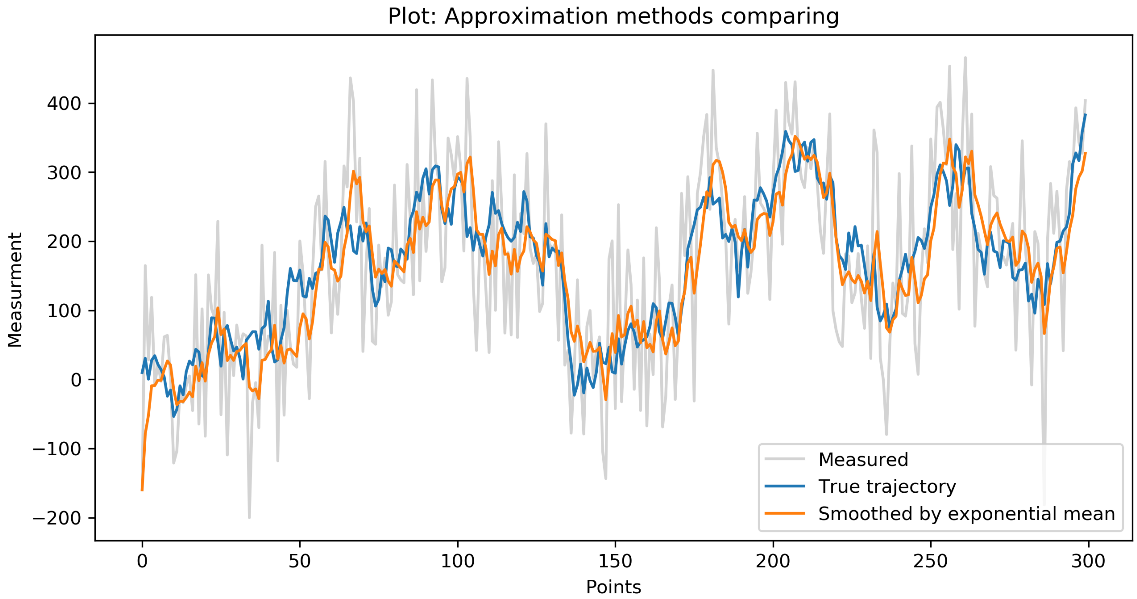
Assignment 3 Part I

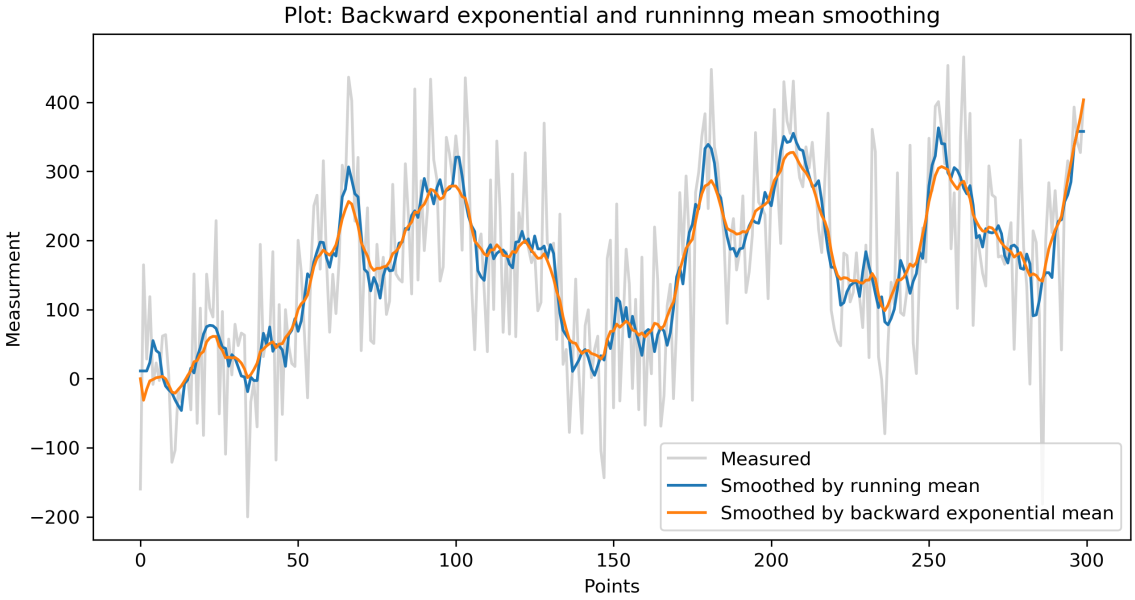
In [9]:

```
#Assignment 2 Part 6
figure(num=None, figsize=(10, 5), dpi=300, facecolor='w', edgecolor='k')
plt.title('Plot: Approximation methods comparing')
plt.plot(z, label='Measured',c='lightgrey')
plt.plot(x, label='True trajectory')
plt.plot(sm_z, label='Smoothed by exponential mean')
plt.xlabel('Points')
plt.ylabel('Measurment')
plt.legend()
plt.show()
```



In [15]:

```
# Assignment 3
# Relationship between solar radio flux F10.7 and sunspot number
# Team 2:
#      Ekaterina Karmanova
#      Timur Chikichev
#      Yaroslav Okunev
#      Nikita Mikhailovskiy
#
# Skoltech, 04.10.2019

#backward smoothing
sm_z_back = np.zeros(c)
length = len(sm_z_back)
sm_z_back[length-1]=z[length-1]
for i in range (length-2,0,-1):
    sm_z_back[i] = sm_z_back[i+1] + a*(sm_z[i]-sm_z_back[i+1])

figure(num=None, figsize=(10, 5), dpi=300, facecolor='w', edgecolor='k')
plt.title('Plot: Backward exponential and runninng mean smoothing')
plt.plot(z, label='Measured', c='lightgrey')
plt.plot(sm_z_r, label='Smoothed by running mean')
plt.plot(sm_z_back, label='Smoothed by backward exponential mean')
plt.xlabel('Points')
plt.ylabel('Measurment')
plt.legend()
plt.show()
```



In [14]:

```
def devInd(z,x):
    Id=[0,0]
    for i in range(0,len(z)):
                Id[0]+=(z[i]-x[i])**2
    for i in range(0,len(z)-2):
                Id[1]+=(x[i+2]-2*x[i+1]+x[i])**2
    return Id
Idex=devInd(z,sm_z)
print('Indicator for forward exp. smoothing =',Idex)
Idbex=devInd(z,sm_z_back)
print('Indicator for backward exp. smoothing =',Idbex)
```

```
Indicator for forward exp. smoothing = [2126522.1308989194, 442778.4085260
831]
Indicator for backward exp. smoothing = [2409214.250412533, 18484.32082341
0028]
```

In [ ]:

```python

import numpy as np

class Random_walking_model():
    @staticmethod
    def TrueTrajectory(initial, diffs):
        array = [initial]
        for di in diffs:
            array.append(array[-1] + di)
        return np.array(array)

    def generate_signal(self, initial, sigma, n):
        diffs = np.random.normal(0, sigma, n-1)
        self.x = Random_walking_model.TrueTrajectory(initial, diffs)

    def __init__(self, initial, sigma, n):
        self.sigma = sigma
        self.generate_signal(initial, sigma, n)

def rvm_alpha(sig):
    '''linear regression koefficient(alpha)
    for random valking model process'''
    psi = sig[0]**2 / sig[1]**2

    alpha = (-psi + np.sqrt(psi**2 + 4 * psi)) / 2
    print('Optimal smoothing coefficient=', alpha)
    return alpha

```

File: smoothing.py

```python

# Assignment 3
# Determining and removing drawbacks of exponential and running mean.
# Team 2:
#     Ekaterina Karmanova
#     Timur Chikichev
#     Iaroslav Okunevich
#     Nikita Mikhailovskiy
#
# Skoltech, 04.10.2019

import numpy as np

class Smooth():
    def __init__(self, z):
        self.z = z

class RunningMean(Smooth):
    @staticmethod
    def smooth(z, m):
        x = np.zeros(len(z))
        sumFirst = 0
        sumLast = 0
        step = int((m-1)/2)
        for i in range(0, step):
            sumFirst += z[i]
            sumLast += z[len(z)-i-1]
        for i in range(step, len(z)-step):
            for j in range(i-step, i+step+1):
                x[i] += z[j]
            x[i] /= m
        for i in range(0, step):
            x[i] = sumFirst/step
            x[len(x)-i-1] = sumLast/step
        return x

    @staticmethod
    def window_width(alpha):
        return round((2-alpha)/alpha)

    # def calc_window_width(self, alpha):
    #     self.m = RunningMean.window_width(alpha)

    def __init__(self, z, alpha):
        super().__init__(z)
        # Smooth.__init__(z)
        self.m = RunningMean.window_width(alpha)
        # return m

    def run(self, m):
        ret = RunningMean.smooth(self.z, m)
        return ret

class FB_exp_smoothing(Smooth):
    @staticmethod
    def forward(alpfa, z):
        array = np.zeros(len(z))
        array[0] = 10
        for i in range(1, len(z)):
            array[i] = array[i - 1] + alpfa * (z[i] - array[i - 1])
        return array
```

```python

    @staticmethod
    def backward(alpha, fz):
        array = np.zeros_like(fz)
        array[len(fz)-1] = fz[len(fz)-1]
        for i in range(len(fz)-2, -1, -1):
            array[i] = array[i+1]+alpha*(fz[i]-array[i+1])
        return array

    def run_f(self, alpha):
        res = FB_exp_smoothing.forward(alpha, self.z)
        return res

    def run_b(self, f, alpha):
        res = FB_exp_smoothing.backward(alpha, f)
        return res

    def run_fb(self, alpha):
        forward = FB_exp_smoothing.forward(alpha, self.z)
        res = FB_exp_smoothing.backward(alpha, forward)
        return res

```

File: lab3_tr1.py

```python
# Assignment 3
# Determining and removing drawbacks of exponential and running mean.
# Team 2:
#      Ekaterina Karmanova
#      Timur Chikichev
#      Iaroslav Okunevich
#      Nikita Mikhailovskiy
#
# Skoltech, 04.10.2019

import os
try:
  os.chdir(os.path.join(os.getcwd(), 'lab3'))
  print(os.getcwd())
except:
  pass

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure

import smoothing as sm
from walking_model import Random_walking_model, rvm_alpha

class Measurements():
    def __init__(self, x, sigma):
        n = len(x)
        noise = np.random.normal(0, sigma, n)
        self.z = x + noise

    def DevVarInd(self, x):
        #find deviation and variability indicators.
        Id = 0
        Iv = 0
        z = self.z
        for i in range(len(z)):
            Id += (z[i]-x[i])**2
        for i in range(0, len(z)-2):
            Iv += (x[i+2] - 2*x[i+1] + x[i])**2
        return Id, Iv


def draw_plots(plots=[], show = False):
    for pi, plabel in plots:
        plt.plot(pi, label=plabel)

    plt.legend()
    if show:
        plt.show()

def new_plot(title, xl, yl, plots = [], show = False):
    figure(num=None, figsize=(10, 5), dpi=100,
           facecolor='w', edgecolor='k')
    plt.title(title)
    plt.ylabel(yl)
    plt.xlabel(xl)

    draw_plots(plots, show)

#Part II
```

```python
def part2():
    a = np.random.normal(0, np.sqrt(10), 300)
    sig = np.sqrt(500)

    def Trajectory(size, acc, t):
        vel = Random_walking_model.TrueTrajectory(0, acc * t)
        trajectory = Random_walking_model.TrueTrajectory(5, vel[:-1]*t + acc * t * t / 2)

        new_plot('Plot result', 'Measurement', 'Points',
        [
            [acc, 'Acceleration'],
            [vel, 'Velosity'],
            [trajectory, 'Trajectory']
        ], show = False)
        plt.savefig(fname='Trajectory.png')
        return trajectory

    traject = Trajectory(300, a, 0.1)
    Known_vals = Measurements(traject, sig)
    z = Known_vals.z

    RM = sm.RunningMean(z, 1)

    new_plot('Plot result',
            'Measurements', 'Points', [
                [traject, 'Trajectory'],
                [z, 'Measure']
            ], show = False)
    plt.savefig(fname = 'running mean 1.png')

    def eval_params(z, chvals, b, run):
        krits = []
        for ai in chvals:
            r = run(ai)
            ki = Known_vals.DevVarInd(r)
            res = np.sum([ki[i]*b[i] for i in range(2)])
            print('a: {} ki: {} res: {}'.format(
              ai, [ki[i]*b[i] for i in range(2)], res))
            krits.append(res)
        idmin = np.argmin(krits, 0)
        ret = chvals[idmin - 1: idmin + 2]
        return ret, krits[idmin]

    def plot_f(z, chvals, run, aname):
        new_plot('Plot result', 'Number', 'Count')
        plt.plot(z, label='Measure', c='lightgrey')
        for ai in chvals:
            r = run(ai)
            ki = Known_vals.DevVarInd(r)
            res = np.sum([ki[i]*b[i] for i in range(2)])
            plt.plot(r, label='{}, val: {}, res: {}'.format(aname, ai, res))
        plt.legend()
        # plt.show()

    n = 10
    b = [0.02, 0.08]
    chvals = np.round(np.linspace(0, z.shape[0] * 0.4, n))[1:]
    bestm, resm = eval_params(z, chvals, b, RM.run)
    plot_f(z, bestm, RM.run, 'Running mean')
    plt.savefig(fname='Running mean best.png')
```

```python

    Expsmoothing = sm.FB_exp_smoothing(z)

    probe_alpha = np.linspace(0, 1, n + 2)[1:-1]
    best_alpha, resalpha = eval_params(z, probe_alpha, b, Expsmoothing.run_fb)

    print("bestm: {}, best_alpha: {}".format(bestm, best_alpha))
    plot_f(z, best_alpha, Expsmoothing.run_fb, 'exp fb smoothing')

    plt.savefig(fname='exp fb smoothing best.png')

    def proper_choise(m_vals, alpha_vals):
        m = m_vals
        a = alpha_vals

        res = 'exponential' if a < m else 'running mean'
        print("mean:{}, exp:{}".format(m_vals, alpha_vals))
        print(res)

    proper_choise(resm, resalpha)

def main():
    part2()

if __name__ == "__main__":
    main()
```

Assignment 3 Part II - Second trajectory

In [1]:

```
# Assignment 3 Part 2 Trajectory 2
# Relationship between solar radio flux F10.7 and sunspot number
# Team 2:
#     Ekaterina Karmanova
#     Timur Chikichev
#     Yaroslav Okunev
#     Nikita Mikhailovskiy
#
# Skoltech, 04.10.2019
```

In [2]:

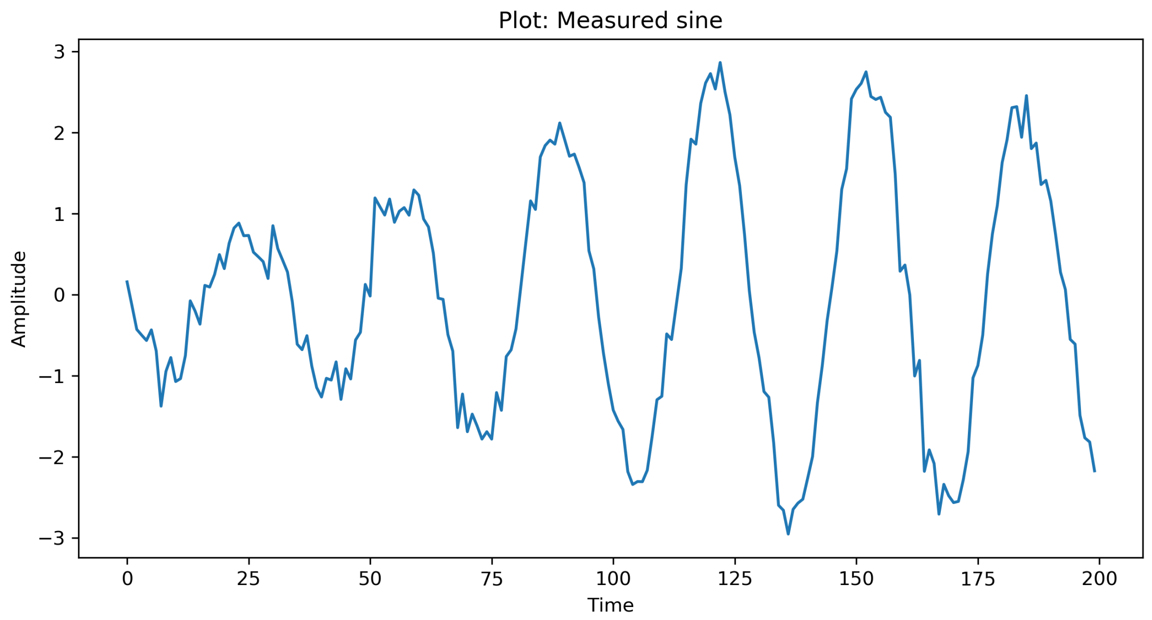
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
```

In [3]:

```
#4 second trajectory
#initial acceleration
A1 = 1
#N. of points
C = 200
#Period counts
T = 32
#Variance
sw = 0.08
s2w = sw**2

#noise
w = np.random.normal(0,sw,C)
#Angle frequency
afreq = 2*np.pi/T

#determine acceleration
A = np.zeros(C)
A[0] = A1
for i in range (1,C):
    A[i] = A[i-1] + w[i]

X = np.zeros(C)
for i in range (C):
    X[i] = A[i]*np.sin(afreq*i+3)
```

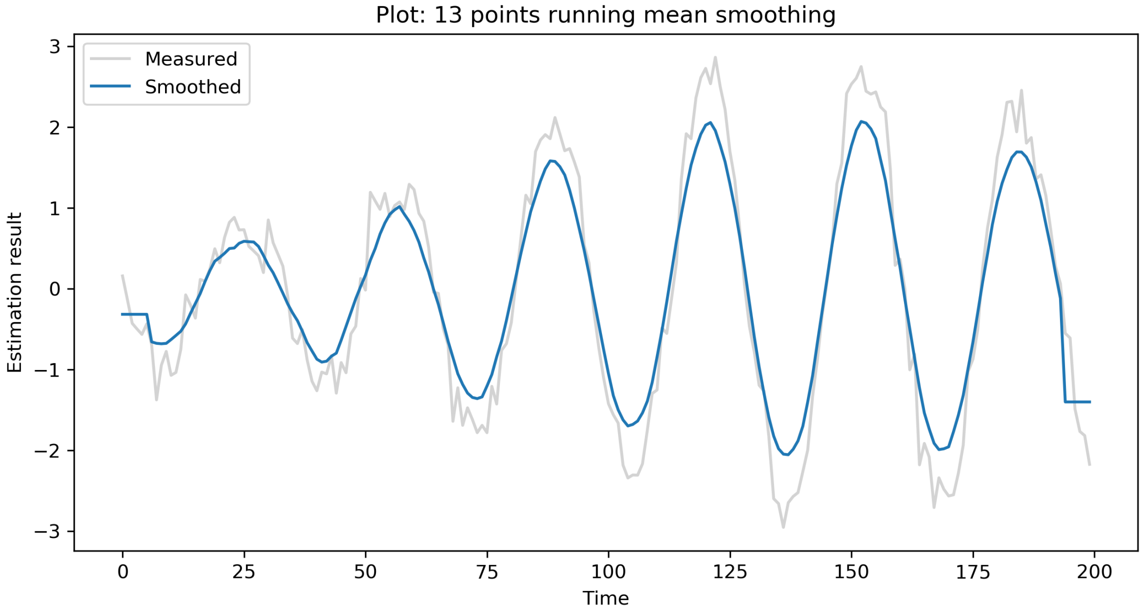
In [4]:

```
#5 Generating measurments
#Variance
s2n = 0.05
sn = np.sqrt(s2n)
Z = np.zeros(C)
n = np.random.normal(0,sn,C)

#determine measurment
for i in range (C):
    Z[i] = X[i] + n[i]

figure(num=None, figsize=(10, 5), dpi=300, facecolor='w', edgecolor='k')
plt.title('Plot: Measured sine')
plt.plot(Z)
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.show()
```



In [5]:

```python
#6 Running mean
def rm(array, m):
    length = array.size
    smooth_arr = np.empty([length])
    diff = int((m-1)/2)

    #Pre-calculation: mean of the first six entries in dataset
    mean_first = 0
    for i in range (diff):
        mean_first += array[i]
    mean_first /= diff

    #Pre-calculation: mean of the last six entries in dataset
    mean_last = 0
    for i in range (diff):
        mean_last += array[length - i - 1]
    mean_last /= diff

    #Calculation for smoothing data in array including M period points
    for i in range (0,length):
        if i < diff:
            smooth_arr[i] = mean_first
        elif i > (length - diff-1):
            smooth_arr[i] = mean_last
        else:
            sum_it = 0
            for n in range (m):
                it = n - diff
                sum_it += 1/m*array[i-it]
            smooth_arr[i] = sum_it
    return smooth_arr

sm_z = rm(Z,13)

figure(num=None, figsize=(10, 5), dpi=300, facecolor='w', edgecolor='k')
plt.title('Plot: 13 points running mean smoothing')
plt.plot(Z, label='Measured', c='lightgrey')
plt.plot(sm_z, label='Smoothed')
plt.xlabel('Time')
plt.ylabel('Estimation result')
plt.legend()
plt.show()
```

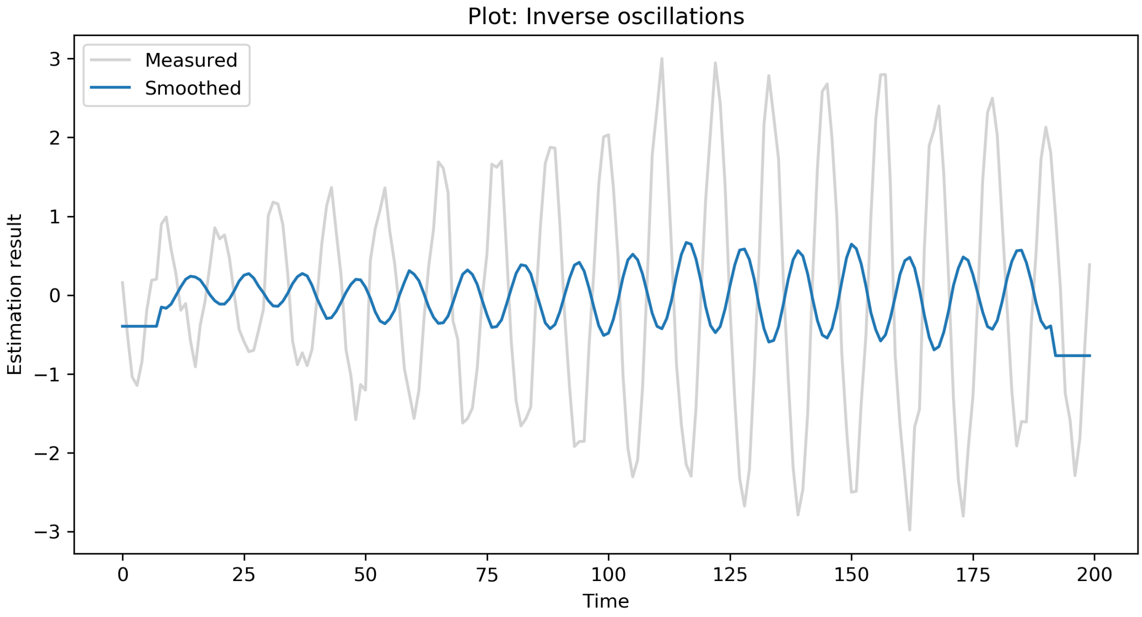


In [6]:

```
#7 Period of oscillation
M = 17
```

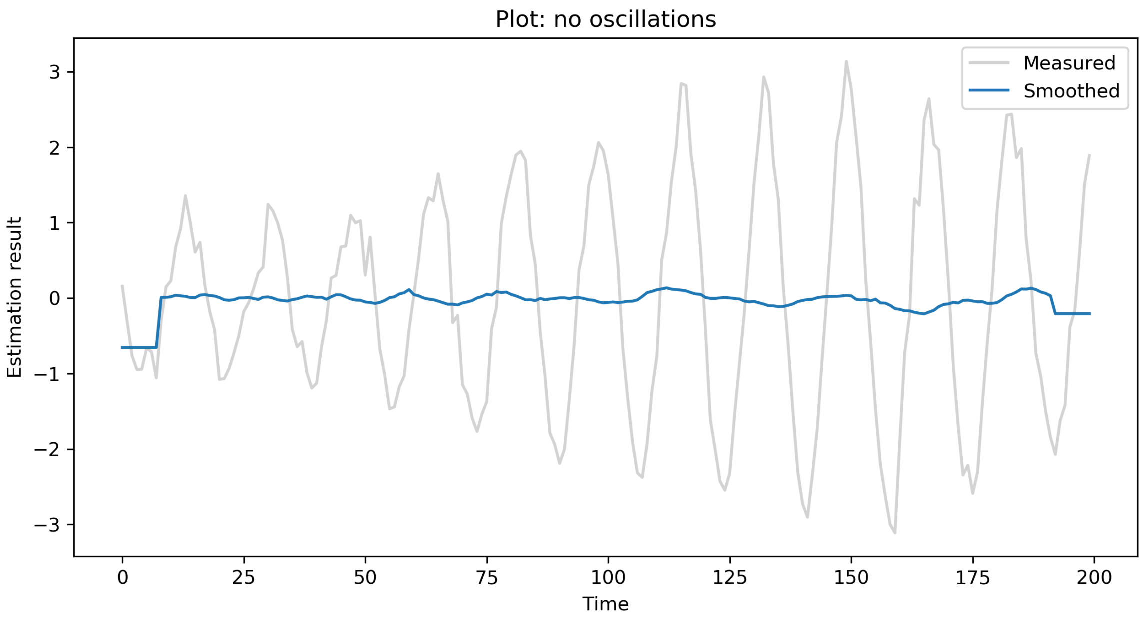
In [7]:

```
#a 1.5*M
afreq_a = 2*1.5*np.pi/M
#b M
afreq_b = 2*np.pi/M
#c < 0.5M
afreq_c = 2*0.4*np.pi/M

#a
Xa = np.zeros(C)
for i in range (C):
    Xa[i] = A[i]*np.sin(afreq_a*i+3)
#determine measurment
Za = np.zeros(C)
for i in range (C):
    Za[i] = Xa[i] + n[i]

sm_za = rm(Za,M)

figure(num=None, figsize=(10, 5), dpi=300, facecolor='w', edgecolor='k')
plt.title('Plot: Inverse oscillations')
plt.plot(Za, label='Measured', c='lightgrey')
plt.plot(sm_za, label='Smoothed')
plt.xlabel('Time')
plt.ylabel('Estimation result')
plt.legend()
plt.show()
```



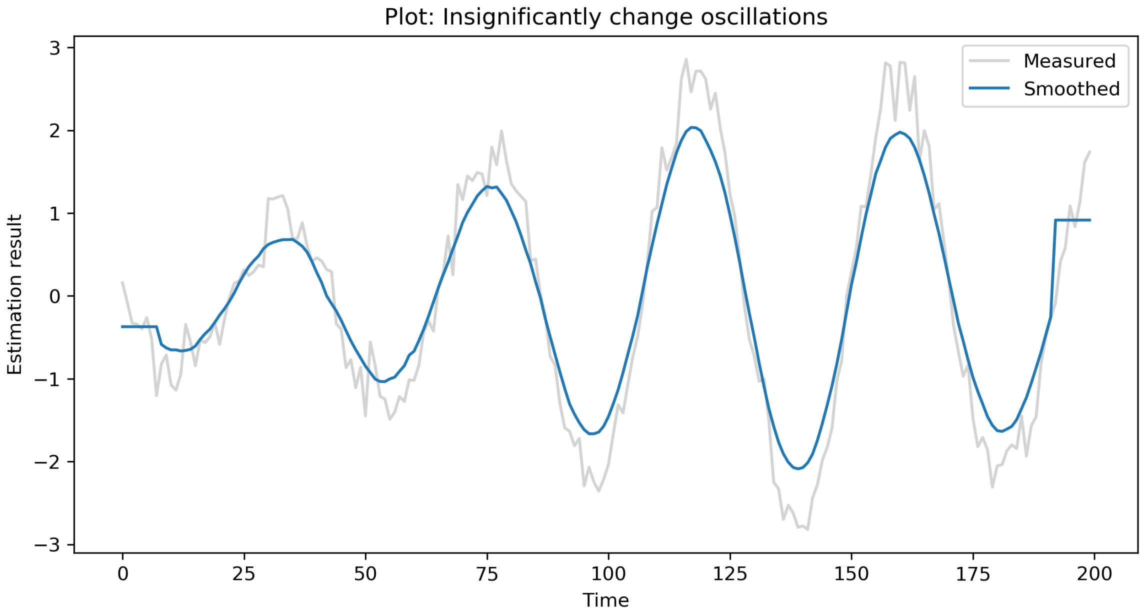
In [8]:

```
#b
Xb = np.zeros(C)
for i in range (C):
    Xb[i] = A[i]*np.sin(afreq_b*i+3)
#determine measurment
Zb = np.zeros(C)
for i in range (C):
    Zb[i] = Xb[i] + n[i]

sm_zb = rm(Zb,M)

figure(num=None, figsize=(10, 5), dpi=300, facecolor='w', edgecolor='k')
plt.title('Plot: no oscillations')
plt.plot(Zb, label='Measured', c='lightgrey')
plt.plot(sm_zb, label='Smoothed')
plt.xlabel('Time')
plt.ylabel('Estimation result')
plt.legend()
plt.show()
```



In [9]:

```
#c
Xc = np.zeros(C)
for i in range (C):
    Xc[i] = A[i]*np.sin(afreq_c*i+3)
#determine measurment
Zc = np.zeros(C)
for i in range (C):
    Zc[i] = Xc[i] + n[i]

sm_zc = rm(Zc,M)

figure(num=None, figsize=(10, 5), dpi=300, facecolor='w', edgecolor='k')
plt.title('Plot: Insignificantly change oscillations')
plt.plot(Zc, label='Measured', c='lightgrey')
plt.plot(sm_zc, label='Smoothed')
plt.xlabel('Time')
plt.ylabel('Estimation result')
plt.legend()
plt.show()
```



In [ ]: