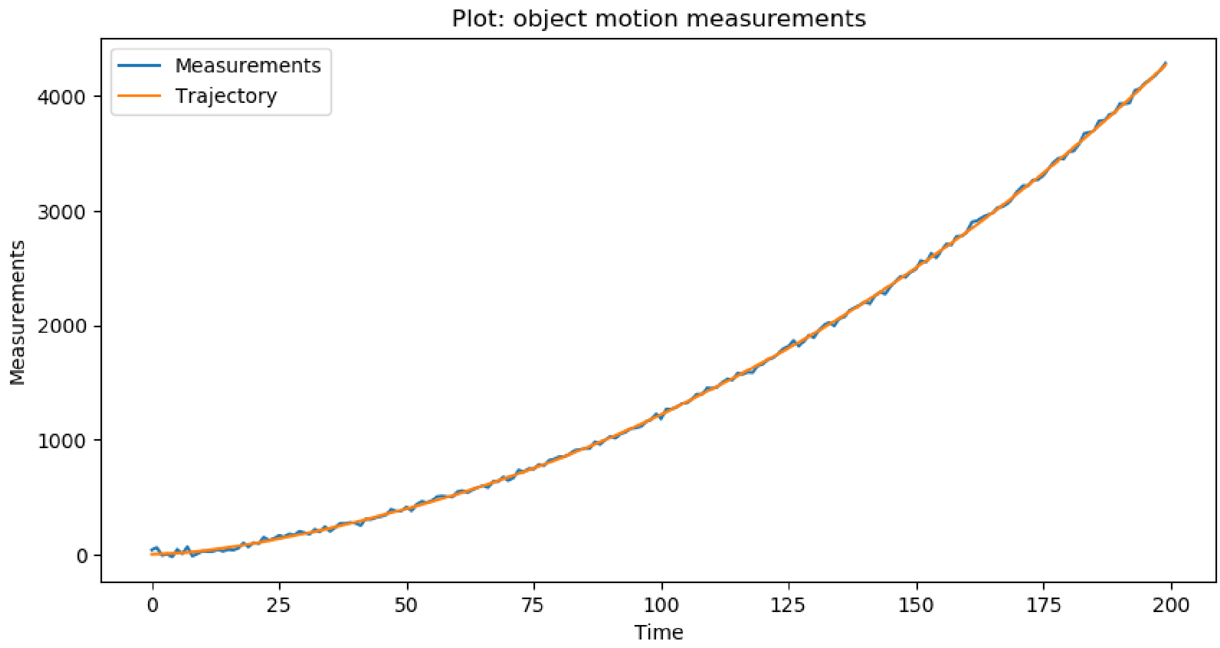
In [1]:
```
# Assignment 6
# Analysis of accuracy decrease of tracking in conditions of biased state noise.
# Team 2:
#     Ekaterina Karmanova
#     Timur Chikichev
#     Iaroslav Okunevich
#     Nikita Mikhailovskiy
#
# Skoltech, 10.10.2019
```

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
```

In [3]:
```
#Part I
#1
# Array with σa2 and ση2
sigmasTrue=[0.2,20]
a= np.random.normal(0, sigmasTrue[0], 200)
#q -bias (mathematical expectation) of random noise
q=0.2
a+=q

def Velocity (size,acc,t):
        vel=np.zeros((size))
        vel[0]=1
        for i in range(1,size):
                vel[i]=vel[i-1]+acc[i-1]*t
        return vel

def Trajectory(size,acc,vel,t):
        trajectory=np.zeros((size))
        trajectory[0]=5
        for i in range(1,size):
                trajectory[i]=trajectory[i-1]+vel[i-1]*t+acc[i]*t*t/2
        return trajectory
velocity=Velocity(200,a,1)
traject=Trajectory(200,a,velocity,1)
```

In [4]:
```
#2
def Measurements(x,ny):
        z=np.zeros(len(x))
        for i in range(0, len(x)):
            z[i]=x[i]+ny[i]
        return z

ny= np.random.normal(0, sigmasTrue[1], 200)
z=Measurements(traject,ny)
```

In [5]:
```
figure(num=None, figsize=(10, 5), dpi=100, facecolor='w', edgecolor='k')
plt.title('Plot: object motion measurements')
plt.ylabel('Measurements')
plt.xlabel('Time')
plt.plot(z,label='Measurements')
plt.plot(traject,label='Trajectory')
plt.legend()
plt.show()
```

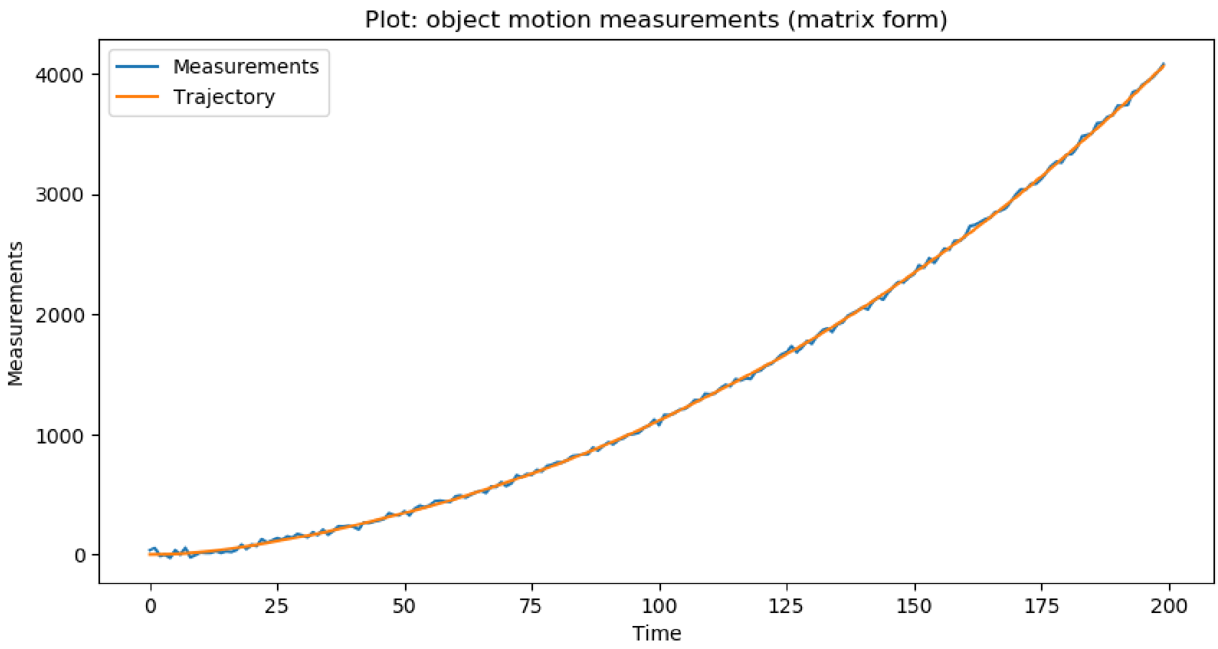


In [6]:
```
#3
def SetupFi(t):
    return np.matrix([[1,t],[0,1]])
def SetupG(t):
    return np.matrix([[(t**2)/2],[t]])
H=np.matrix([1,0])
FI=SetupFi(1)
G=SetupG(1)
```

In [7]:
```
def TrajectoryMatrix(size,acc,fi,g):
        trajectory=np.zeros([size,2])
        trajectory = trajectory[:,:]
        trajectory[0,0]=2
        trajectory[0,1]=0
        for i in range(1,size):
            matrix=np.matrix([[trajectory[i-1,0]],
                              [trajectory[i-1,1]]])
            column=fi.dot(matrix)+g*acc[i]
            trajectory[i]=column.transpose()
        trajectory=trajectory.transpose()
        return trajectory

trajectoryMatrix=TrajectoryMatrix(200,a,FI,G)
```

In [8]:
```
def MeasurementsMatrix(x,ny,h):
        z=np.zeros(len(ny))
        for i in range(0, len(ny)):
            matrix=np.matrix([[x[0,i]],
                             [x[1,i]]])
            z[i]=h.dot(matrix)+ny[i]
        return z

z=MeasurementsMatrix(trajectoryMatrix,ny,H)
```

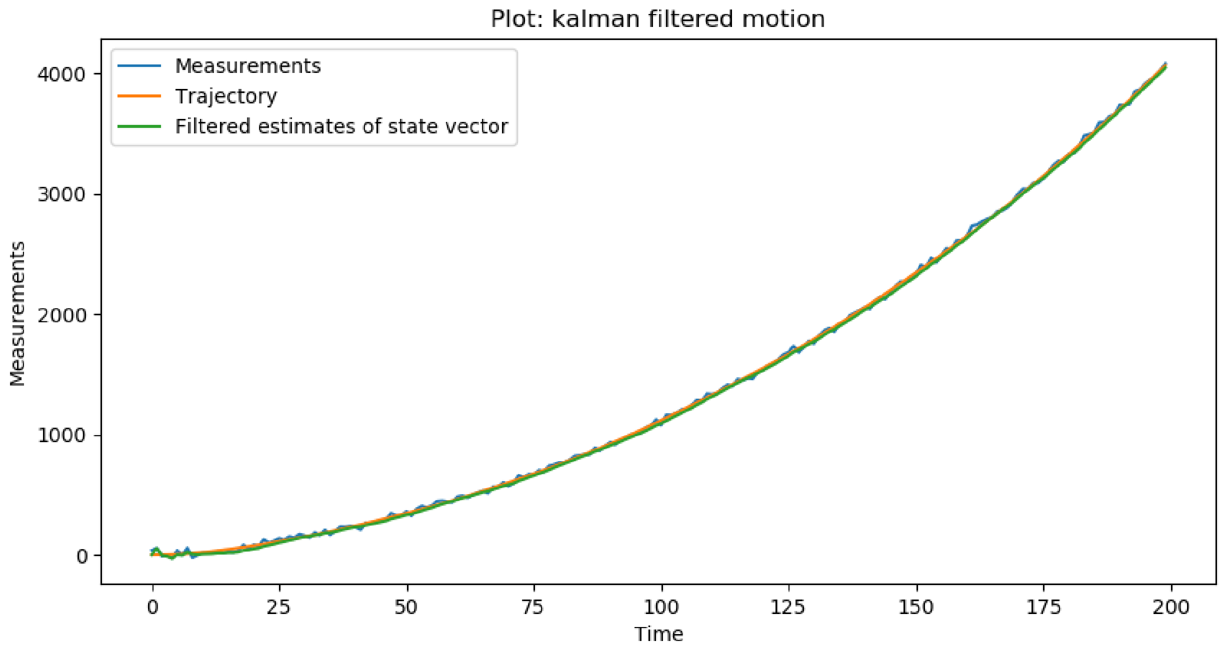
In [9]:
```
figure(num=None, figsize=(10, 5), dpi=100, facecolor='w', edgecolor='k')
plt.title('Plot: object motion measurements (matrix form)')
plt.ylabel('Measurements')
plt.xlabel('Time')
plt.plot(z,label='Measurements')
plt.plot(trajectoryMatrix[0],label='Trajectory')
plt.legend()
plt.show()
```



In [10]:
```
def SetupQ(G,sigmaA):
    return np.dot(G,G.transpose())*sigmaA
Q=SetupQ(G,sigmasTrue[0]**2)

R=sigmasTrue[1]**2

def PPrediction(p00,fi,q):
    return np.dot(np.dot(fi,p00),fi.transpose())+q

def XPrediction(fi,x10):
    return np.dot(fi,x10)

def FindKI(pIIMinus1,h,r):
    onePart=np.dot(pIIMinus1,h.transpose())
    secondpart=np.dot(np.dot(h,pIIMinus1),h.transpose())+r
    return onePart*secondpart.I

def FindPII(pIIMinus1,k,h):
    return np.dot(np.eye(2)-np.dot(k,h),pIIMinus1)

def FindXI(xI,k,zI,h):
    return xI+k*(zI-h*xI)
```

In [11]:
```
def Kalman(z,h,fi,q,r,pFiltration=np.matrix([[10000, 0],
                                             [0, 10000]])):
    arrayK=np.zeros(len(z))
    pFilterArray=np.zeros(len(z))
    tracjectKalman=np.zeros((2,len(z)))
    tracjectKalman[0,0]=2
    tracjectKalman[1,0]=0
    x0=np.matrix([[2],
                  [0]])
    pFilterArray[0]=np.sqrt(pFiltration[0,0])
    for i in range(1,len(z)):
        pPrediction=PPrediction(pFiltration, fi, q)
        xPred=XPrediction(fi,x0)
        k=FindKI(pPrediction,h,r)
        x0=FindXI(xPred,k,z[i],h)
        pFiltration=FindPII(pPrediction,k,h)

        pFilterArray[i]=np.sqrt(pFiltration[0,0])
        tracjectKalman[0,i]=x0[0]
        tracjectKalman[1,i]=x0[1]
        arrayK[i]=k[0]
    arrayK=np.delete(arrayK, 0)
    pFilterArray=np.delete(pFilterArray, 0)
    return tracjectKalman, arrayK, pFilterArray
tracjectKalman, arrayK, pFilterArray=Kalman(z,H,FI,Q,R)
```

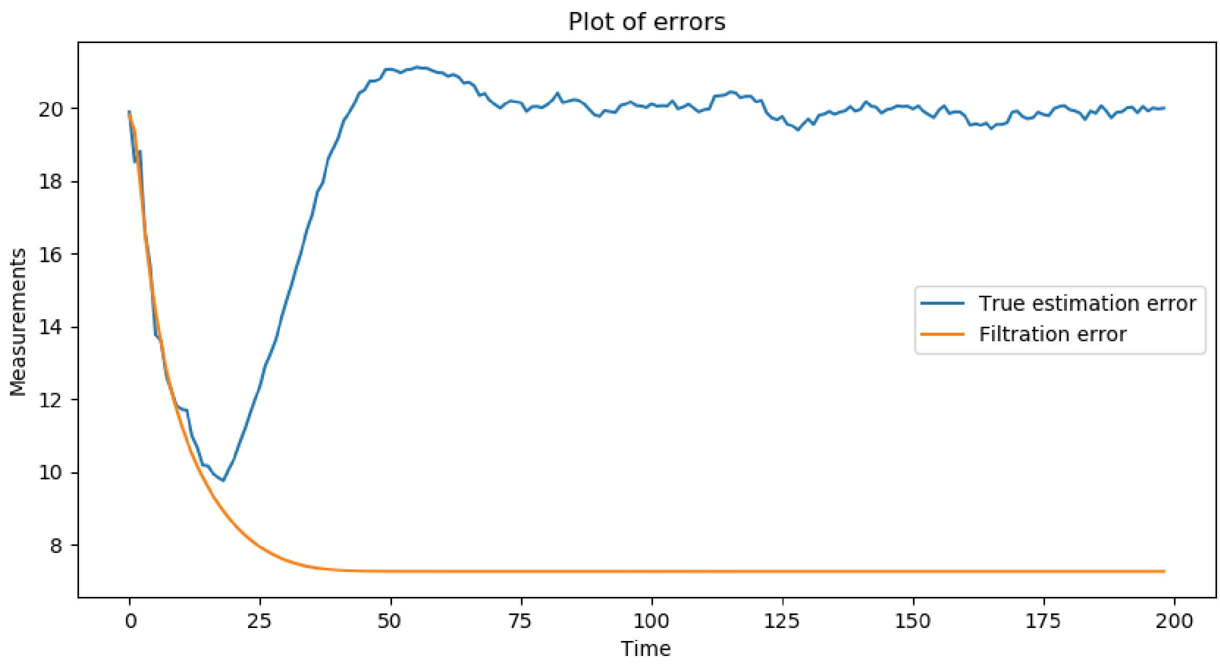
In [12]:
```
#4
figure(num=None, figsize=(10, 5), dpi=100, facecolor='w', edgecolor='k')
plt.title('Plot: kalman filtered motion')
plt.ylabel('Measurements')
plt.xlabel('Time')
plt.plot(z,label='Measurements')
plt.plot(trajectoryMatrix[0],label='Trajectory')
plt.plot(tracjectKalman[0],label='Filtered estimates of state vector')
plt.legend()
plt.show()
```



In [13]:
```
#5
def FinalError(H,FI,Q,R,G,runs,exp=False,pFiltration=np.matrix([[10000, 0],[0, 1
0000]])):
    error=0
    finalError=np.zeros(200)
    for i in range(runs):

        a= np.random.normal(0, 0.2, 200)
        a+=0.2
        trajectoryMatrix=TrajectoryMatrix(200,a,FI,G)
        ny= np.random.normal(0, 20, 200)
        z=MeasurementsMatrix(trajectoryMatrix,ny,H)
        tracjectKalman, garbage, garbage1=Kalman(z,H,FI,Q,R,pFiltration)

        for j in range(int(trajectoryMatrix.size/len(trajectoryMatrix))):
            finalError[j]+=(trajectoryMatrix[0,j]-tracjectKalman[0,j])**2
    for i in range(int(trajectoryMatrix.size/len(trajectoryMatrix))):
        finalError[i]=np.sqrt(finalError[i]/(runs-1))
    finalError=np.delete(finalError, 0)
    return finalError
```

In [14]:
```
finalError=FinalError(H,FI,Q,R,G,500)
```

In [15]:
```
#6
figure(num=None, figsize=(10, 5), dpi=100, facecolor='w', edgecolor='k')
plt.title('Plot of errors')
plt.ylabel('Measurements')
plt.xlabel('Time')
plt.plot(finalError,label='True estimation error')
plt.plot(pFilterArray,label='Filtration error')
plt.legend()
plt.show()
```

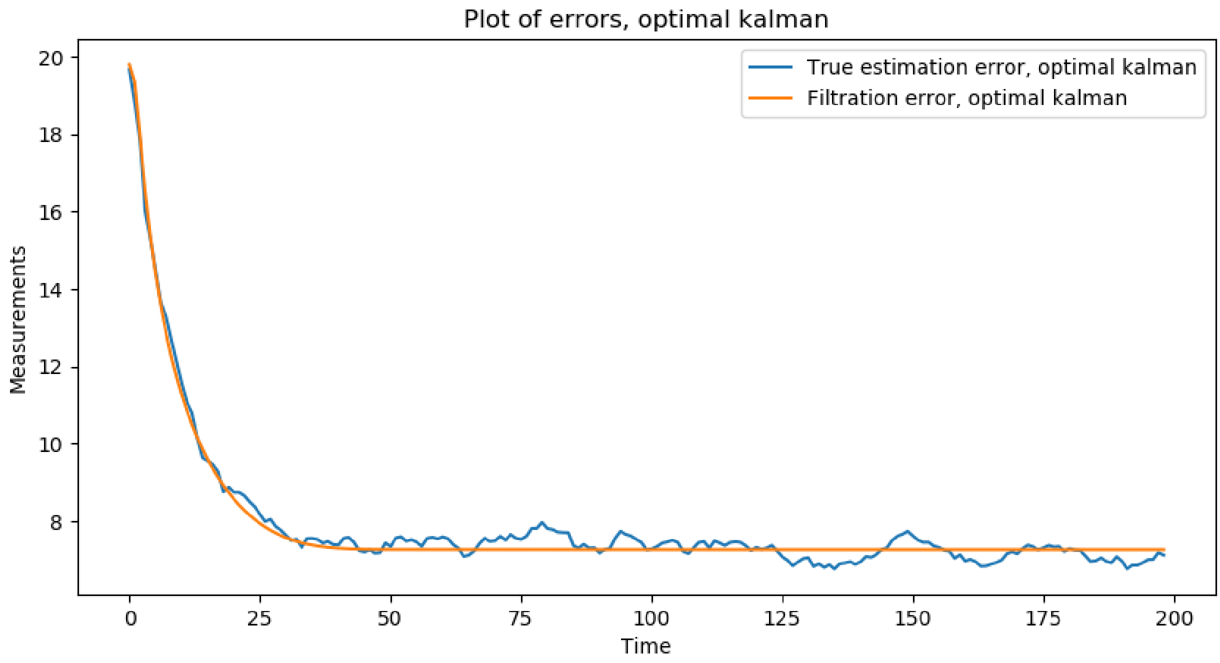


In [16]:
```
#7
def XPredictionOptimal(fi,x10,g):
    return np.dot(fi,x10)+g*0.2
```

In [17]:
```
def KalmanOptimal(z,h,fi,q,r,g,pFiltration=np.matrix([[10000, 0],
                                                       [0, 10000]])):
    arrayK=np.zeros(len(z))
    pFilterArray=np.zeros(len(z))
    tracjectKalman=np.zeros((2,len(z)))
    tracjectKalman[0,0]=2
    tracjectKalman[1,0]=0
    x0=np.matrix([[2],
                  [0]])
    pFilterArray[0]=np.sqrt(pFiltration[0,0])
    for i in range(1,len(z)):
        pPrediction=PPrediction(pFiltration, fi, q)
        xPred=XPredictionOptimal(fi,x0,g)
        k=FindKI(pPrediction,h,r)
        x0=FindXI(xPred,k,z[i],h)
        pFiltration=FindPII(pPrediction,k,h)

        pFilterArray[i]=np.sqrt(pFiltration[0,0])
        tracjectKalman[0,i]=x0[0]
        tracjectKalman[1,i]=x0[1]
        arrayK[i]=k[0]
    arrayK=np.delete(arrayK, 0)
    pFilterArray=np.delete(pFilterArray, 0)
    return tracjectKalman, arrayK, pFilterArray

def FinalErrorOptimal(H,FI,Q,R,G,runs,exp=False,pFiltration=np.matrix([[10000, 0
],[0, 10000]])):
    error=0
    finalError=np.zeros(200)
    for i in range(runs):

        a= np.random.normal(0, 0.2, 200)
        a+=0.2
        trajectoryMatrix=TrajectoryMatrix(200,a,FI,G)
        ny= np.random.normal(0, 20, 200)
        z=MeasurementsMatrix(trajectoryMatrix,ny,H)
        tracjectKalman, garbage, garbage1=KalmanOptimal(z,H,FI,Q,R,G,pFiltration
)

        for j in range(int(trajectoryMatrix.size/len(trajectoryMatrix))):
            finalError[j]+=(trajectoryMatrix[0,j]-tracjectKalman[0,j])**2
    for i in range(int(trajectoryMatrix.size/len(trajectoryMatrix))):
        finalError[i]=np.sqrt(finalError[i]/(runs-1))
    finalError=np.delete(finalError, 0)
    return finalError
```

In [18]:
```
#8
finalErrorOptimal=FinalErrorOptimal(H,FI,Q,R,G,500)
tracjectKalmanOptimal, arrayKOptimal, pFilterArrayOptimal=KalmanOptimal(z,H,FI,Q
,R,G)
```

In [19]:
```
figure(num=None, figsize=(10, 5), dpi=100, facecolor='w', edgecolor='k')
plt.title('Plot of errors, optimal kalman')
plt.ylabel('Measurements')
plt.xlabel('Time')
plt.plot(finalErrorOptimal,label='True estimation error, optimal kalman')
plt.plot(pFilterArrayOptimal,label='Filtration error, optimal kalman')
plt.legend()
plt.show()
```



In [ ]: