Setting

In [1]:
```
%pwd
```

Out[1]:
```
'C:¥¥Users¥¥david¥¥Downloads¥¥CODE'
```

In [2]:
```
%cd ../
```

```
C:¥Users¥david¥Downloads
```

外為ユーロ円のデータとnumpy

- ファイルからデータをロードする
- VWAP (Volume Weighted Average Price) ボリューム加重平均価格
- 統計集計
- 移動平均線 (sma: simple moving average)
- Bollinger Bands ボリンジャーバンド
- Trend lines トレンド線

Hourly and daily data (60分と１日ごとのデータ)

In [8]:
```
more EURJPY60_200612.csv
```

In [3]:
```
#VWAP
from __future__ import print_function
import numpy as np

c,v=np.loadtxt('EURJPY60_200612.csv', delimiter='¥t', usecols=(4,5), unpack=True)
vwap = np.average(c, weights=v)
print("VWAP =", vwap) #

print("mean =", np.mean(c))

t = np.arange(len(c))
print("twap =", np.average(c, weights=t))
```

```
VWAP = 125.27609776755843
mean = 126.53013832
twap = 126.35331029484588
```

In [4]:
```
type(c)
```

Out[4]:
```
numpy.ndarray
```

In [8]:
```
c[-10:]
```

Out[8]:
```
array([121.392, 121.381, 121.202, 120.924, 120.697, 120.784, 120.728,
       120.646, 120.657, 120.563])
```

In [9]:
```
v[-10:]
```

Out[9]:
```
array([26142., 25616., 18205., 20897., 18636., 17707.,  9850.,  3911.,
        7506.,  7421.])
```

In [11]:
```
import csv
f = open("EURJPY60_200612.csv",'rt')
myReader = csv.reader(f)
```

In [12]:
```
i=0
for row in myReader:
    print(row)
    data = row[0].split('¥t')
    print (data)
    i+=1
    if i>2: break
```

```
['2012-05-31 00:00¥t97.716¥t97.740¥t97.460¥t97.502¥t12273']
['2012-05-31 00:00', '97.716', '97.740', '97.460', '97.502', '12273']
['2012-05-31 01:00¥t97.503¥t97.571¥t97.394¥t97.513¥t9539']
['2012-05-31 01:00', '97.503', '97.571', '97.394', '97.513', '9539']
['2012-05-31 02:00¥t97.508¥t97.508¥t97.355¥t97.396¥t7039']
['2012-05-31 02:00', '97.508', '97.508', '97.355', '97.396', '7039']
```

In [13]:
```
h,l=np.loadtxt('EURJPY60_200612.csv', delimiter='¥t', usecols=(2,3), unpack=True)
print("highest =", np.max(h))
print("lowest =", np.min(l))
print((np.max(h) + np.min(l)) /2)

print("Spread high price", np.ptp(h))
print("Spread low price", np.ptp(l))
```

```
highest = 149.788
lowest = 94.11
121.94900000000001
Spread high price 55.540000000000006
Spread low price 55.33800000000001
```

In [14]:
```
c = np.loadtxt('EURJPY60_200612.csv', delimiter='¥t', usecols=(4,), unpack=True)
print("median =", np.median(c))
sorted = np.msort(c)
print("sorted =", sorted)

N = len(c)
print("middle =", sorted[int((N - 1)/2)])
print("average middle =", (sorted[int(N /2)] + sorted[int((N - 1) / 2)]) / 2)

print("variance =", np.var(c))
print("variance from definition =", np.mean((c - c.mean())**2))
```

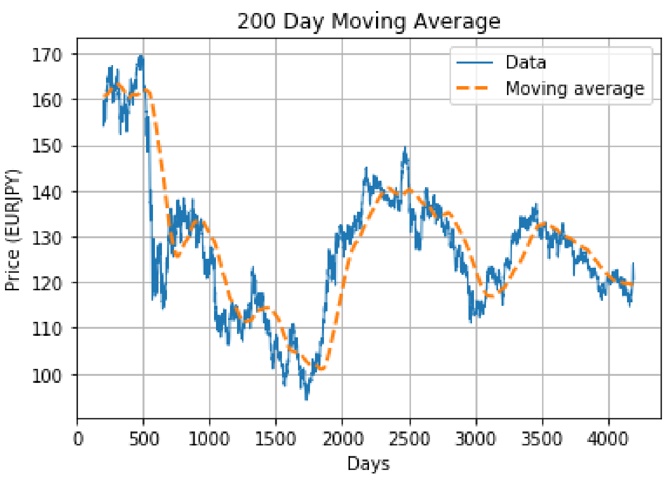
```
median = 128.284
sorted = [ 94.209  94.219  94.234 ... 149.432 149.505 149.675]
middle = 128.284
average middle = 128.284
variance = 108.75988114094758
variance from definition = 108.75988114094758
```

In [25]:
```
c = np.loadtxt('EURJPY1440_200612.csv', delimiter='¥t', usecols=(5,), unpack=True)
returns = np.diff( c ) / c[ : -1]
print("Standard deviation =", np.std(returns))

logreturns = np.diff( np.log(c) )

posretindices = np.where(returns > 0)
print("Indices with positive returns", posretindices)

annual_volatility = np.std(logreturns)/np.mean(logreturns)
annual_volatility = annual_volatility / np.sqrt(1./252.)
print("Annual volatility", annual_volatility)

print("Monthly volatility", annual_volatility * np.sqrt(1./12.))
```

```
Standard deviation = 949.3220345860681
Indices with positive returns (array([   5,    7,   11, ..., 4195, 4196, 4197], dtype=int64),)
Annual volatility 307941.91258452716
Monthly volatility 88895.17306272248
```

In [26]:
```
import matplotlib.pyplot as plt

N = 200

weights = np.ones(N) / N
print("Weights", weights)

c = np.loadtxt('EURJPY1440_200612.csv', delimiter='¥t', usecols=(4,), unpack=True)
sma = np.convolve(weights, c)[N-1:-N+1]
t = np.arange(N - 1, len(c))
plt.plot(t, c[N-1:], lw=1.0, label="Data")
plt.plot(t, sma, '--', lw=2.0, label="Moving average")
plt.title("200 Day Moving Average")
plt.xlabel("Days")
plt.ylabel("Price (EURJPY)")
plt.grid()
plt.legend()
plt.show()
```

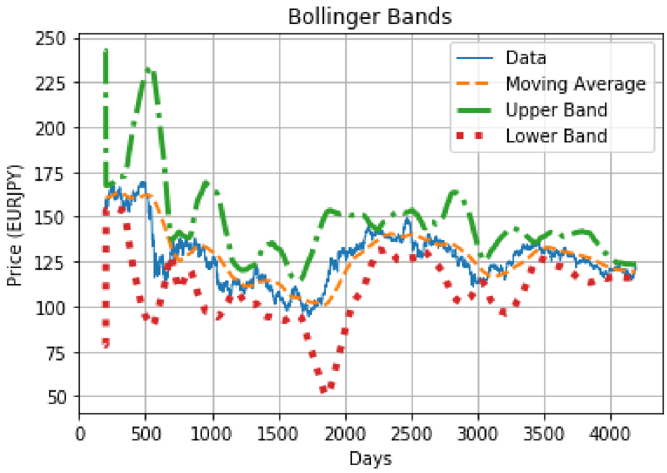
```
Weights [0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005
 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005
 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005
 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005
 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005
 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005
 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005
 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005
 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005
 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005
 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005
 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005
 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005
 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005
 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005
 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005
 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005]
```



In [30]:

```
N = 200

weights = np.ones(N) / N
print("Weights", weights)

c = np.loadtxt('EURJPY1440_200612.csv', delimiter='¥t', usecols=(4,), unpack=True)
sma = np.convolve(weights, c)[N-1:-N+1]
deviation = []
C = len(c)

for i in range(N - 1, C):
   if i + N < C:
      dev = c[i: i + N]
   else:
      dev = c[-N:]

   averages = np.zeros(N)
   averages.fill(sma[i - N - 1])
   dev = dev - averages
   dev = dev ** 2
   dev = np.sqrt(np.mean(dev))
   deviation.append(dev)

deviation = 2 * np.array(deviation)
print(len(deviation), len(sma))
upperBB = sma + deviation
lowerBB = sma - deviation

c_slice = c[N-1:]
between_bands = np.where((c_slice < upperBB) & (c_slice > lowerBB))

#print(lowerBB[between_bands])
#print(c[between_bands])
#print(upperBB[between_bands])
between_bands = len(np.ravel(between_bands))
print("Ratio between bands", float(between_bands)/len(c_slice))

t = np.arange(N - 1, C)
plt.plot(t, c_slice, lw=1.0, label='Data')
plt.plot(t, sma, '--', lw=2.0, label='Moving Average')
plt.plot(t, upperBB, '-.', lw=3.0, label='Upper Band')
plt.plot(t, lowerBB, ':', lw=4.0, label='Lower Band')
plt.title('Bollinger Bands')
plt.xlabel('Days')
plt.ylabel('Price (EURJPY)')
plt.grid()
plt.legend()
plt.show()
```

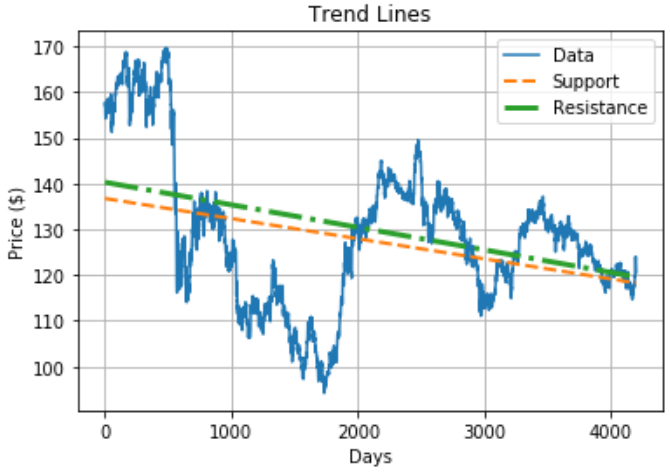
```
Weights [0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005
 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005
 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005
 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005
 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005
 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005
 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005
 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005
 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005
 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005
 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005
 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005
 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005
 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005
 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005
 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005
 0.005 0.005 0.005 0.005 0.005 0.005 0.005 0.005]
4000 4000
Ratio between bands 0.9795
```



In [29]:

```
def fit_line(t, y):
    ''' Fits t to a line y = at + b '''
    A = np.vstack([t, np.ones_like(t)]).T

    return np.linalg.lstsq(A, y)[0]

# Determine pivots
h, l, c = np.loadtxt('EURJPY1440_200612.csv', delimiter='¥t', usecols=(2,3,4,), unpack=True)
pivots = (h + l + c) / 3
print("Pivots", pivots)

# Fit trend lines
t = np.arange(len(c))
sa, sb = fit_line(t, pivots - (h - l))
ra, rb = fit_line(t, pivots + (h - l))

support = sa * t + sb
resistance = ra * t + rb
condition = (c > support) & (c < resistance)
print("Condition", condition)
between_bands = np.where(condition)
#print(support[between_bands])
#print(c[between_bands])
#print(resistance[between_bands])
between_bands = len(np.ravel(between_bands))
print("Number points between bands", between_bands)
print("Ratio between bands", float(between_bands)/len(c))

print("Tomorrows support", sa * (t[-1] + 1) + sb)
print("Tomorrows resistance", ra * (t[-1] + 1) + rb)

a1 = c[c > support]
a2 = c[c < resistance]
print("Number of points between bands 2nd approach" ,len(np.intersect1d(a1, a2)))

# Plotting
plt.plot(t, c, label='Data')
plt.plot(t, support, '--', lw=2.0, label='Support')
plt.plot(t, resistance, '-.', lw=3.0, label='Resistance')
plt.title('Trend Lines')
plt.xlabel('Days')
plt.ylabel('Price (EURJPY)')
plt.grid()
plt.legend()
plt.show()
```

```
Pivots [157.02433333 157.492      157.412      ... 122.01866667 121.85166667
 120.96166667]
Condition [False False False ... False False False]
Number points between bands 327
Ratio between bands 0.0778756846868302
Tomorrows support 118.25809876989643
Tomorrows resistance 119.58452665915654
Number of points between bands 2nd approach 354

D:¥Anaconda3¥lib¥site-packages¥ipykernel_launcher.py:5: FutureWarning: `rcond` parameter will cha
nge to the default of machine precision times ``max(M, N)`` where M and N are the input matrix di
mensions.
To use the future default and silence this warning we advise to pass `rcond=None`, to keep using
the old, explicitly pass `rcond=-1`.
  """
```



Another time frame 4 Hours data

In [18]:
```
c,v=np.loadtxt('EURJPY240.csv', delimiter='¥t', usecols=(4,5), unpack=True)
vwap = np.average(c, weights=v)
print("VWAP =", vwap) #

print("mean =", np.mean(c))

t = np.arange(len(c))
print("twap =", np.average(c, weights=t))
```

```
VWAP = 138.22147426016633
mean = 128.70700069376994
twap = 125.44415018345862
```

In [19]:
```
h,l=np.loadtxt('EURJPY240.csv', delimiter='¥t', usecols=(2,3), unpack=True)
print("highest =", np.max(h))
print("lowest =", np.min(l))
print((np.max(h) + np.min(l)) /2)

print("Spread high price", np.ptp(h))
print("Spread low price", np.ptp(l))
```

```
highest = 169.96
lowest = 94.11
132.035
Spread high price 75.563
Spread low price 75.355
```

In [20]:
```
c = np.loadtxt('EURJPY240.csv', delimiter='¥t', usecols=(4,), unpack=True)
print("median =", np.median(c))
sorted = np.msort(c)
print("sorted =", sorted)

N = len(c)
print("middle =", sorted[int((N - 1)/2)])
print("average middle =", (sorted[int(N /2)] + sorted[int((N - 1) / 2)]) / 2)

print("variance =", np.var(c))
print("variance from definition =", np.mean((c - c.mean())**2))
```

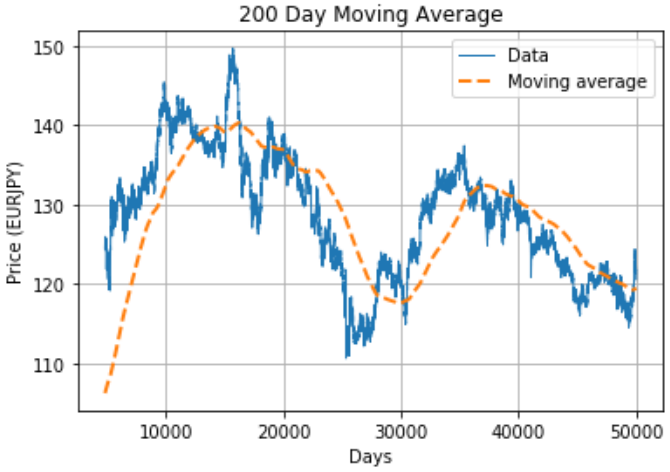
```
median = 128.084
sorted = [ 94.219  94.307  94.328 ... 169.65  169.745 169.775]
middle = 128.084
average middle = 128.084
variance = 270.9633866302663
variance from definition = 270.9633866302663
```

In [21]:
```
c = np.loadtxt('EURJPY240.csv', delimiter='¥t', usecols=(4,), unpack=True)
returns = np.diff( c ) / c[ : -1]
print("Standard deviation =", np.std(returns))

logreturns = np.diff( np.log(c) )

posretindices = np.where(returns > 0)
print("Indices with positive returns", posretindices)

annual_volatility = np.std(logreturns)/np.mean(logreturns)
annual_volatility = annual_volatility / np.sqrt(1./252.)
print("Annual volatility", annual_volatility)

print("Monthly volatility", annual_volatility * np.sqrt(1./12.))
```

```
Standard deviation = 0.0031729844957206314
Indices with positive returns (array([    2,     3,     4, ..., 21612, 21615, 21616], dtype=int6
4),)
Annual volatility -4121.133814650613
Monthly volatility -1189.6688586275004
```

In [31]:
```
N = 200 * 24

weights = np.ones(N) / N
print("Weights", weights)

c = np.loadtxt('EURJPY60_200612.csv', delimiter='¥t', usecols=(4,), unpack=True)
sma = np.convolve(weights, c)[N-1:-N+1]
t = np.arange(N - 1, len(c))
plt.plot(t, c[N-1:], lw=1.0, label="Data")
plt.plot(t, sma, '--', lw=2.0, label="Moving average")
plt.title("200 Day Moving Average")
plt.xlabel("Days")
plt.ylabel("Price (EURJPY)")
plt.grid()
plt.legend()
plt.show()
```

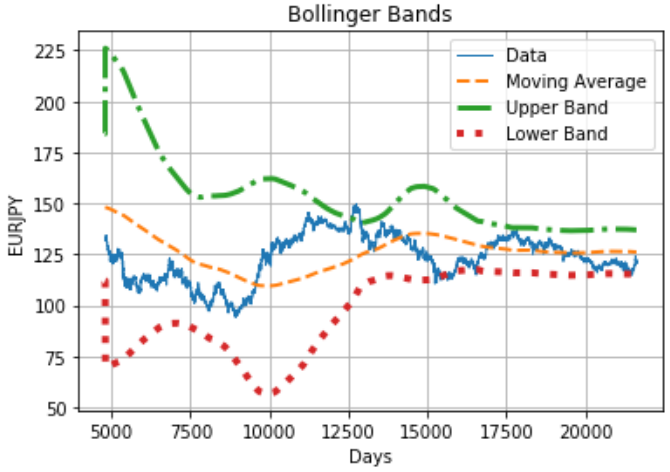
```
Weights [0.00020833 0.00020833 0.00020833 ... 0.00020833 0.00020833 0.00020833]
```



In [32]:

```
N = 200 * 24

weights = np.ones(N) / N
print("Weights", weights)

c = np.loadtxt('EURJPY240.csv', delimiter='¥t', usecols=(4,), unpack=True)
sma = np.convolve(weights, c)[N-1:-N+1]
deviation = []
C = len(c)

for i in range(N - 1, C):
   if i + N < C:
      dev = c[i: i + N]
   else:
      dev = c[-N:]

   averages = np.zeros(N)
   averages.fill(sma[i - N - 1])
   dev = dev - averages
   dev = dev ** 2
   dev = np.sqrt(np.mean(dev))
   deviation.append(dev)

deviation = 2 * np.array(deviation)
print(len(deviation), len(sma))
upperBB = sma + deviation
lowerBB = sma - deviation

c_slice = c[N-1:]
between_bands = np.where((c_slice < upperBB) & (c_slice > lowerBB))

print(lowerBB[between_bands])
print(c[between_bands])
print(upperBB[between_bands])
between_bands = len(np.ravel(between_bands))
print("Ratio between bands", float(between_bands)/len(c_slice))

t = np.arange(N - 1, C)
plt.plot(t, c_slice, lw=1.0, label='Data')
plt.plot(t, sma, '--', lw=2.0, label='Moving Average')
plt.plot(t, upperBB, '-.', lw=3.0, label='Upper Band')
plt.plot(t, lowerBB, ':', lw=4.0, label='Lower Band')
plt.title('Bollinger Bands')
plt.xlabel('Days')
plt.ylabel('EURJPY')
plt.grid()
plt.legend()
plt.show()
```

```
Weights [0.00020833 0.00020833 0.00020833 ... 0.00020833 0.00020833 0.00020833]
16822 16822
[112.04705887 112.04115249  70.10050067 ... 115.17077688 115.17004479
 115.16931557]
[157.067 157.039 156.79  ... 124.3   124.065 124.143]
[184.13218447 184.12809418 226.05894641 ... 136.95201854 136.95128563
 136.95055568]
Ratio between bands 0.9525621210319819
```



In [ ]: