )

         print(corr.vwerr(plot=True,errinfo=einfo))
```



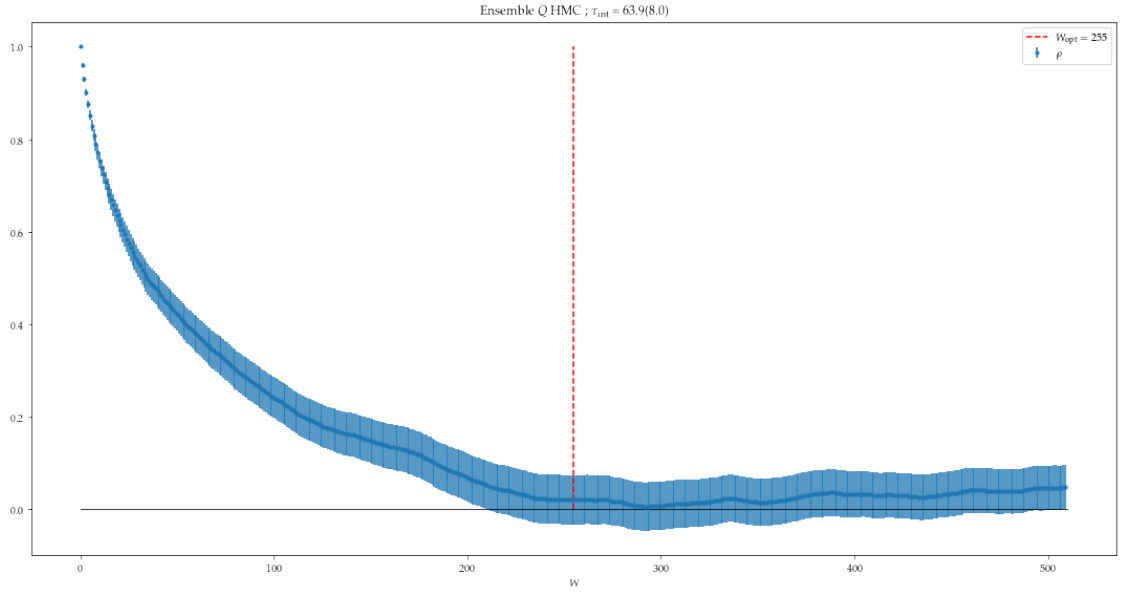

```
[0.7355934847022595, 0.003750142107685617]
```

1.3 t^2E at $t = 3$

```
In [79]: t2Eat3hmc = np.loadtxt("t2Eat3hmc.txt")
         t2Eat3hmc = t2Eat3hmc[:49000]
         MCtimeHMC = np.arange(1, len(t2Eat6hmc)+1, 1)

         t2Eat3altw = np.loadtxt("t2Eat3altw.txt")
         t2Eat3altw = t2Eat3altw[:29000]
         MCtimealtw = np.arange(1, len(t2Eat6altw)+1, 1)

         #t2Eat6alti = np.loadtxt("t2Eat6alti.txt")
         #MCtimealti = np.arange(1, len(t2Eat6alti)+1, 1)
         #
         #t2Eat6doubaltw = np.loadtxt("t2Eat6doubaltw.txt")
         #MCtimedoubaltw = np.arange(1, len(t2Eat6doubaltw)+1, 1)

         Nhmc = len(t2Eat6hmc)
         Naltw = len(t2Eat6altw)
         Ndoubaltw = len(t2Eat6doubaltw)
         Nalti = len(t2Eat6alti)

In [76]: Nhmc

Out[76]: 49000
```

1.3.1 HMC

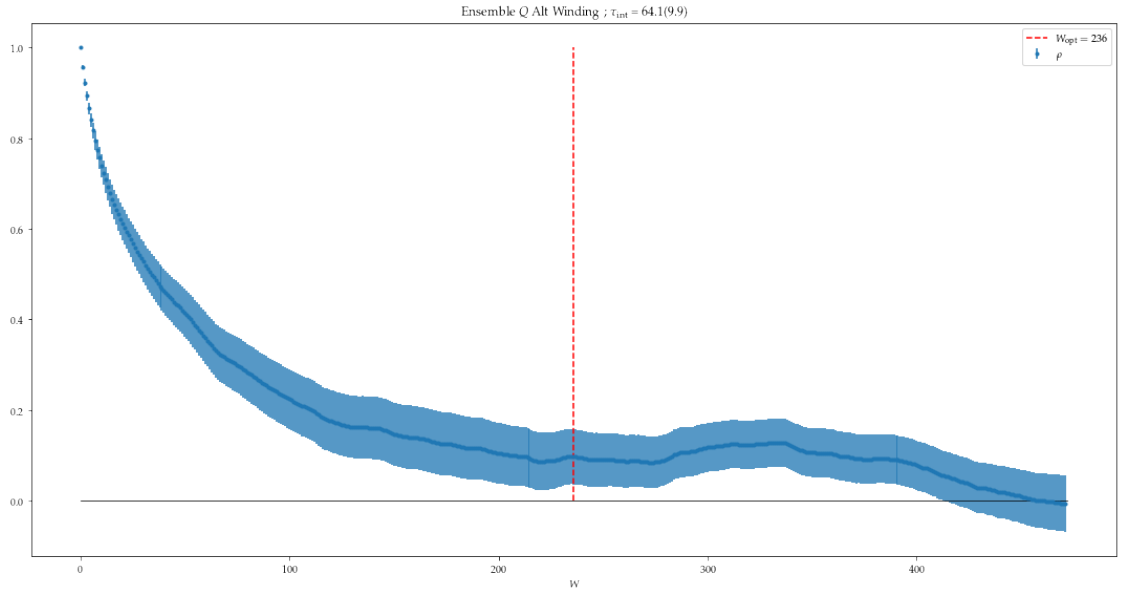
```
In [80]: corr = observa()
         einfo = errinfo()
         einfo.addEnsemble(0,Stau=1.0)
         corr.primary_observable(0,'$Q$ HMC', [0], ['RO'], [MCtimeHMC.tolist()], [(t2Eat3hmc).t
         [qhmc, eqhmc] = corr.vwerr(errinfo=einfo, )

         print(corr.vwerr(plot=True,errinfo=einfo))
```




```
[0.0860729033160204, 0.0006674214615664195]
```

1.3.2 Alt Winding

```
In [81]: corr = observa()
         einfo = errinfo()
         einfo.addEnsemble(0,Stau=1.0)
         corr.primary_observable(0,'$Q$ Alt Winding', [0], ['RO'], [MCtimealtw.tolist()], [(t2
         [qaltw, eqaltw] = corr.vwerr(errinfo=einfo, )

         print(corr.vwerr(plot=True,errinfo=einfo))
```




```
[0.0832572622986207, 0.0008308000737363403]
```

1.4 t^2E at $t = 6$

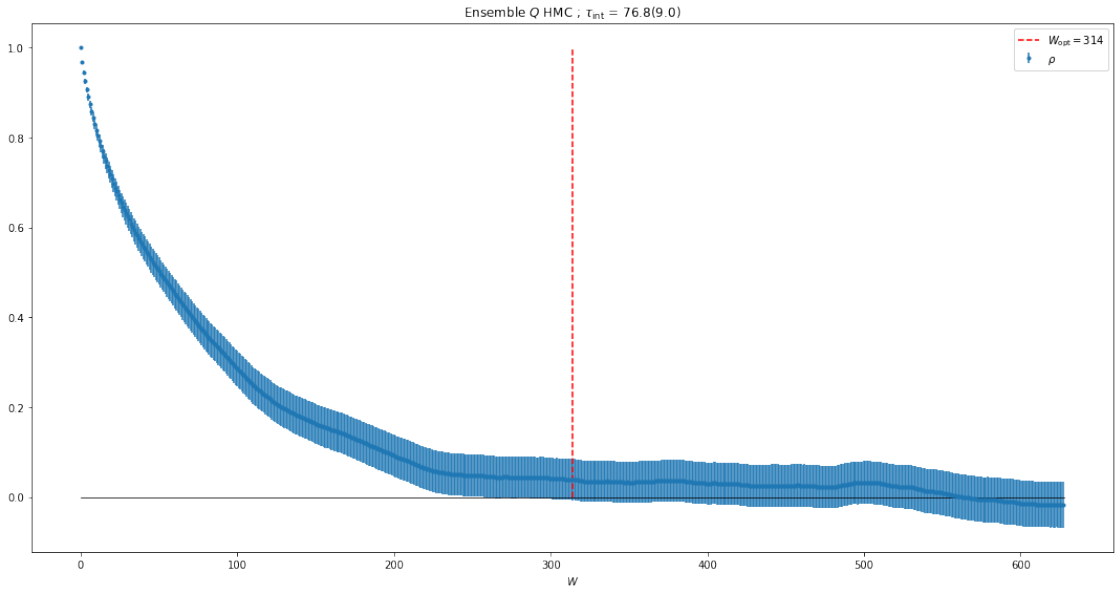
```
In [20]: t2Eat6hmc = np.loadtxt("t2Eat6hmc.txt")
         t2Eat6hmc = t2Eat6hmc[:69000]
         MCtimeHMC = np.arange(1, len(t2Eat6hmc)+1, 1)

         t2Eat6altw = np.loadtxt("t2Eat6altw.txt")
         t2Eat6altw = t2Eat6altw[:49000]
         MCtimealtw = np.arange(1, len(t2Eat6altw)+1, 1)

         t2Eat6alti = np.loadtxt("t2Eat6alti.txt")
         MCtimealti = np.arange(1, len(t2Eat6alti)+1, 1)

         t2Eat6doubaltw = np.loadtxt("t2Eat6doubaltw.txt")
         MCtimedoubaltw = np.arange(1, len(t2Eat6doubaltw)+1, 1)

         Nhmc = len(t2Eat6hmc)
         Naltw = len(t2Eat6altw)
         Ndoubaltw = len(t2Eat6doubaltw)
         Nalti = len(t2Eat6alti)

In [19]: Nhmc

Out[19]: 69000
```

1.4.1 HMC

```
In [22]: a = 2.5
```

value is {{a}}

```
In [16]: corr = observa()
         einfo = errinfo()
         einfo.addEnsemble(0,Stau=1.0)
         corr.primary_observable(0,'$Q$ HMC', [0], ['RO'], [MCtimeHMC.tolist()], [(t2Eat6hmc).t
         [qhmc, eqhmc] = corr.vwerr(errinfo=einfo, )

         print(corr.vwerr(plot=True,errinfo=einfo))
```

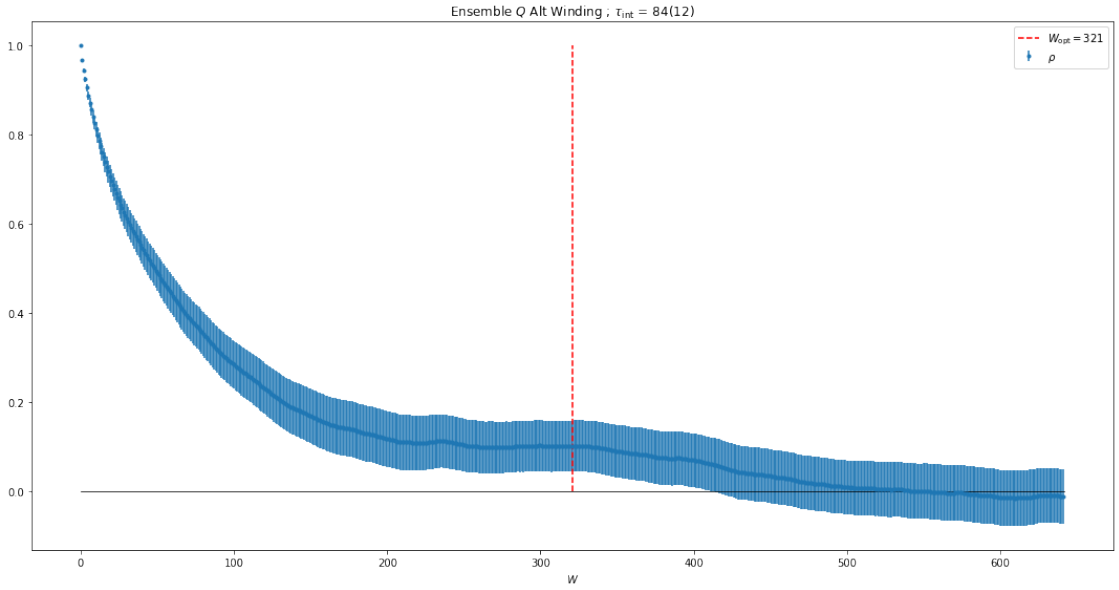



```
[0.15266109705804343, 0.002100366383712266]
```

qhmc

1.4.2 Alt Winding

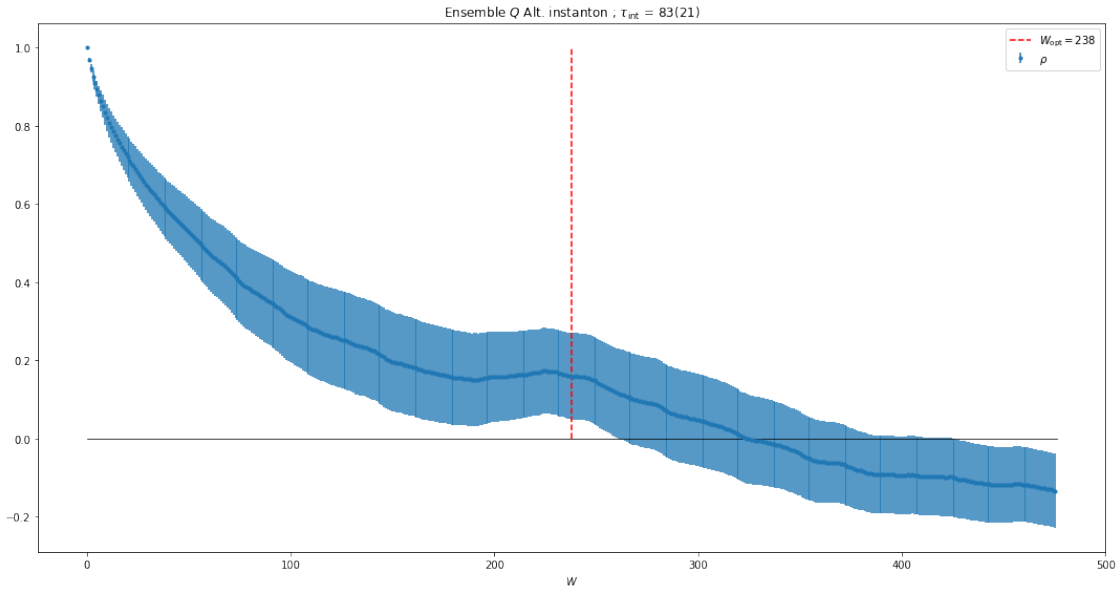
```
In [21]: corr = observa()
         einfo = errinfo()
         einfo.addEnsemble(0,Stau=1.0)
         corr.primary_observable(0,'$Q$ Alt Winding', [0], ['RO'], [MCtimealtw.tolist()], [(t2
         [qaltw, eqaltw] = corr.vwerr(errinfo=einfo, )

         print(corr.vwerr(plot=True,errinfo=einfo))
```




```
[0.14784941006316327, 0.002551051766441731]
```

1.4.3 Alt instanton

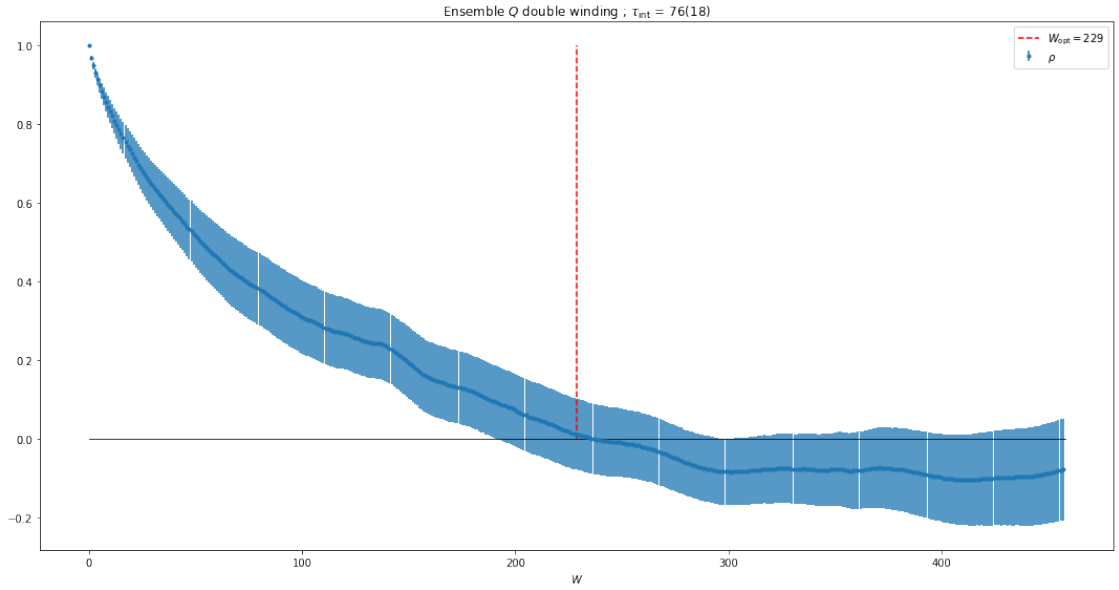
```
In [11]: corr = observa()
         einfo = errinfo()
         einfo.addEnsemble(0,Stau=1.0)
         corr.primary_observable(0,'$Q$ Alt. instanton', [0], ['R0'], [MCtimealti.tolist()], [
         [qalti, eqalti] = corr.vwerr(errinfo=einfo, )

         print(corr.vwerr(plot=True,errinfo=einfo))
```




```
[0.1468619911095, 0.005760415291199244]
```

1.4.4 Double winding

```
In [12]: corr = observa()
         einfo = errinfo()
         einfo.addEnsemble(0,Stau=1.0)
         corr.primary_observable(0,'$Q$ double winding', [0], ['R0'], [MCtimedoubaltw.tolist()]
         [qdoubaltw, eqdoubaltw] = corr.vwerr(errinfo=einfo, )

         print(corr.vwerr(plot=True,errinfo=einfo))
```




```
[0.15304974108391306, 0.005234934776222587]
```

1.4.5 Binning

```
In [13]: errors = []
         ns = []

         burn_in = 0 #how many initial states to discard
         discard = 1 #pick 1 every discard number of states

         gls = (t2Eat6hmc).tolist()

         for n in range(1,1001):

             gls2=[]


             if( (np.double(len(gls))/np.double(n)).is_integer() == True  ):
                 ns.append(n);
                 for i in range(int(len(gls)/n)):

                     med = 0.0
                     for j in range(0,n):
                         med += gls[n*i+j]

                     gls2.append(med/n)
                     #gls2 = np.double(med)/np.double(n)
```

```
        #gls3.append(gls2)


        value = np.mean(np.asarray(gls2[burn_in::discard]))
        evalue = np.std(np.asarray(gls2[burn_in::discard]))/np.sqrt(len(gls2[burn_in:
        errors.append(evalue)
        #disc.append(abs(mom3.evalf()-value)/evalue)

    print("{}% completed!".format(100.0*n/400.0), end='\r')

errorshmc = (errors)
nshmc = (ns)


errors = []
ns = []

gls = (t2Eat6altw).tolist()

for n in range(1,1001):

    gls2=[]


    if( (np.double(len(gls))/np.double(n)).is_integer() == True  ):
        ns.append(n);
        for i in range(int(len(gls)/n)):

            med = 0.0
            for j in range(0,n):
                med += gls[n*i+j]

            gls2.append(med/n)
            #gls2 = np.double(med)/np.double(n)
        #gls3.append(gls2)


        value = np.mean(np.asarray(gls2[burn_in::discard]))
        evalue = np.std(np.asarray(gls2[burn_in::discard]))/np.sqrt(len(gls2[burn_in:
        errors.append(evalue)
        #disc.append(abs(mom3.evalf()-value)/evalue)

    print("{}% completed!".format(100.0*n/400.0), end='\r')

errorsaltw = (errors)
nsaltw = (ns)
```

```
errors = []
ns = []

gls = (t2Eat6alti).tolist()


for n in range(1,1001):

    gls2=[]


    if( (np.double(len(gls))/np.double(n)).is_integer() == True  ):
        ns.append(n);
        for i in range(int(len(gls)/n)):

            med = 0.0
            for j in range(0,n):
                med += gls[n*i+j]

            gls2.append(med/n)
            #gls2 = np.double(med)/np.double(n)
        #gls3.append(gls2)


        value = np.mean(np.asarray(gls2[burn_in::discard]))
        evalue = np.std(np.asarray(gls2[burn_in::discard]))/np.sqrt(len(gls2[burn_in:
        errors.append(evalue)
        #disc.append(abs(mom3.evalf()-value)/evalue)

    print("{}% completed!".format(100.0*n/400.0), end='\r')

errorsalti = (errors)
nsalti = (ns)



errors = []
ns = []

gls = (t2Eat6doubaltw).tolist()

for n in range(1,1001):

    gls2=[]


    if( (np.double(len(gls))/np.double(n)).is_integer() == True  ):
        ns.append(n);
```

```
                for i in range(int(len(gls)/n)):

                    med = 0.0
                    for j in range(0,n):
                        med += gls[n*i+j]

                    gls2.append(med/n)
                    #gls2 = np.double(med)/np.double(n)
                #gls3.append(gls2)


                value = np.mean(np.asarray(gls2[burn_in::discard]))
                evalue = np.std(np.asarray(gls2[burn_in::discard]))/np.sqrt(len(gls2[burn_in:
                errors.append(evalue)
                #disc.append(abs(mom3.evalf()-value)/evalue)

            print("{}% completed!".format(100.0*n/400.0), end='\r')

        errorsdoubaltw = (errors)
        nsdoubaltw = (ns)
```

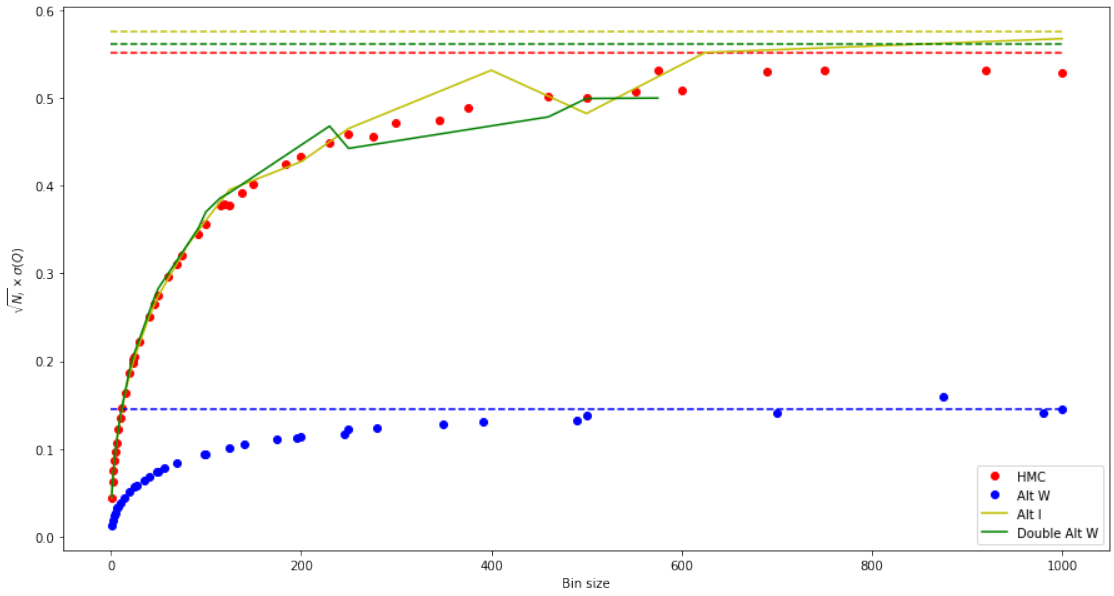
```
250.0% completed!!
```

In [88]:
```
(eqaltw)**2.0/(eqhmc)**2.0*12.0/39.5
```

Out[88]:
```
0.641056655804379
```

In [14]:
```
plt.plot(np.array(nshmc), np.sqrt(Nhmc)*np.array(errorshmc),'ro', label="HMC")
plt.plot([0,1000],[np.sqrt(Nhmc)*eqhmc,np.sqrt(Nhmc)*eqhmc], 'r--')

plt.plot(np.array(nsaltw), np.sqrt(Naltw)*np.array(errorsaltw),'bo', label="Alt W")
plt.plot([0,1000],[np.sqrt(Naltw)*eqaltw,np.sqrt(Naltw)*eqaltw], 'b--')

plt.plot(np.array(nsalti), np.sqrt(Nalti)*np.array(errorsalti),'y-', label="Alt I")
plt.plot([0,1000],[np.sqrt(Nalti)*eqalti,np.sqrt(Nalti)*eqalti], 'y--')

plt.plot(np.array(nsdoubaltw), np.sqrt(Ndoubaltw)*np.array(errorsdoubaltw),'g-', label
plt.plot([0,1000],[np.sqrt(Ndoubaltw)*eqdoubaltw,np.sqrt(Ndoubaltw)*eqdoubaltw], 'g--

plt.legend()

plt.ylabel("$\sqrt{N_i} \\times \sigma (Q)$")
plt.xlabel("Bin size")
#plt.xlim(0.0,0.1)
```

Out[14]:
```
Text(0.5,0,'Bin size')
```



1.5 Q

In [359]:
```
topchargehmc = np.loadtxt("topchargeHMC.txt")
topchargehmc = topchargehmc[:69000]
topchargealtw = np.loadtxt("topchargealtw.txt")
topchargealtw = topchargealtw[:49000]
topchargedoubaltw = np.loadtxt("topchargedoubaltw.txt")
topchargedoubaltw = topchargedoubaltw[:11000]
topchargealti = np.loadtxt("topchargealti.txt")

topchargealtw2 = np.loadtxt("topchargealtw2.txt")
topchargealtw2 = topchargealtw2[:22000]
topchargealtw3 = np.loadtxt("topchargealtw3.txt")
topchargealtw3 = topchargealtw3[:16000]

MCtimehmc = np.arange(1, len(topchargehmc)+1, 1)
MCtimealtw = np.arange(1, len(topchargealtw)+1, 1)
MCtimedoubaltw = np.arange(1, len(topchargedoubaltw)+1, 1)
MCtimealti = np.arange(1, len(topchargealti)+1, 1)
MCtimealtw2 = np.arange(1, len(topchargealtw2)+1, 1)
MCtimealtw3 = np.arange(1, len(topchargealtw3)+1, 1)

Nhmc = len(topchargehmc)
Naltw = len(topchargealtw)
Ndoubaltw = len(topchargedoubaltw)
Nalti = len(topchargealti)
Naltw2 = len(topchargealtw2)
```

```
Naltw3 = len(topchargealtw3)
```

```
In [360]: print("Configuraciones HMC: {}".format(len(topchargeHMC)))
          print("Configuraciones altwinding: {}".format(len(topchargealtw)))
          print("Configuraciones double winding: {}".format(len(topchargedoubaltw)))
          print("Configuraciones alt instanton: {}".format(len(topchargealti)))
          print("Configuraciones alt instanton: {}".format(len(topchargealtw2)))
          print("Configuraciones alt instanton: {}".format(len(topchargealtw3)))
```

```
Configuraciones HMC: 69000
Configuraciones altwinding: 49000
Configuraciones double winding: 11000
Configuraciones alt instanton: 10000
Configuraciones alt instanton: 22000
Configuraciones alt instanton: 16000
```

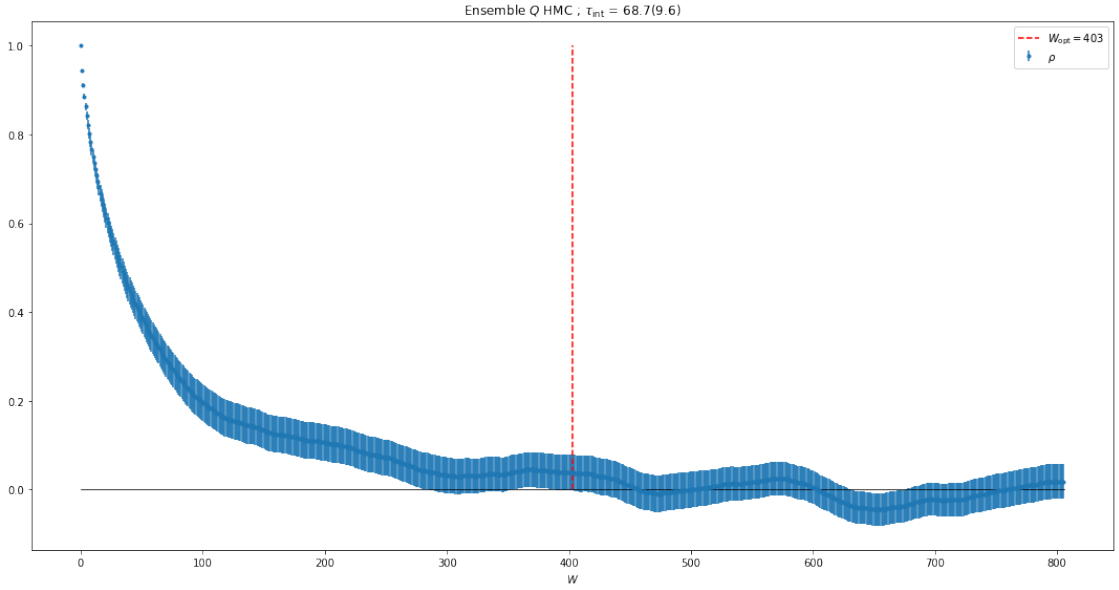
Algorithm	Statistics	Q	τ_{int}	$\tau_{\text{int}}/\tau_{\text{int}}^{(\text{HMC})}$
HMC	{{Nhmc}}	{{printwe(Qhmc,eQhmc)}}	{{printwe(tauQhmc, etauQhmc)}}	-
Alt Winding	{{Naltw}}	{{printwe(Qaltw,eQaltw)}}	{{printwe(tauQaltw, etauQaltw)}}	{{printwe(tauR_Qaltw, etauR_Qaltw)}}
Alt Instanton	{{Nalti}}	{{printwe(Qalti,eQalti)}}	{{printwe(tauQalti, etauQalti)}}	{{printwe(tauR_Qalti, etauR_Qalti)}}
Alt Double Winding	{{Ndoubaltw}}	{{printwe(Qdoubaltw,eQdoubaltw)}}	{{printwe(tauQdoubaltw, etauQdoubaltw)}}	{{printwe(tauR_Qdoubaltw, etauR_Qdoubaltw)}}
Alt Winding 2	{{Naltw2}}	{{printwe(Qaltw2,eQaltw2)}}	{{printwe(tauQaltw2, etauQaltw2)}}	{{printwe(tauR_Qaltw2, etauR_Qaltw2)}}
Alt Winding 3	{{Naltw3}}	{{printwe(Qaltw3,eQaltw3)}}	{{printwe(tauQaltw3, etauQaltw3)}}	{{printwe(tauR_Qaltw3, etauR_Qaltw3)}}

1.5.1 HMC

$P =$ {{printwe(Qhmc, eQhmc)}}
$\tau_{int,Q} =$ {{printwe(tauQhmc, etauQhmc)}}

```
In [240]: corr_qhmc = observa()
          einfo = errinfo()
          einfo.addEnsemble(0, Stau=1.5)
          corr_qhmc.primary_observable(0,'$Q$ HMC', [0], ['RO'], [MCtimehmc.tolist()], [(topcha

          [Qhmc, eQhmc]= corr_qhmc.vwerr(errinfo=einfo)
          [tauQhmc, etauQhmc] = corr_qhmc.tauint()
          tauQhmc = tauQhmc[0][0][0]
          etauQhmc = etauQhmc[0][0][0]

          print(corr_qhmc.vwerr(plot=True,errinfo=einfo))
```

```
printwe(Qhmc,eQhmc)
```

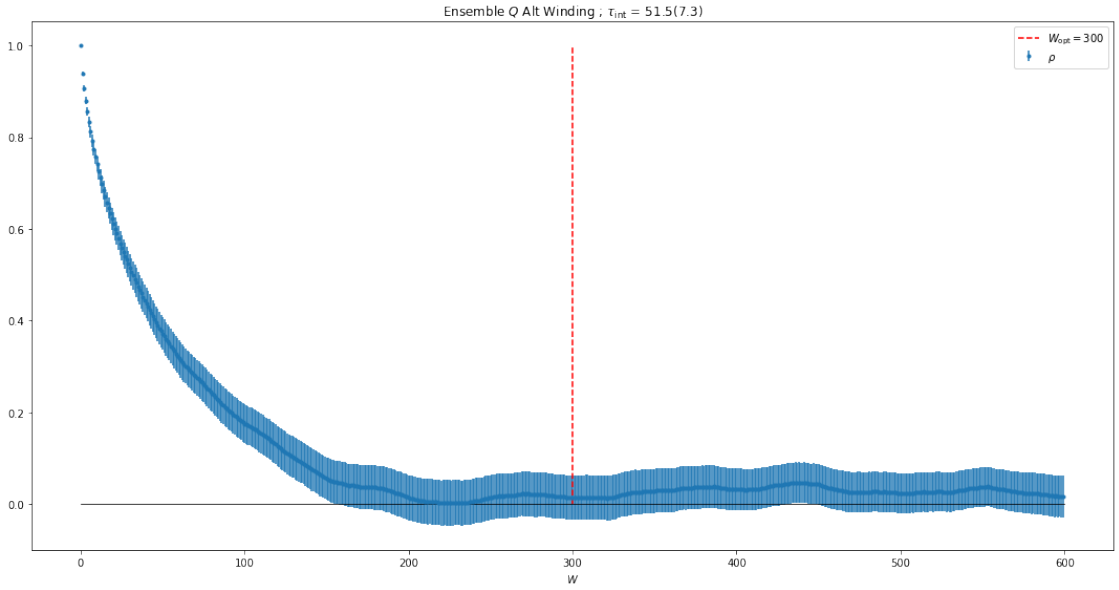



```
[-0.0029484408451943356, 0.0404168404662309]
-0.003(40)
```

1.5.2 Alternating Winding

$Q = \{\{\text{printwe(Qaltw, eQaltw)}\}\}$
$\tau_{int,Q} = \{\{\text{printwe(tauQaltw, etauQaltw)}\}\}$
$\tau_{int}/\tau_{int}^{(HMC)} = \{\{\text{printwe(tauR\_Qaltw, etauR\_Qaltw)}\}\}$

```
In [251]: corr_qaltw = observa()
          einfo = errinfo()
          einfo.addEnsemble(0, Stau=1.5)
          corr_qaltw.primary_observable(0,'$Q$ Alt Winding', [0], ['R0'], [MCtimealtw.tolist()]

          [Qaltw, eQaltw]= corr_qaltw.vwerr(errinfo=einfo)
          [tauQaltw, etauQaltw] = corr_qaltw.tauint()
          tauQaltw = tauQaltw[0][0][0]
          etauQaltw = etauQaltw[0][0][0]

          print(corr_qaltw.vwerr(plot=True,errinfo=einfo))

          printwe(Qaltw,eQaltw)

          tauR_Qaltw = tauQaltw/tauQhmc
          etauR_Qaltw = tauR_Qaltw *  np.sqrt( (etauQhmc/tauQhmc)**2.0 + (etauQaltw/tauQaltw)**
```

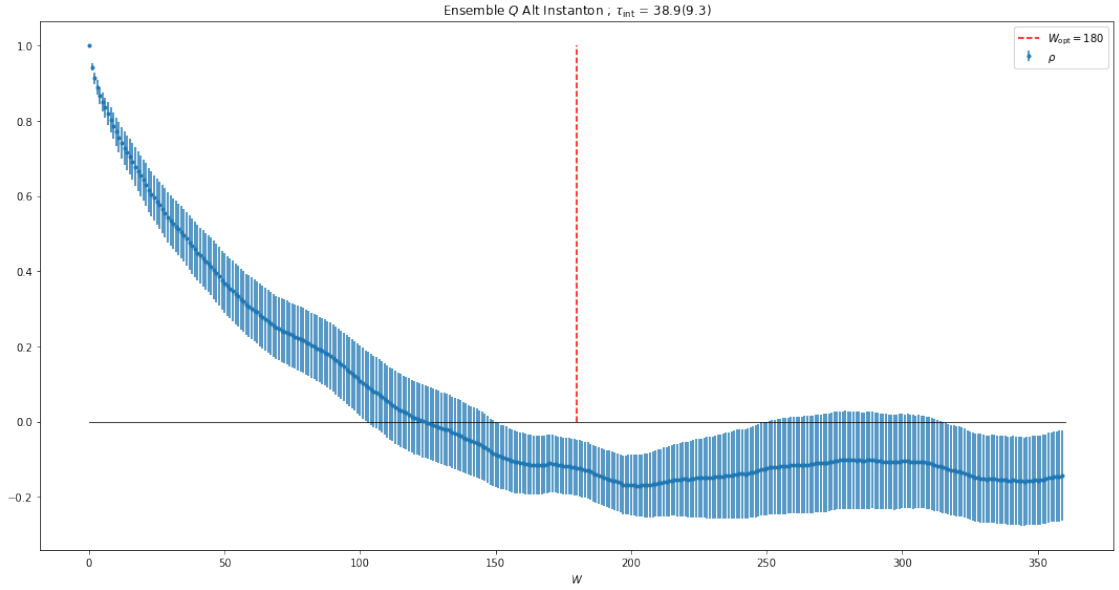



```
[-0.0700155782083696, 0.0399013919622557]
-0.07(40)
```

1.5.3 Alternating instanton

$Q = \{\{\text{printwe(Qalti, eQalti)}\}\}$
$\tau_{int,Q} = \{\{\text{printwe(tauQalti, etauQalti)}\}\}$
$\tau_{int}/\tau_{int}^{(HMC)} = \{\{\text{printwe(tauR\_Qalti, etauR\_Qalti)}\}\}$

```
In [252]: corr_qalti = observa()
          einfo = errinfo()
          einfo.addEnsemble(0, Stau=1.5)
          corr_qalti.primary_observable(0,'$Q$ Alt Instanton', [0], ['R0'], [MCtimealti.tolist

          [Qalti, eQalti]= corr_qalti.vwerr(errinfo=einfo)
          [tauQalti, etauQalti] = corr_qalti.tauint()
          tauQalti = tauQalti[0][0][0]
          etauQalti = etauQalti[0][0][0]

          print(corr_qalti.vwerr(plot=True,errinfo=einfo))

          printwe(Qalti,eQalti)

          tauR_Qalti = tauQalti/tauQhmc
          etauR_Qalti = tauR_Qalti *  np.sqrt( (etauQhmc/tauQhmc)**2.0 + (etauQalti/tauQalti)*
```

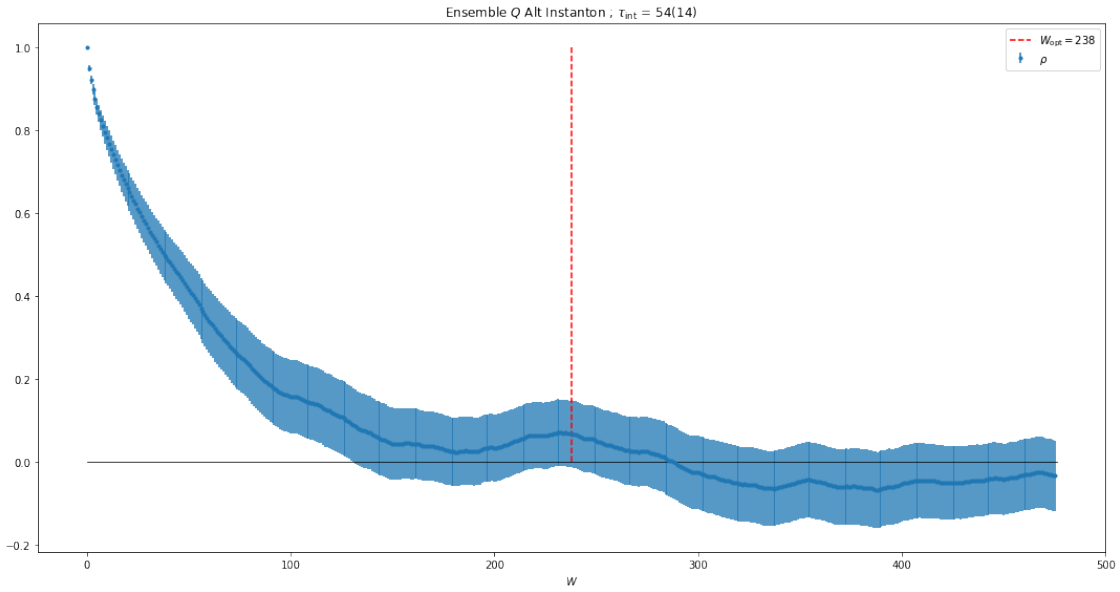



```
[0.14106735732349657, 0.07680352873332864]
0.141(77)
```

1.5.4 Double Winding

$Q =$ {{printwe(Qdoubaltw, eQdoubaltw)}}
$\tau_{int,Q} =$ {{printwe(tauQdoubaltw, etauQdoubaltw)}}
$\tau_{int}/\tau_{int}^{(HMC)} =$ {{printwe(tauR_Qdoubaltw, etauR_Qdoubaltw)}}

```
In [255]: corr_qdoubaltw = observa()
          einfo = errinfo()
          einfo.addEnsemble(0, Stau=1.5)
          corr_qdoubaltw.primary_observable(0,'$Q$ Alt Instanton', [0], ['R0'], [MCtimedoubaltw

          [Qdoubaltw, eQdoubaltw]= corr_qdoubaltw.vwerr(errinfo=einfo)
          [tauQdoubaltw, etauQdoubaltw] = corr_qdoubaltw.tauint()
          tauQdoubaltw = tauQdoubaltw[0][0][0]
          etauQdoubaltw = etauQdoubaltw[0][0][0]

          print(corr_qdoubaltw.vwerr(plot=True,errinfo=einfo))

          printwe(Qdoubaltw,eQdoubaltw)

          tauR_Qdoubaltw = tauQdoubaltw/tauQhmc
          etauR_Qdoubaltw = tauR_Qdoubaltw *  np.sqrt( (etauQhmc/tauQhmc)**2.0 + (etauQdoubaltw
```

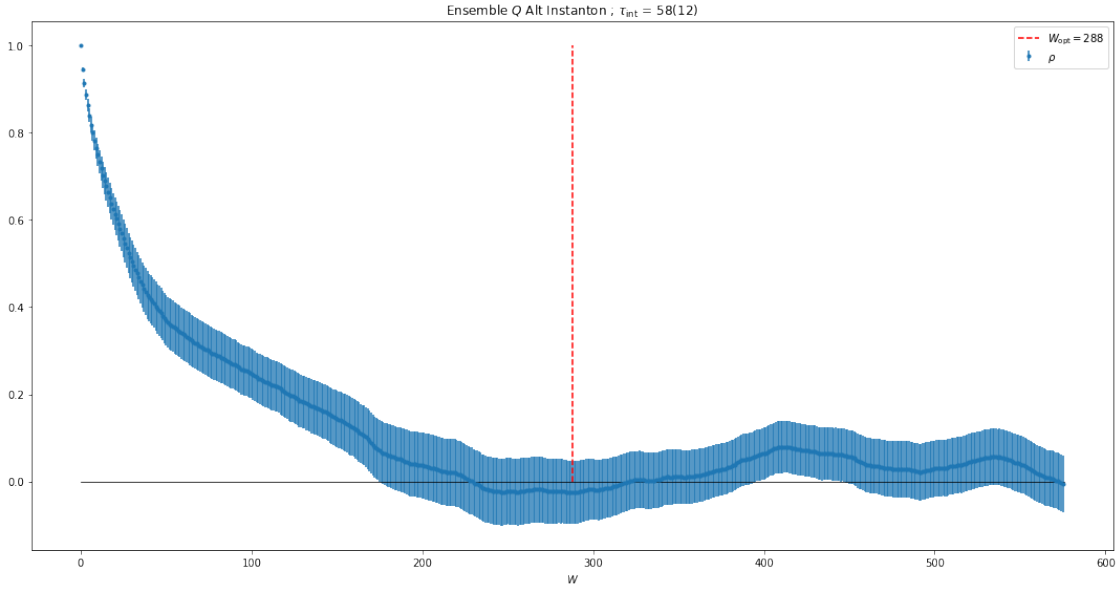



```
[-0.14008025448150208, 0.09506703292075573]
-0.14(95)
```

1.5.5 Alternating Winding half side 2

$Q = \{\{\text{printwe(Qaltw2, eQaltw2)}\}\}$
$\tau_{int,Q} = \{\{\text{printwe(tauQaltw2, etauQaltw2)}\}\}$
$\tau_{int}/\tau_{int}^{(HMC)} = \{\{\text{printwe(tauR\_Qaltw2, etauR\_Qaltw2)}\}\}$

```
In [357]: corr_qaltw2 = observa()
          einfo = errinfo()
          einfo.addEnsemble(0, Stau=1.5)
          corr_qaltw2.primary_observable(0,'$Q$ Alt Instanton', [0], ['R0'], [MCtimealtw2.tolis

          [Qaltw2, eQaltw2]= corr_qaltw2.vwerr(errinfo=einfo)
          [tauQaltw2, etauQaltw2] = corr_qaltw2.tauint()
          tauQaltw2 = tauQaltw2[0][0][0]
          etauQaltw2 = etauQaltw2[0][0][0]

          print(corr_qaltw2.vwerr(plot=True,errinfo=einfo))

          printwe(Qaltw2,eQaltw2)

          tauR_Qaltw2 = tauQaltw2/tauQhmc
          etauR_Qaltw2 = tauR_Qaltw2 *  np.sqrt( (etauQhmc/tauQhmc)**2.0 + (etauQaltw2/tauQaltw
```

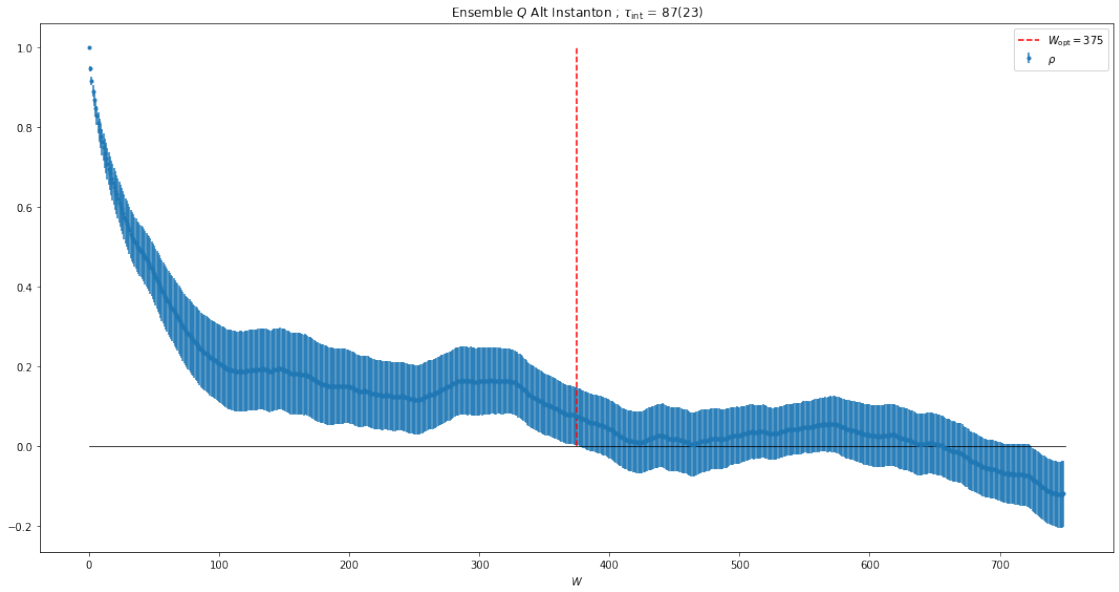



```
[-0.04345579221622017, 0.0653456523523522]
-0.043(65)
```

1.5.6 Alternating Winding half side 3

$Q = \{\{\text{printwe(Qaltw3, eQaltw3)}\}\}$
$\tau_{int,Q} = \{\{\text{printwe(tauQaltw3, etauQaltw3)}\}\}$
$\tau_{int}/\tau_{int}^{(HMC)} = \{\{\text{printwe(tauR\_Qaltw3, etauR\_Qaltw3)}\}\}$

```
In [361]: corr_qaltw3 = observa()
          einfo = errinfo()
          einfo.addEnsemble(0, Stau=1.5)
          corr_qaltw3.primary_observable(0,'$Q$ Alt Instanton', [0], ['R0'], [MCtimealtw3.tolis

          [Qaltw3, eQaltw3]= corr_qaltw3.vwerr(errinfo=einfo)
          [tauQaltw3, etauQaltw3] = corr_qaltw3.tauint()
          tauQaltw3 = tauQaltw3[0][0][0]
          etauQaltw3 = etauQaltw3[0][0][0]

          print(corr_qaltw3.vwerr(plot=True,errinfo=einfo))

          printwe(Qaltw3,eQaltw3)

          tauR_Qaltw3 = tauQaltw3/tauQhmc
          etauR_Qaltw3 = tauR_Qaltw3 *  np.sqrt( (etauQhmc/tauQhmc)**2.0 + (etauQaltw3/tauQaltw
```




```
[0.07214080833018939, 0.09838212376404576]
0.072(98)
```

1.5.7 Binning

```
In [27]: errors = []
         ns = []

         burn_in = 0 #how many initial states to discard
         discard = 1 #pick 1 every discard number of states

         gls = (topchargeHMC).tolist()

         for n in range(1,1001):

             gls2=[]


             if( (np.double(len(gls))/np.double(n)).is_integer() == True  ):
                 ns.append(n);
                 for i in range(int(len(gls)/n)):

                     med = 0.0
                     for j in range(0,n):
                         med += gls[n*i+j]

                     gls2.append(med/n)
```

```
            #gls2 = np.double(med)/np.double(n)
        #gls3.append(gls2)


        value = np.mean(np.asarray(gls2[burn_in::discard]))
        evalue = np.std(np.asarray(gls2[burn_in::discard]))/np.sqrt(len(gls2[burn_in:
        errors.append(evalue)
        #disc.append(abs(mom3.evalf()-value)/evalue)

    print("{}% completed!".format(100.0*n/400.0), end='\r')

errorshmc = (errors)
nshmc = (ns)


errors = []
ns = []

gls = (topchargealtw).tolist()

for n in range(1,1001):

    gls2=[]


    if( (np.double(len(gls))/np.double(n)).is_integer() == True  ):
        ns.append(n);
        for i in range(int(len(gls)/n)):

            med = 0.0
            for j in range(0,n):
                med += gls[n*i+j]

            gls2.append(med/n)
            #gls2 = np.double(med)/np.double(n)
        #gls3.append(gls2)


        value = np.mean(np.asarray(gls2[burn_in::discard]))
        evalue = np.std(np.asarray(gls2[burn_in::discard]))/np.sqrt(len(gls2[burn_in:
        errors.append(evalue)
        #disc.append(abs(mom3.evalf()-value)/evalue)

    print("{}% completed!".format(100.0*n/400.0), end='\r')

errorsaltw = (errors)
nsaltw = (ns)
```

```
errors = []
ns = []

gls = (topchargealti).tolist()


for n in range(1,1001):

    gls2=[]


    if( (np.double(len(gls))/np.double(n)).is_integer() == True  ):
        ns.append(n);
        for i in range(int(len(gls)/n)):

            med = 0.0
            for j in range(0,n):
                med += gls[n*i+j]

            gls2.append(med/n)
            #gls2 = np.double(med)/np.double(n)
        #gls3.append(gls2)


        value = np.mean(np.asarray(gls2[burn_in::discard]))
        evalue = np.std(np.asarray(gls2[burn_in::discard]))/np.sqrt(len(gls2[burn_in:
        errors.append(evalue)
        #disc.append(abs(mom3.evalf()-value)/evalue)

    print("{}% completed!".format(100.0*n/400.0), end='\r')

errorsalti = (errors)
nsalti = (ns)



errors = []
ns = []

gls = (topchargedoubaltw).tolist()

for n in range(1,1001):

    gls2=[]


    if( (np.double(len(gls))/np.double(n)).is_integer() == True  ):
```

```
                ns.append(n);
                for i in range(int(len(gls)/n)):

                    med = 0.0
                    for j in range(0,n):
                        med += gls[n*i+j]

                    gls2.append(med/n)
                    #gls2 = np.double(med)/np.double(n)
                #gls3.append(gls2)


                value = np.mean(np.asarray(gls2[burn_in::discard]))
                evalue = np.std(np.asarray(gls2[burn_in::discard]))/np.sqrt(len(gls2[burn_in:
                errors.append(evalue)
                #disc.append(abs(mom3.evalf()-value)/evalue)

            print("{}% completed!".format(100.0*n/400.0), end='\r')

        errorsdoubaltw = (errors)
        nsdoubaltw = (ns)
```

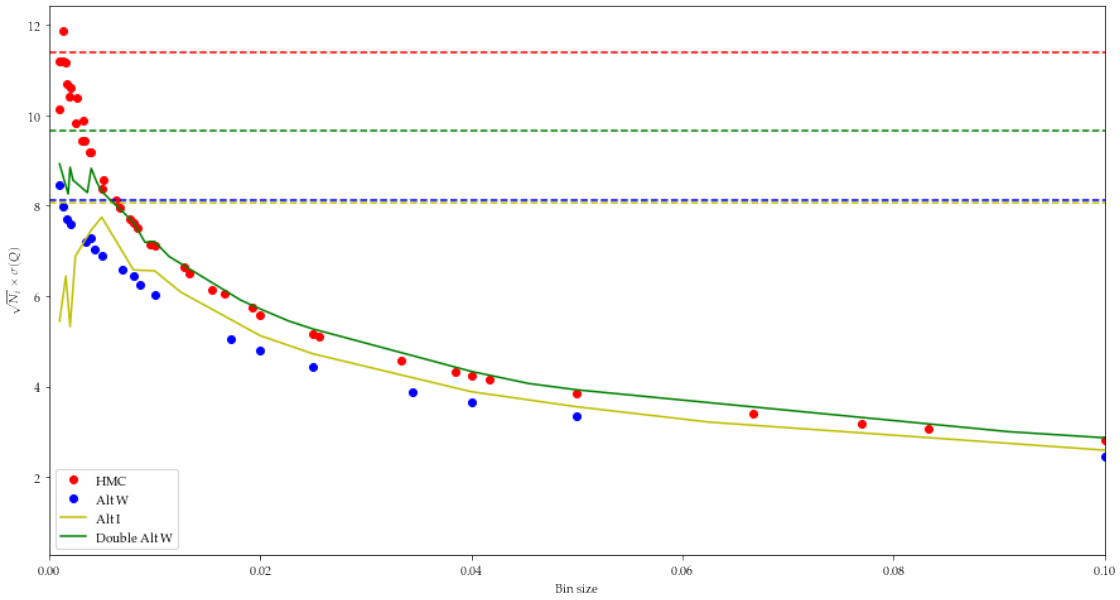
```
250.0% completed!!
```

In [29]:
```
Nhmc
```

Out[29]:
```
39000
```

In [31]:
```
plt.plot(1.0/np.array(nshmc), np.sqrt(Nhmc)*np.array(errorshmc),'ro', label="HMC")
plt.plot([0,1],[np.sqrt(Nhmc)*eqhmc,np.sqrt(Nhmc)*eqhmc], 'r--')

plt.plot(1.0/np.array(nsaltw), np.sqrt(Naltw)*np.array(errorsaltw),'bo', label="Alt W
plt.plot([0,1],[np.sqrt(Naltw)*eqaltw,np.sqrt(Naltw)*eqaltw], 'b--')

plt.plot(1.0/np.array(nsalti), np.sqrt(Nalti)*np.array(errorsalti),'y-', label="Alt I
plt.plot([0,1],[np.sqrt(Nalti)*eqalti,np.sqrt(Nalti)*eqalti], 'y--')

plt.plot(1.0/np.array(nsdoubaltw), np.sqrt(Ndoubaltw)*np.array(errorsdoubaltw),'g-',
plt.plot([0,1],[np.sqrt(Ndoubaltw)*eqdoubaltw,np.sqrt(Ndoubaltw)*eqdoubaltw], 'g--')

plt.legend()

plt.ylabel("$\sqrt{N_i} \\times \sigma (Q)$")
plt.xlabel("Bin size")
plt.xlim(0.0,0.1)
```

Out[31]:
```
(0.0, 0.1)
```



Binning errors This computes the errors of the error with a Jackknife method. You can compare the results with automatic windowing procedure: the relative errors of the error of the error should be similar to the relative error of the autocorrelation time.

```
In [42]: import numpy as np
         import glob
         import numpy.ma as ma
         import time

         def iden(x):
             return x

         def stdev(x):
             return np.std(x)/np.sqrt(len(x))

         def removeelement (array, index):
             return np.delete(array, (index), axis=0)


         def jackknife(x, func):
             """Jackknife estimate of the estimator func"""
             n = len(x)
             idx = np.arange(n)
             avg = 0.
             for i in idx:
                 xaux = np.apply_along_axis(stdev,0,removeelement(x,i))
                 #       avg = avg + np.mean(np.apply_along_axis(func, 1, removeelement(x,i)))
```

```
            avg = avg + func(xaux)
        return (avg/n)

    def jackknifevar(x, xmean, func):
        n = len(x)
        idx = np.arange(n)
        var = 0.
        for i in idx:
            #  print (np.apply_along_axis(func, 1, removeelement(x,i)))
            #  print np.mean(np.apply_along_axis(func, 1, removeelement(x,i)))
            xi = func(np.apply_along_axis(stdev,0,removeelement(x,i)))
            var = var + (xi-xmean)**2
        return np.sqrt(var*(n-1)/n)
```

```
In [8]: err = jackknife(gls2,iden)
```

```
In [9]: jackknifevar(gls2,err,iden)
```

```
Out[9]: 0.0042627183143538775
```

```
In [10]: err
```

```
Out[10]: 0.03809522962034062
```

```
In [11]: evalue
```

```
Out[11]: 0.037825741661202214
```

```
In [43]: errors = []
         errorsbar = []
         ns = []

         burn_in = 0 #how many initial states to discard
         discard = 1 #pick 1 every discard number of states

         gls = (topchargeHMC).tolist()

         for n in range(50,1001):

             gls2=[]


             if( (np.double(len(gls))/np.double(n)).is_integer() == True  ):
                 ns.append(n);
                 for i in range(int(len(gls)/n)):

                     med = 0.0
                     for j in range(0,n):
                         med += gls[n*i+j]
```

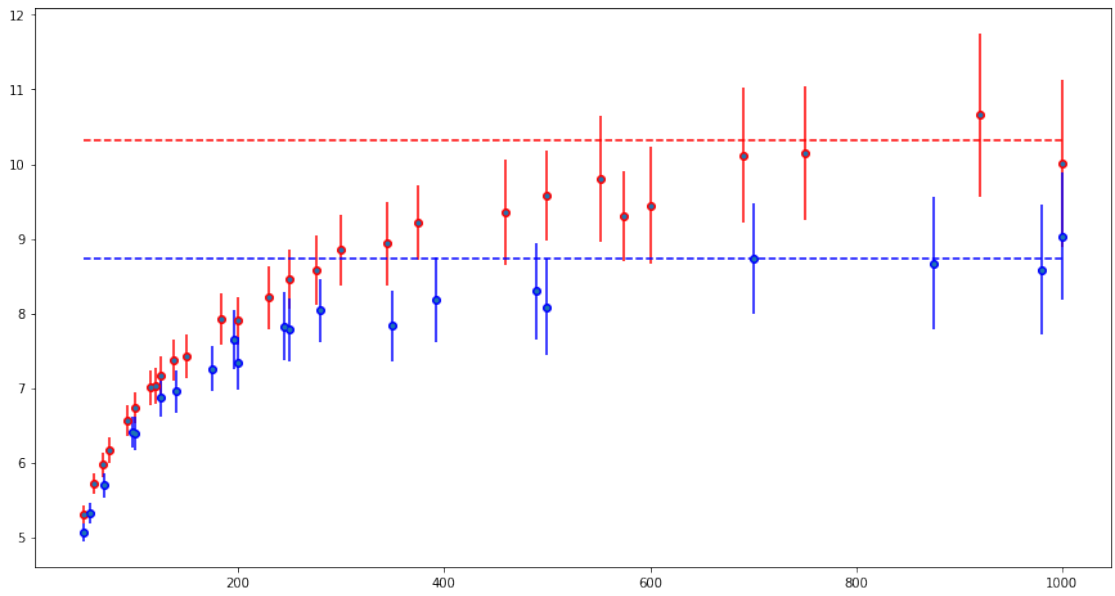
```
            gls2.append(med/n)
            #gls2 = np.double(med)/np.double(n)
        #gls3.append(gls2)


        value = np.mean(np.asarray(gls2[burn_in::discard]))
        #evalue = np.std(np.asarray(gls2[burn_in::discard]))/np.sqrt(len(gls2[burn_in
        evalue = jackknife(gls2,iden)
        evalueerr = jackknifevar(gls2,evalue,iden)
        errors.append(evalue)
        errorsbar.append(evalueerr)
        #disc.append(abs(mom3.evalf()-value)/evalue)

    print("{}% completed!".format(100.0*n/400.0), end='\r')

errorshmc = (errors)
errorsbarhmc = errorsbar
nshmc = (ns)


errors = []
errorsbar = []
ns = []

gls = (topchargealtw).tolist()

for n in range(50,1001):

    gls2=[]


    if( (np.double(len(gls))/np.double(n)).is_integer() == True  ):
        ns.append(n);
        for i in range(int(len(gls)/n)):

            med = 0.0
            for j in range(0,n):
                med += gls[n*i+j]

            gls2.append(med/n)
            #gls2 = np.double(med)/np.double(n)
        #gls3.append(gls2)


        value = np.mean(np.asarray(gls2[burn_in::discard]))
        #evalue = np.std(np.asarray(gls2[burn_in::discard]))/np.sqrt(len(gls2[burn_in
        evalue = jackknife(gls2,iden)
```

```
            evalueerr = jackknifevar(gls2,evalue,iden)
            errors.append(evalue)
            errorsbar.append(evalueerr)
            #disc.append(abs(mom3.evalf()-value)/evalue)

        print("{}% completed!".format(100.0*n/400.0), end='\r')

    errorsaltw = (errors)
    errorsbaraltw = errorsbar
    nsaltw = (ns)
```

```
250.0% completed!!
```

```
In [44]: plt.plot(np.array(nshmc), np.sqrt(Nhmc)*np.array(errorshmc),'ro', label="HMC")
         plt.errorbar(np.array(nshmc), np.sqrt(Nhmc)*np.array(errorshmc), yerr=np.sqrt(Nhmc)*np
         plt.plot([50,1000],[np.sqrt(Nhmc)*eqhmc,np.sqrt(Nhmc)*eqhmc], 'r--')

         plt.plot(np.array(nsaltw), np.sqrt(Naltw)*np.array(errorsaltw),'bo', label="Alt W")
         plt.errorbar(np.array(nsaltw), np.sqrt(Naltw)*np.array(errorsaltw),np.sqrt(Naltw)*np.a
         plt.plot([50,1000],[np.sqrt(Naltw)*eqaltw,np.sqrt(Naltw)*eqaltw], 'b--')
```

```
Out[44]: [<matplotlib.lines.Line2D at 0x7fcbc4d75090>]
```



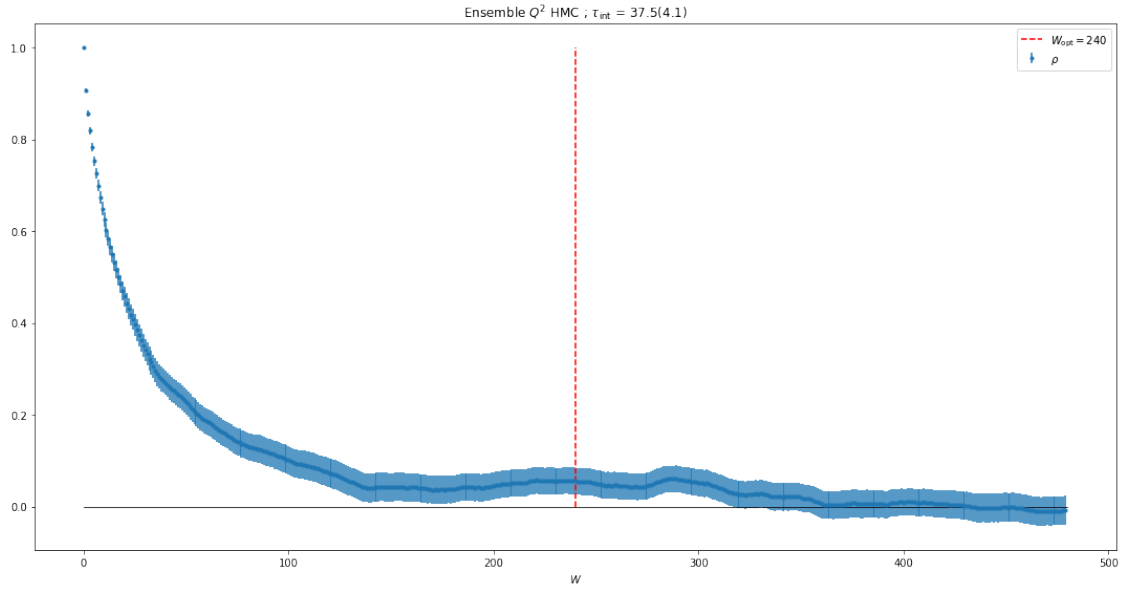
1.6 Q^2

Algorithm	Statistics	Q^2	τ_{int}	$\tau_{\text{int}}/\tau_{\text{int}}^{(\text{HMC})}$
HMC	{{Nhmc}}	{{printwe(Q2hmc,eQ2hmc)}}	{{printwe(tauQ2hmc, etauQ2hmc)}}	-
Alt Winding	{{Naltw}}	{{printwe(Q2altw,eQ2altw)}}	{{printwe(tauQ2altw, etauQ2altw)}}	{{printwe(tauR_Q2altw, etauR_Q2altw)}}
Alt Instanton	{{Nalti}}	{{printwe(Q2alti,eQ2alti)}}	{{printwe(tauQ2alti, etauQ2alti)}}	{{printwe(tauR_Q2alti, etauR_Q2alti)}}
Alt Double Winding	{{Ndoubaltw}}	{{printwe(Q2doubaltw,eQ2doubaltw)}}	{{printwe(tauQ2doubaltw, etauQ2doubaltw)}}	{{printwe(tauR_Q2doubaltw, etauR_Q2doubaltw)}}
Alt Instanton	{{Naltw2}}	{{printwe(Q2altw2,eQ2altw2)}}	{{printwe(tauQ2altw2, etauQ2altw2)}}	{{printwe(tauR_Q2altw2, etauR_Q2altw2)}}
Alt Instanton	{{Naltw3}}	{{printwe(Q2altw3,eQ2altw3)}}	{{printwe(tauQ2altw3, etauQ2altw3)}}	{{printwe(tauR_Q2altw3, etauR_Q2altw3)}}

1.6.1 HMC

$P =$ {{printwe(Q2hmc, eQ2hmc)}}
$\tau_{int,Q^2} =$ {{printwe(tauQ2hmc, etauQ2hmc)}}

```
In [258]: corr_q2hmc = observa()
          einfo = errinfo()
          einfo.addEnsemble(0, Stau=1.5)
          corr_q2hmc.primary_observable(0,'$Q^2$ HMC', [0], ['R0'], [MCtimehmc.tolist()], [(to

          [Q2hmc, eQ2hmc]= corr_q2hmc.vwerr(errinfo=einfo)
          [tauQ2hmc, etauQ2hmc] = corr_q2hmc.tauint()
          tauQ2hmc = tauQ2hmc[0][0][0]
          etauQ2hmc = etauQ2hmc[0][0][0]

          print(corr_q2hmc.vwerr(plot=True,errinfo=einfo))

          printwe(Q2hmc,eQ2hmc)
```

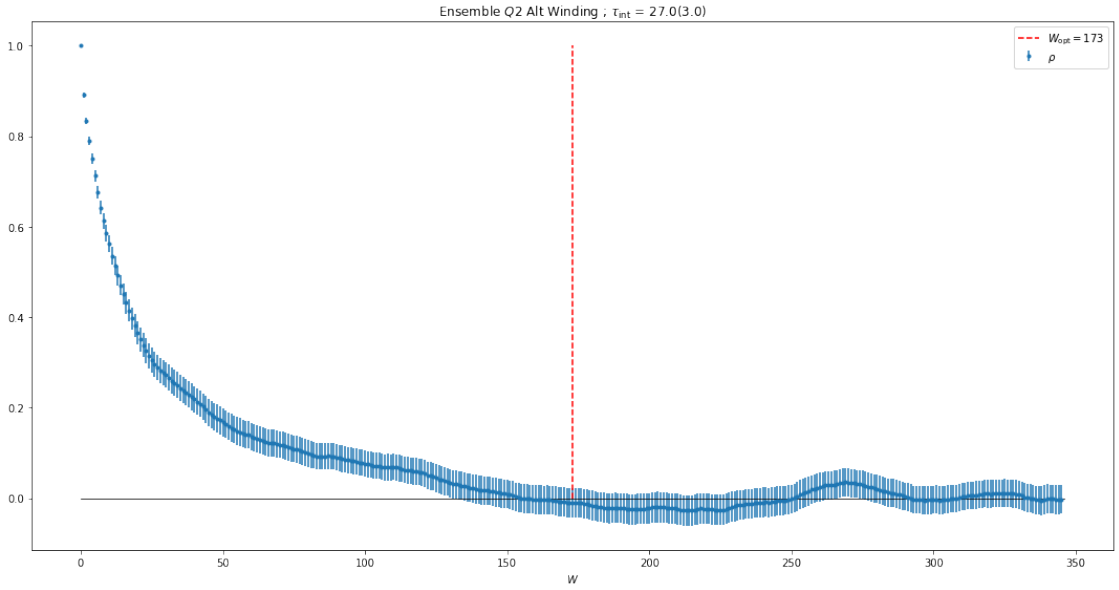



```
[0.8191819850747745, 0.04432421659296179]
0.819(44)
```

1.6.2 Alternating Winding

$Q^2 =$ {{printwe(Q2altw, eQ2altw)}}
$\tau_{int,Q^2} =$ {{printwe(tauQ2altw, etauQ2altw)}}
$\tau_{int}/\tau_{int}^{(HMC)} =$ {{printwe(tauR_Q2altw, etauR_Q2altw)}}

```
In [261]: corr_q2altw = observa()
          einfo = errinfo()
          einfo.addEnsemble(0, Stau=1.5)
          corr_q2altw.primary_observable(0,'$Q2$ Alt Winding', [0], ['R0'], [MCtimealtw.tolist

          [Q2altw, eQ2altw]= corr_q2altw.vwerr(errinfo=einfo)
          [tauQ2altw, etauQ2altw] = corr_q2altw.tauint()
          tauQ2altw = tauQ2altw[0][0][0]
          etauQ2altw = etauQ2altw[0][0][0]

          print(corr_q2altw.vwerr(plot=True,errinfo=einfo))

          printwe(Q2altw,eQ2altw)

          tauR_Q2altw = tauQ2altw/tauQ2hmc
          etauR_Q2altw = tauR_Q2altw *  np.sqrt( (etauQ2hmc/tauQ2hmc)**2.0 + (etauQ2altw/tauQ2
```

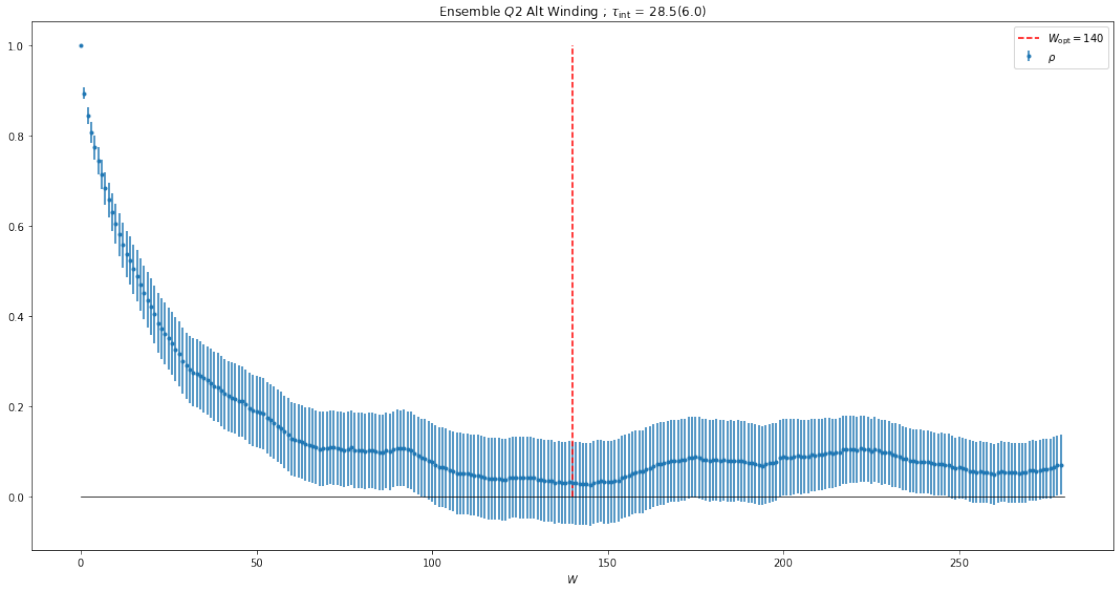



```
[0.7607011398656379, 0.04046302417028977]
0.761(40)
```

1.6.3 Alternating instanton

$Q^2 =$ {{printwe(Q2alti, eQ2alti)}}
$\tau_{int,Q^2} =$ {{printwe(tauQ2alti, etauQ2alti)}}
$\tau_{int}/\tau_{int}^{(HMC)} =$ {{printwe(tauR_Q2alti, etauR_Q2alti)}}

```
In [263]: corr_q2alti = observa()
          einfo = errinfo()
          einfo.addEnsemble(0, Stau=1.5)
          corr_q2alti.primary_observable(0,'$Q2$ Alt Winding', [0], ['R0'], [MCtimealti.tolist

          [Q2alti, eQ2alti]= corr_q2alti.vwerr(errinfo=einfo)
          [tauQ2alti, etauQ2alti] = corr_q2alti.tauint()
          tauQ2alti = tauQ2alti[0][0][0]
          etauQ2alti = etauQ2alti[0][0][0]

          print(corr_q2alti.vwerr(plot=True,errinfo=einfo))

          printwe(Q2alti,eQ2alti)

          tauR_Q2alti = tauQ2alti/tauQ2hmc
          etauR_Q2alti = tauR_Q2alti *  np.sqrt( (etauQ2hmc/tauQ2hmc)**2.0 + (etauQ2alti/tauQ2
```

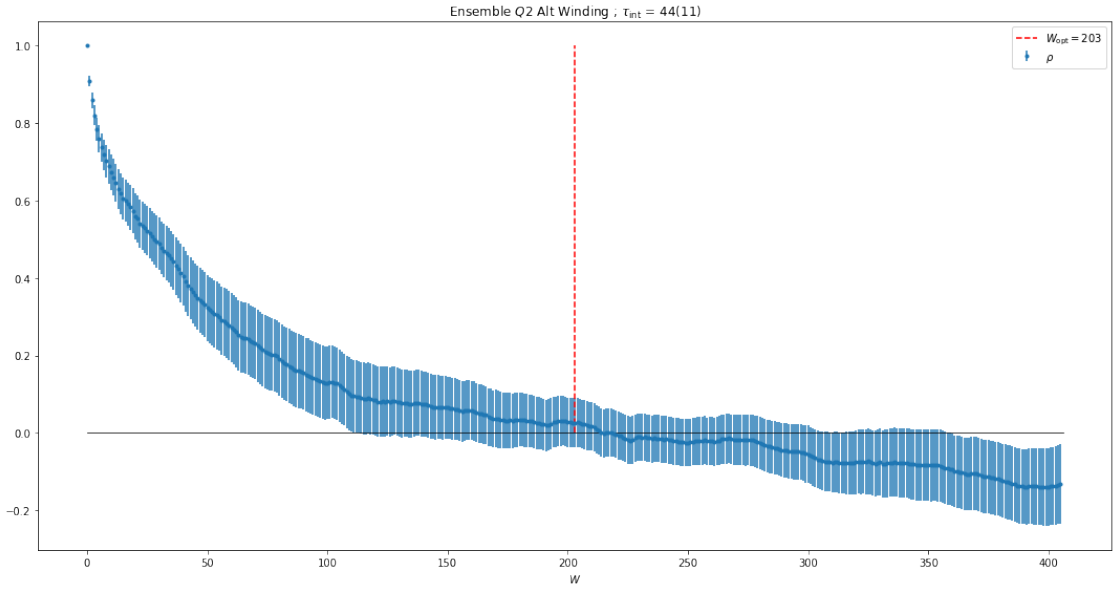



```
[0.7714815003631615, 0.09498620447061464]
0.771(95)
```

1.6.4 Double Winding

$Q^2 =$ {{printwe(Q2doubaltw, eQ2doubaltw)}}
$\tau_{int,Q^2} =$ {{printwe(tauQ2doubaltw, etauQ2doubaltw)}}
$\tau_{int}/\tau_{int}^{(HMC)} =$ {{printwe(tauR_Q2doubaltw, etauR_Q2doubaltw)}}

```
In [265]: corr_q2doubaltw = observa()
          einfo = errinfo()
          einfo.addEnsemble(0, Stau=1.5)
          corr_q2doubaltw.primary_observable(0,'$Q2$ Alt Winding', [0], ['R0'], [MCtimedoubaltw

          [Q2doubaltw, eQ2doubaltw]= corr_q2doubaltw.vwerr(errinfo=einfo)
          [tauQ2doubaltw, etauQ2doubaltw] = corr_q2doubaltw.tauint()
          tauQ2doubaltw = tauQ2doubaltw[0][0][0]
          etauQ2doubaltw = etauQ2doubaltw[0][0][0]

          print(corr_q2doubaltw.vwerr(plot=True,errinfo=einfo))

          printwe(Q2doubaltw,eQ2doubaltw)

          tauR_Q2doubaltw = tauQ2doubaltw/tauQ2hmc
          etauR_Q2doubaltw = tauR_Q2doubaltw *  np.sqrt( (etauQ2hmc/tauQ2hmc)**2.0 + (etauQ2dou
```

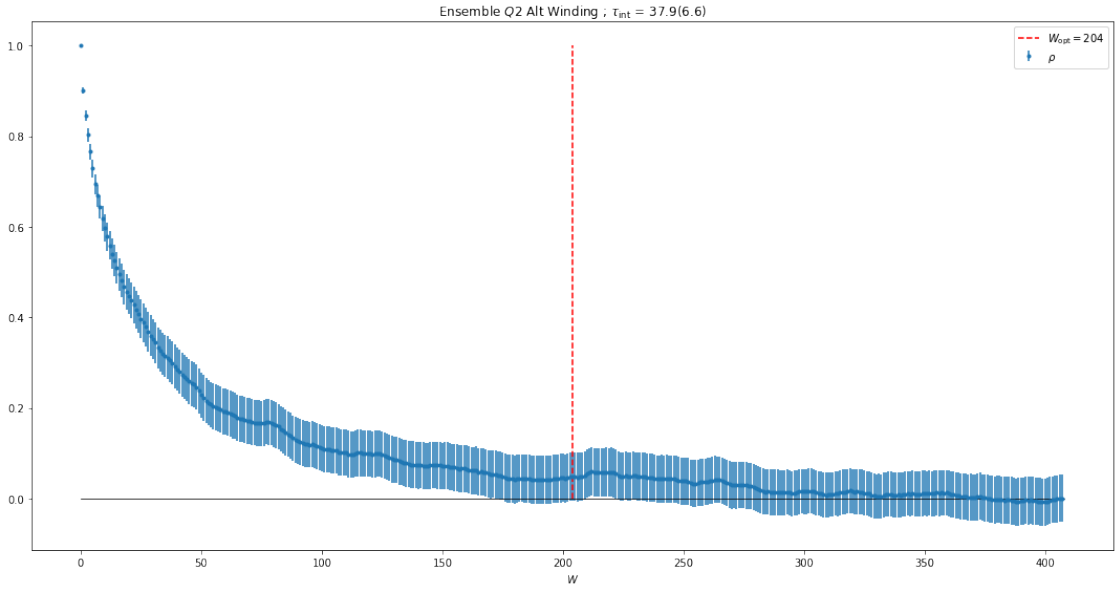



```
[0.9304809269445213, 0.1285291691817063]
0.93(13)
```

1.6.5 Alternating Winding 2

$Q^2 =$ {{printwe(Q2altw2, eQ2altw2)}}
$\tau_{int,Q^2} =$ {{printwe(tauQ2altw2, etauQ2altw2)}}
$\tau_{int}/\tau_{int}^{(HMC)} =$ {{printwe(tauR_Q2altw2, etauR_Q2altw2)}}

```
In [362]: corr_q2altw2 = observa()
          einfo = errinfo()
          einfo.addEnsemble(0, Stau=1.5)
          corr_q2altw2.primary_observable(0,'$Q2$ Alt Winding', [0], ['R0'], [MCtimealtw2.tolis

          [Q2altw2, eQ2altw2]= corr_q2altw2.vwerr(errinfo=einfo)
          [tauQ2altw2, etauQ2altw2] = corr_q2altw2.tauint()
          tauQ2altw2 = tauQ2altw2[0][0][0]
          etauQ2altw2 = etauQ2altw2[0][0][0]

          print(corr_q2altw2.vwerr(plot=True,errinfo=einfo))

          printwe(Q2altw2,eQ2altw2)

          tauR_Q2altw2 = tauQ2altw2/tauQ2hmc
          etauR_Q2altw2 = tauR_Q2altw2 *  np.sqrt( (etauQ2hmc/tauQ2hmc)**2.0 + (etauQ2altw2/tau
```

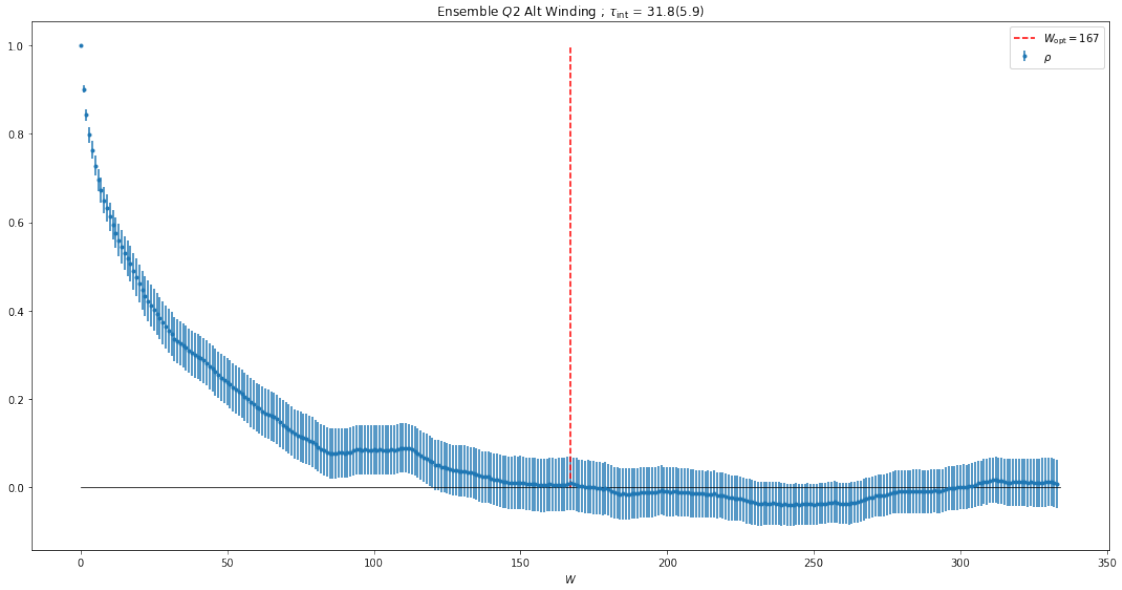



```
[0.8114269888738852, 0.07800976752252282]
0.811(78)
```

1.6.6 Alternating Winding 3

$Q^2 =$ {{printwe(Q2altw3, eQ2altw3)}}
$\tau_{int,Q^2} =$ {{printwe(tauQ2altw3, etauQ2altw3)}}
$\tau_{int}/\tau_{int}^{(HMC)} =$ {{printwe(tauR_Q2altw3, etauR_Q2altw3)}}

```
In [363]: corr_q2altw3 = observa()
          einfo = errinfo()
          einfo.addEnsemble(0, Stau=1.5)
          corr_q2altw3.primary_observable(0,'$Q2$ Alt Winding', [0], ['R0'], [MCtimealtw3.tolis

          [Q2altw3, eQ2altw3]= corr_q2altw3.vwerr(errinfo=einfo)
          [tauQ2altw3, etauQ2altw3] = corr_q2altw3.tauint()
          tauQ2altw3 = tauQ2altw3[0][0][0]
          etauQ2altw3 = etauQ2altw3[0][0][0]

          print(corr_q2altw3.vwerr(plot=True,errinfo=einfo))

          printwe(Q2altw3,eQ2altw3)

          tauR_Q2altw3 = tauQ2altw3/tauQ2hmc
          etauR_Q2altw3 = tauR_Q2altw3 *  np.sqrt( (etauQ2hmc/tauQ2hmc)**2.0 + (etauQ2altw3/tau
```




```
[0.8830498775128622, 0.08261842332889216]
0.883(83)
```

1.6.7 Binning

```
In [143]: errors = []
          ns = []

          burn_in = 0 #how many initial states to discard
          discard = 1 #pick 1 every discard number of states

          gls = (topchargeHMC**2).tolist()

          for n in range(1,1001):

              gls2=[]


              if( (np.double(len(gls))/np.double(n)).is_integer() == True  ):
                  ns.append(n);
                  for i in range(int(len(gls)/n)):

                      med = 0.0
                      for j in range(0,n):
                          med += gls[n*i+j]

                      gls2.append(med/n)
```

```
            #gls2 = np.double(med)/np.double(n)
        #gls3.append(gls2)


        value = np.mean(np.asarray(gls2[burn_in::discard]))
        evalue = np.std(np.asarray(gls2[burn_in::discard]))/np.sqrt(len(gls2[burn_in
        errors.append(evalue)
        #disc.append(abs(mom3.evalf()-value)/evalue)

    print("{}% completed!".format(100.0*n/400.0), end='\r')

errorshmc = (errors)
nshmc = (ns)


errors = []
ns = []

gls = (topchargealtw**2).tolist()

for n in range(1,1001):

    gls2=[]


    if( (np.double(len(gls))/np.double(n)).is_integer() == True  ):
        ns.append(n);
        for i in range(int(len(gls)/n)):

            med = 0.0
            for j in range(0,n):
                med += gls[n*i+j]

            gls2.append(med/n)
            #gls2 = np.double(med)/np.double(n)
        #gls3.append(gls2)


        value = np.mean(np.asarray(gls2[burn_in::discard]))
        evalue = np.std(np.asarray(gls2[burn_in::discard]))/np.sqrt(len(gls2[burn_in
        errors.append(evalue)
        #disc.append(abs(mom3.evalf()-value)/evalue)

    print("{}% completed!".format(100.0*n/400.0), end='\r')

errorsaltw = (errors)
nsaltw = (ns)
```

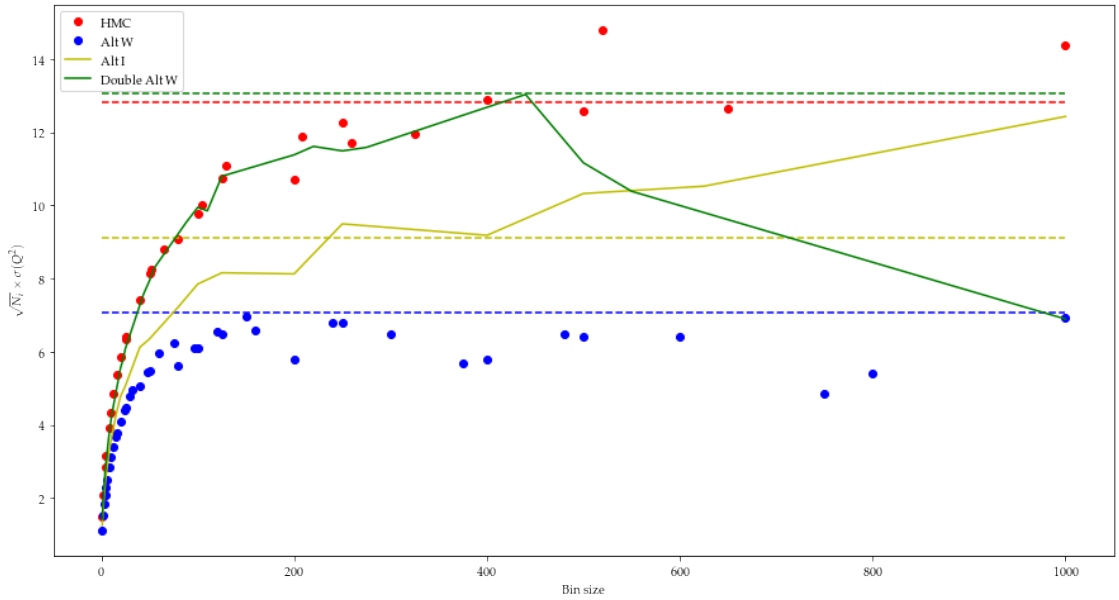
```
errors = []
ns = []

gls = (topchargealti**2).tolist()


for n in range(1,1001):

    gls2=[]


    if( (np.double(len(gls))/np.double(n)).is_integer() == True  ):
        ns.append(n);
        for i in range(int(len(gls)/n)):

            med = 0.0
            for j in range(0,n):
                med += gls[n*i+j]

            gls2.append(med/n)
            #gls2 = np.double(med)/np.double(n)
        #gls3.append(gls2)


        value = np.mean(np.asarray(gls2[burn_in::discard]))
        evalue = np.std(np.asarray(gls2[burn_in::discard]))/np.sqrt(len(gls2[burn_in
        errors.append(evalue)
        #disc.append(abs(mom3.evalf()-value)/evalue)

    print("{}% completed!".format(100.0*n/400.0), end='\r')

errorsalti = (errors)
nsalti = (ns)



errors = []
ns = []

gls = (topchargedoubaltw**2).tolist()

for n in range(1,1001):

    gls2=[]


    if( (np.double(len(gls))/np.double(n)).is_integer() == True  ):
```

```
                ns.append(n);
                for i in range(int(len(gls)/n)):

                    med = 0.0
                    for j in range(0,n):
                        med += gls[n*i+j]

                    gls2.append(med/n)
                    #gls2 = np.double(med)/np.double(n)
                #gls3.append(gls2)


                value = np.mean(np.asarray(gls2[burn_in::discard]))
                evalue = np.std(np.asarray(gls2[burn_in::discard]))/np.sqrt(len(gls2[burn_in
                errors.append(evalue)
                #disc.append(abs(mom3.evalf()-value)/evalue)

            print("{}% completed!".format(100.0*n/400.0), end='\r')

        errorsdoubaltw = (errors)
        nsdoubaltw = (ns)
```

```
250.0% completed!!
```

```
In [144]: plt.plot(nshmc, np.sqrt(Nhmc)*np.array(errorshmc),'ro', label="HMC")
          plt.plot([0,1000],[np.sqrt(Nhmc)*eqhmc,np.sqrt(Nhmc)*eqhmc], 'r--')

          plt.plot(nsaltw, np.sqrt(Naltw)*np.array(errorsaltw),'bo', label="Alt W")
          plt.plot([0,1000],[np.sqrt(Naltw)*eqaltw,np.sqrt(Naltw)*eqaltw], 'b--')

          plt.plot(nsalti, np.sqrt(Nalti)*np.array(errorsalti),'y-', label="Alt I")
          plt.plot([0,1000],[np.sqrt(Nalti)*eqalti,np.sqrt(Nalti)*eqalti], 'y--')

          plt.plot(nsdoubaltw, np.sqrt(Ndoubaltw)*np.array(errorsdoubaltw),'g-', label="Double
          plt.plot([0,1000],[np.sqrt(Ndoubaltw)*eqdoubaltw,np.sqrt(Ndoubaltw)*eqdoubaltw], 'g-

          plt.legend()

          plt.ylabel("$\sqrt{N_i} \\times \sigma (Q^2)$")
          plt.xlabel("Bin size")
```

```
Out[144]: Text(0.5,0,'Bin size')
```



1.7 Q^4

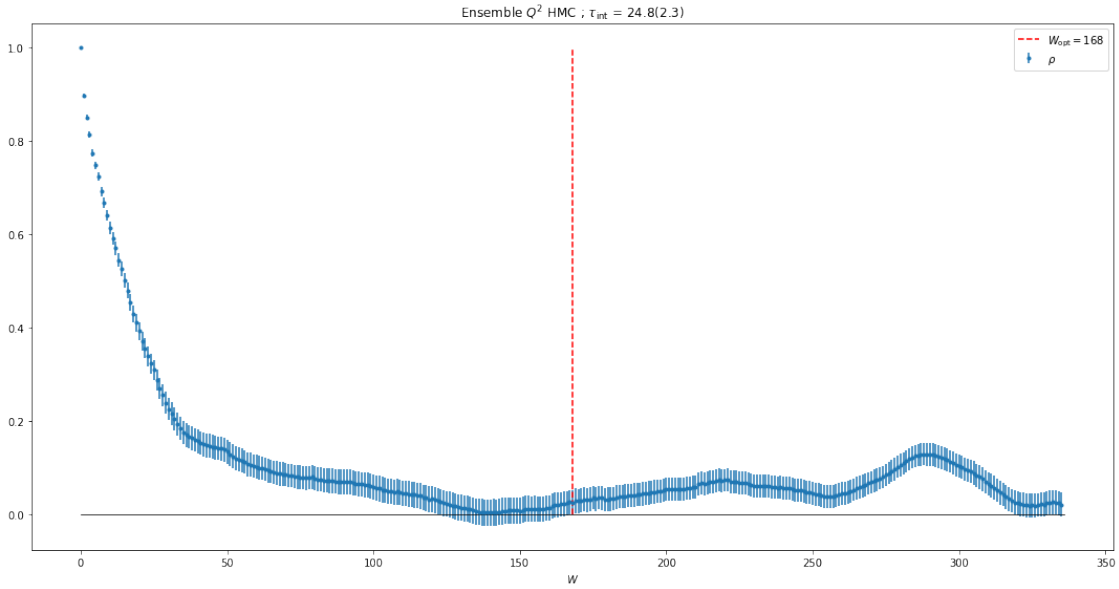
Algorithm	Statistics	Q^4	τ_{int}	$\tau_{\text{int}}/\tau_{\text{int}}^{(\text{HMC})}$
HMC	{{Nhmc}}	{{printwe(Q4hmc,eQ4hmc)}}	{{printwe(tauQ4hmc, etauQ4hmc)}}	-
Alt Winding	{{Naltw}}	{{printwe(Q4altw,eQ4altw)}}	{{printwe(tauQ4altw, etauQ4altw)}}	{{printwe(tauR_Q4altw, etauR_Q4altw)}}
Alt Instanton	{{Nalti}}	{{printwe(Q4alti,eQ4alti)}}	{{printwe(tauQ4alti, etauQ4alti)}}	{{printwe(tauR_Q4alti, etauR_Q4alti)}}
Alt Double Winding	{{Ndoubaltw}}	{{printwe(Q4doubaltw,eQ4doubaltw)}}	{{printwe(tauQ4doubaltw, etauQ4doubaltw)}}	{{printwe(tauR_Q4doubaltw, etauR_Q4doubaltw)}}
Alt Winding 2	{{Naltw2}}	{{printwe(Q4altw2,eQ4altw2)}}	{{printwe(tauQ4altw2, etauQ4altw2)}}	{{printwe(tauR_Q4altw2, etauR_Q4altw2)}}
Alt Winding 3	{{Naltw3}}	{{printwe(Q4altw3,eQ4altw3)}}	{{printwe(tauQ4altw3, etauQ4altw3)}}	{{printwe(tauR_Q4altw3, etauR_Q4altw3)}}

1.7.1 HMC

$P =$ {{printwe(Q4hmc, eQ4hmc)}}
$\tau_{int,Q^4} =$ {{printwe(tauQ4hmc, etauQ4hmc)}}

```
In [269]: corr_q4hmc = observa()
          einfo = errinfo()
          einfo.addEnsemble(0, Stau=1.5)
          corr_q4hmc.primary_observable(0,'$Q^2$ HMC', [0], ['R0'], [MCtimehmc.tolist()], [(to
```

```
[Q4hmc, eQ4hmc]= corr_q4hmc.vwerr(errinfo=einfo)
[tauQ4hmc, etauQ4hmc] = corr_q4hmc.tauint()
tauQ4hmc = tauQ4hmc[0][0][0]
etauQ4hmc = etauQ4hmc[0][0][0]

print(corr_q4hmc.vwerr(plot=True,errinfo=einfo))

printwe(Q4hmc,eQ4hmc)
```



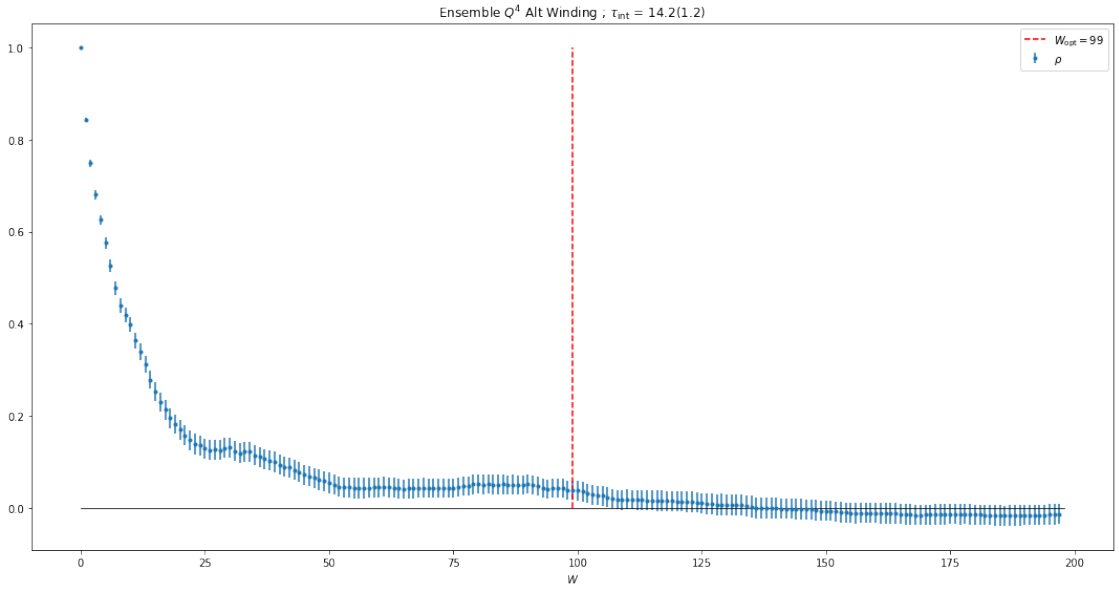

```
[2.4769506093459195, 0.2868759866872537]
2.48(29)
```

1.7.2 Alternating Winding

$Q^4 =$ {{printwe(Q4altw, eQ4altw)}}
$\tau_{int,Q^4} =$ {{printwe(tauQ4altw, etauQ4altw)}}
$\tau_{int}/\tau_{int}^{(HMC)} =$ {{printwe(tauR_Q4altw, etauR_Q4altw)}}

```
In [272]: corr_q4altw = observa()
          einfo = errinfo()
          einfo.addEnsemble(0, Stau=1.5)
          corr_q4altw.primary_observable(0,'$Q^4$ Alt Winding', [0], ['R0'], [MCtimealtw.tolis

          [Q4altw, eQ4altw]= corr_q4altw.vwerr(errinfo=einfo)
          [tauQ4altw, etauQ4altw] = corr_q4altw.tauint()
          tauQ4altw = tauQ4altw[0][0][0]
          etauQ4altw = etauQ4altw[0][0][0]
```

```
print(corr_q4altw.vwerr(plot=True,errinfo=einfo))

printwe(Q4altw,eQ4altw)

tauR_Q4altw = tauQ4altw/tauQ4hmc
etauR_Q4altw = tauR_Q4altw *  np.sqrt( (etauQ4hmc/tauQ4hmc)**2.0 + (etauQ4altw/tauQ4a
```



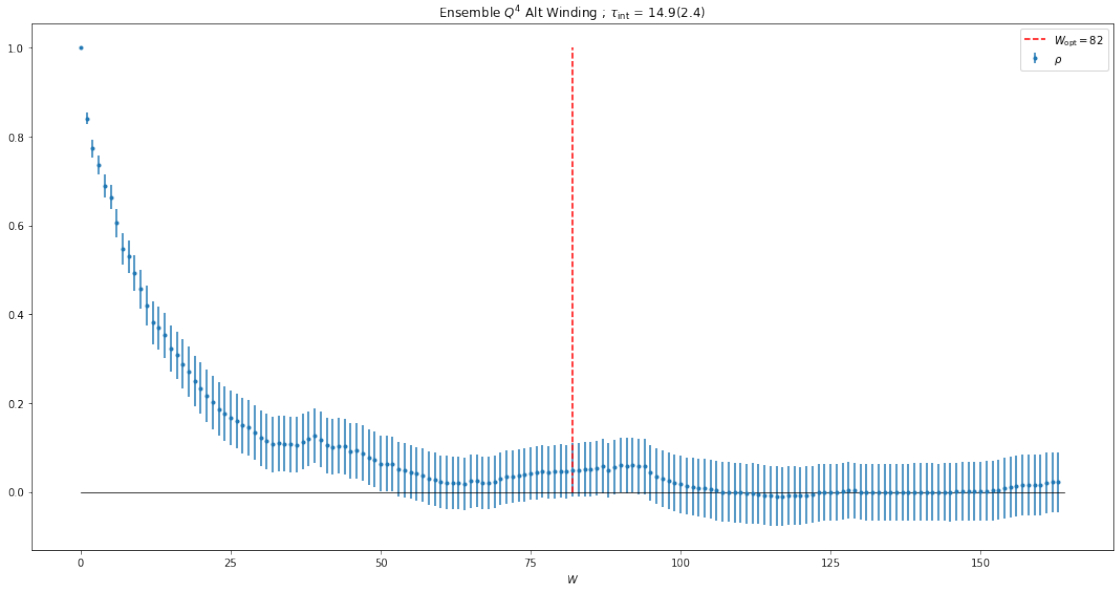

```
[2.0625923840122407, 0.21076036159963832]
2.06(21)
```

1.7.3 Alternating instanton

$Q^4 =$ {{printwe(Q4alti, eQ4alti)}}
$\tau_{int,Q^4} =$ {{printwe(tauQ4alti, etauQ4alti)}}
$\tau_{int}/\tau_{int}^{(HMC)} =$ {{printwe(tauR_Q4alti, etauR_Q4alti)}}

```
In [274]: corr_q4alti = observa()
          einfo = errinfo()
          einfo.addEnsemble(0, Stau=1.5)
          corr_q4alti.primary_observable(0,'$Q^4$ Alt Winding', [0], ['R0'], [MCtimealti.tolist

          [Q4alti, eQ4alti]= corr_q4alti.vwerr(errinfo=einfo)
          [tauQ4alti, etauQ4alti] = corr_q4alti.tauint()
          tauQ4alti = tauQ4alti[0][0][0]
          etauQ4alti = etauQ4alti[0][0][0]
```

```
print(corr_q4alti.vwerr(plot=True,errinfo=einfo))

printwe(Q4alti,eQ4alti)

tauR_Q4alti = tauQ4alti/tauQ4hmc
etauR_Q4alti = tauR_Q4alti *  np.sqrt( (etauQ4hmc/tauQ4hmc)**2.0 + (etauQ4alti/tauQ4a
```



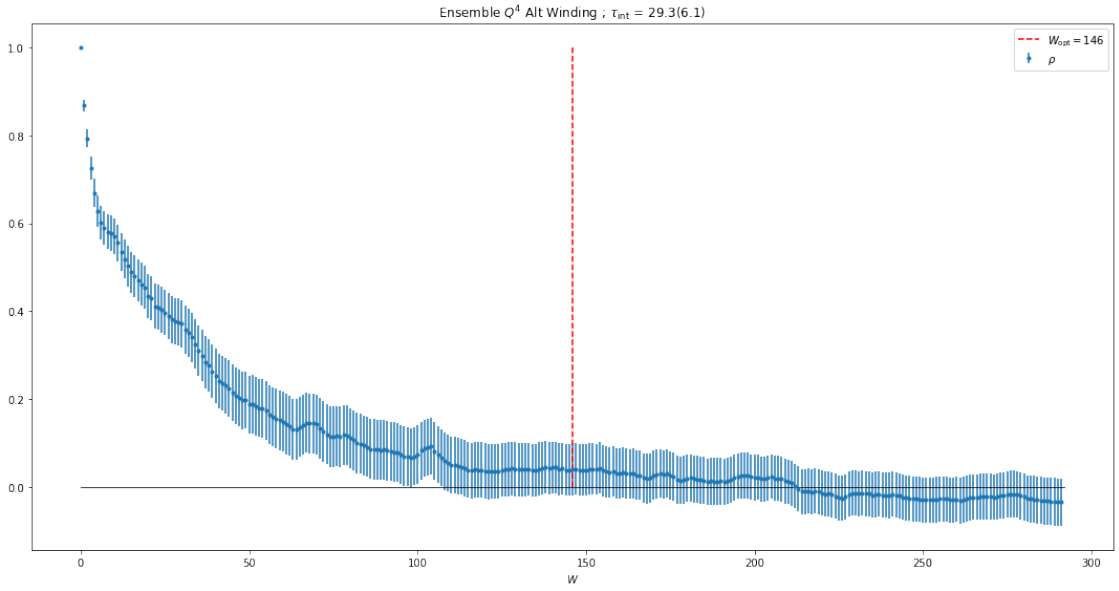

```
[2.1666092092307054, 0.4454383867354963]
2.17(45)
```

1.7.4 Double Winding

$Q^4 =$ {{printwe(Q4doubaltw, eQ4doubaltw)}}
$\tau_{int,Q^4} =$ {{printwe(tauQ4doubaltw, etauQ4doubaltw)}}
$\tau_{int}/\tau_{int}^{(HMC)} =$ {{printwe(tauR_Q4doubaltw, etauR_Q4doubaltw)}}

```
In [343]: corr_q4doubaltw = observa()
          einfo = errinfo()
          einfo.addEnsemble(0, Stau=1.5)
          corr_q4doubaltw.primary_observable(0,'$Q^4$ Alt Winding', [0], ['RO'], [MCtimedoubal

          [Q4doubaltw, eQ4doubaltw]= corr_q4doubaltw.vwerr(errinfo=einfo)
          [tauQ4doubaltw, etauQ4doubaltw] = corr_q4doubaltw.tauint()
          tauQ4doubaltw = tauQ4doubaltw[0][0][0]
          etauQ4doubaltw = etauQ4doubaltw[0][0][0]

          print(corr_q4doubaltw.vwerr(plot=True,errinfo=einfo))
```

```
printwe(Q4doubaltw,eQ4doubaltw)

tauR_Q4doubaltw = tauQ4doubaltw/tauQ4hmc
etauR_Q4doubaltw = tauR_Q4doubaltw *  np.sqrt( (etauQ4hmc/tauQ4hmc)**2.0 + (etauQ4dou
```



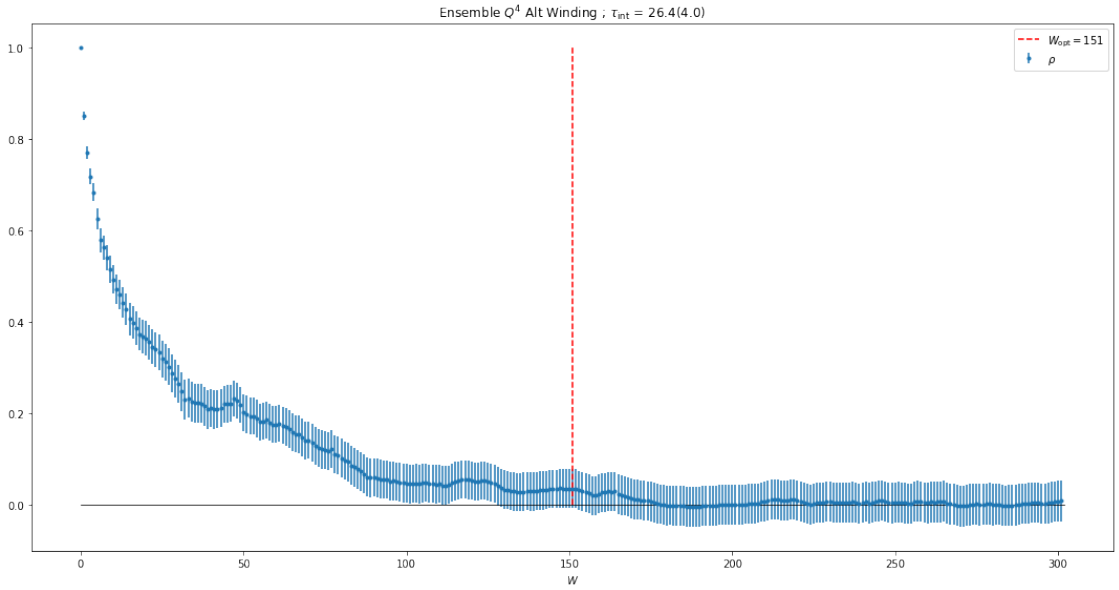

```
[2.9015676860650816, 0.6608553493941995]
2.90(66)
```

1.7.5 Alternating Winding 2

Q^4 = {{printwe(Q4altw2, eQ4altw2)}}
τ_{int,Q^4} = {{printwe(tauQ4altw2, etauQ4altw2)}}
$\tau_{int}/\tau_{int}^{(HMC)}$ = {{printwe(tauR_Q4altw2, etauR_Q4altw2)}}

```
In [364]: corr_q4altw2 = observa()
          einfo = errinfo()
          einfo.addEnsemble(0, Stau=1.5)
          corr_q4altw2.primary_observable(0,'$Q^4$ Alt Winding', [0], ['R0'], [MCtimealtw2.toli

          [Q4altw2, eQ4altw2]= corr_q4altw2.vwerr(errinfo=einfo)
          [tauQ4altw2, etauQ4altw2] = corr_q4altw2.tauint()
          tauQ4altw2 = tauQ4altw2[0][0][0]
          etauQ4altw2 = etauQ4altw2[0][0][0]

          print(corr_q4altw2.vwerr(plot=True,errinfo=einfo))
```

```
printwe(Q4altw2,eQ4altw2)

tauR_Q4altw2 = tauQ4altw2/tauQ4hmc
etauR_Q4altw2 = tauR_Q4altw2 *  np.sqrt( (etauQ4hmc/tauQ4hmc)**2.0 + (etauQ4altw2/tau
```



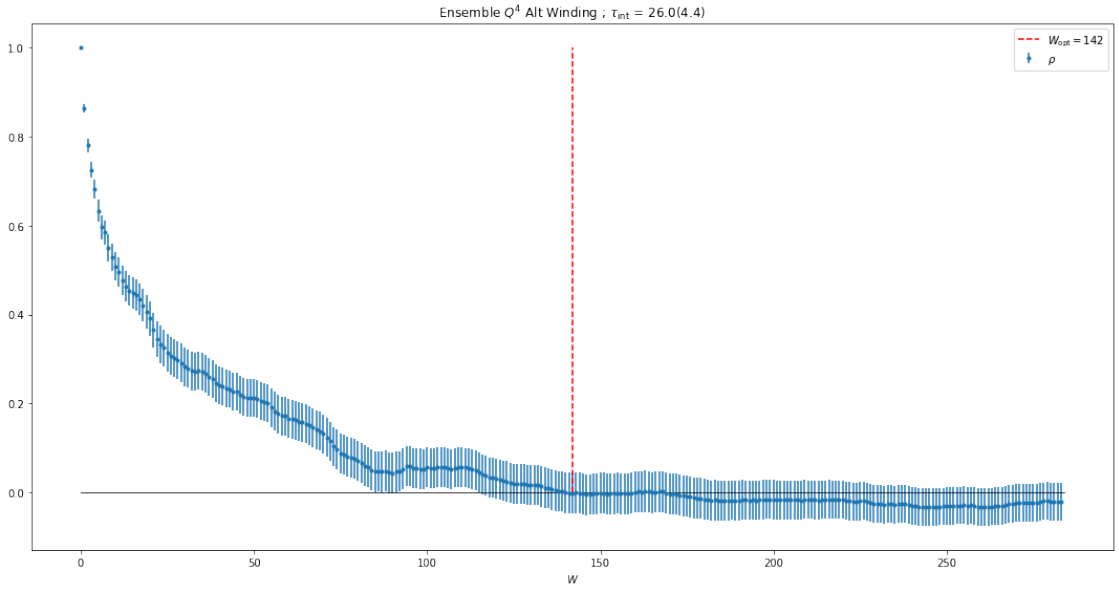

```
[2.419397741376706, 0.46031498784845537]
2.42(46)
```

1.7.6 Alternating Winding 3

$Q^4 = \{\{\text{printwe(Q4altw3, eQ4altw3)}\}\}$
$\tau_{int,Q^4} = \{\{\text{printwe(tauQ4altw3, etauQ4altw3)}\}\}$
$\tau_{int}/\tau_{int}^{(HMC)} = \{\{\text{printwe(tauR\_Q4altw3, etauR\_Q4altw3)}\}\}$

```
In [365]: corr_q4altw3 = observa()
          einfo = errinfo()
          einfo.addEnsemble(0, Stau=1.5)
          corr_q4altw3.primary_observable(0,'$Q^4$ Alt Winding', [0], ['R0'], [MCtimealtw3.toli

          [Q4altw3, eQ4altw3]= corr_q4altw3.vwerr(errinfo=einfo)
          [tauQ4altw3, etauQ4altw3] = corr_q4altw3.tauint()
          tauQ4altw3 = tauQ4altw3[0][0][0]
          etauQ4altw3 = etauQ4altw3[0][0][0]

          print(corr_q4altw3.vwerr(plot=True,errinfo=einfo))

          printwe(Q4altw3,eQ4altw3)
```

```
tauR_Q4altw3 = tauQ4altw3/tauQ4hmc
etauR_Q4altw3 = tauR_Q4altw3 *  np.sqrt( (etauQ4hmc/tauQ4hmc)**2.0 + (etauQ4altw3/tau
```




```
[2.488788196015555, 0.495612964165533]
2.49(50)
```

1.7.7 Binning

```
In [149]: errors = []
          ns = []

          burn_in = 0 #how many initial states to discard
          discard = 1 #pick 1 every discard number of states

          gls = (topchargeHMC**4).tolist()

          for n in range(1,1001):

              gls2=[]


              if( (np.double(len(gls))/np.double(n)).is_integer() == True  ):
                  ns.append(n);
                  for i in range(int(len(gls)/n)):

                      med = 0.0
```

```
            for j in range(0,n):
                med += gls[n*i+j]

            gls2.append(med/n)
            #gls2 = np.double(med)/np.double(n)
        #gls3.append(gls2)


        value = np.mean(np.asarray(gls2[burn_in::discard]))
        evalue = np.std(np.asarray(gls2[burn_in::discard]))/np.sqrt(len(gls2[burn_in
        errors.append(evalue)
        #disc.append(abs(mom3.evalf()-value)/evalue)

    print("{}% completed!".format(100.0*n/400.0), end='\r')

errorshmc = (errors)
nshmc = (ns)


errors = []
ns = []

gls = (topchargealtw**4).tolist()

for n in range(1,1001):

    gls2=[]


    if( (np.double(len(gls))/np.double(n)).is_integer() == True  ):
        ns.append(n);
        for i in range(int(len(gls)/n)):

            med = 0.0
            for j in range(0,n):
                med += gls[n*i+j]

            gls2.append(med/n)
            #gls2 = np.double(med)/np.double(n)
        #gls3.append(gls2)


        value = np.mean(np.asarray(gls2[burn_in::discard]))
        evalue = np.std(np.asarray(gls2[burn_in::discard]))/np.sqrt(len(gls2[burn_in
        errors.append(evalue)
        #disc.append(abs(mom3.evalf()-value)/evalue)

    print("{}% completed!".format(100.0*n/400.0), end='\r')
```

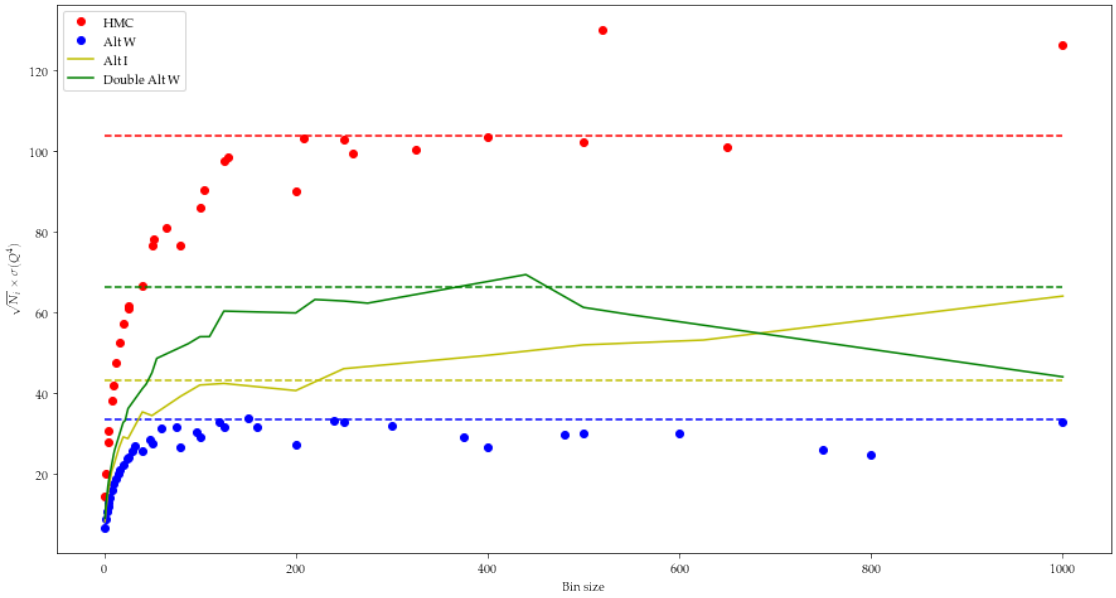
```
errorsaltw = (errors)
nsaltw = (ns)


errors = []
ns = []

gls = (topchargealti**4).tolist()


for n in range(1,1001):

    gls2=[]


    if( (np.double(len(gls))/np.double(n)).is_integer() == True  ):
        ns.append(n);
        for i in range(int(len(gls)/n)):

            med = 0.0
            for j in range(0,n):
                med += gls[n*i+j]

            gls2.append(med/n)
            #gls2 = np.double(med)/np.double(n)
        #gls3.append(gls2)


        value = np.mean(np.asarray(gls2[burn_in::discard]))
        evalue = np.std(np.asarray(gls2[burn_in::discard]))/np.sqrt(len(gls2[burn_in
        errors.append(evalue)
        #disc.append(abs(mom3.evalf()-value)/evalue)

    print("{}% completed!".format(100.0*n/400.0), end='\r')

errorsalti = (errors)
nsalti = (ns)



errors = []
ns = []

gls = (topchargedoubaltw**4).tolist()

for n in range(1,1001):
```

```
            gls2=[]


            if( (np.double(len(gls))/np.double(n)).is_integer() == True  ):
                ns.append(n);
                for i in range(int(len(gls)/n)):

                    med = 0.0
                    for j in range(0,n):
                        med += gls[n*i+j]

                    gls2.append(med/n)
                    #gls2 = np.double(med)/np.double(n)
                #gls3.append(gls2)


                value = np.mean(np.asarray(gls2[burn_in::discard]))
                evalue = np.std(np.asarray(gls2[burn_in::discard]))/np.sqrt(len(gls2[burn_in
                errors.append(evalue)
                #disc.append(abs(mom3.evalf()-value)/evalue)

            print("{}% completed!".format(100.0*n/400.0), end='\r')

        errorsdoubaltw = (errors)
        nsdoubaltw = (ns)
```

```
250.0% completed!!
```

```
In [150]: plt.plot(nshmc, np.sqrt(Nhmc)*np.array(errorshmc),'ro', label="HMC")
          plt.plot([0,1000],[np.sqrt(Nhmc)*eqhmc,np.sqrt(Nhmc)*eqhmc], 'r--')

          plt.plot(nsaltw, np.sqrt(Naltw)*np.array(errorsaltw),'bo', label="Alt W")
          plt.plot([0,1000],[np.sqrt(Naltw)*eqaltw,np.sqrt(Naltw)*eqaltw], 'b--')

          plt.plot(nsalti, np.sqrt(Nalti)*np.array(errorsalti),'y-', label="Alt I")
          plt.plot([0,1000],[np.sqrt(Nalti)*eqalti,np.sqrt(Nalti)*eqalti], 'y--')

          plt.plot(nsdoubaltw, np.sqrt(Ndoubaltw)*np.array(errorsdoubaltw),'g-', label="Double
          plt.plot([0,1000],[np.sqrt(Ndoubaltw)*eqdoubaltw,np.sqrt(Ndoubaltw)*eqdoubaltw], 'g-

          plt.legend()

          plt.ylabel("$\sqrt{N_i} \\times \sigma (Q^4)$")
          plt.xlabel("Bin size")
```

```
Out[150]: Text(0.5,0,'Bin size')
```


HMC
Alt W
Alt I
Double Alt W
120
100
80
60
40
20
0
200
400
600
800
1000
Bin size