Explanation and validation of solar radiation calculation

based on statistical data from measurements and methods described in *solar engineering for thermal processes ISBN: 978-1-118-67160-3*

The functions written below closely resemble the script which wraps these functions.

The only external modules used are: numpy, matplotlib and if you want to use jupyter notebooks, a lot more.

In [34]:
```
# Required imports
import datetime
import csv
import Get_Radiation as R
import matplotlib.pyplot as plt #pip install matplotlib
import numpy as np # pip install numpy
%matplotlib notebook
```

- Solar engineering for thermal processes describes multiple steps to approximate possible solar radiation for a given location and time. The base for these calculations are measurements taken from over the world, this lead to an empirical model.
- However, measurements are available from all over the world.
    - Instead of using the model to determine all values, the hourly cloudiness index can be calculated with measurements and accurate clear-sky solar radiation calculations.
    - The Diffuse and direct component are estimated with a correlation on **kt = Qglob/Qext**
        - where Qglob is global radiation and Qext is extraterestial radiation (as seen by the atmosphere) (p.71)
    - The fraction **Id/I** is a function of kt
        - where Id is diffuse radiation on horizontal plane, I is global radiation on horizontal plane.

To be able to accurately approximate real solar radiation (clear and cloudy sky), several steps are taken.

1. first a dataset has to be known for the given location, preferably multiple years in hour data. Global radiation data is fine. From this, average hourly **kt** values are determined for the given dataset. This takes disturbances in the atmosphere into account.
2. The PySolar toolbox is used to calculate global radiation (=! extraterestial radiation). This is used to calculate the value **Qglob,real / Qglob,pysolar** for every datapoint.
    A. from these points, statistical data is saved to be able to generate a similar set with similar statistical properties.
    B. this is where a bigger set becomes better, a year can have 8760 hours. if you have 40 years of data, the standard deviation of 40 datapoints per hour is calculated.
3. The combination of 1 and 2 together with **Id/I** can be used to recreate data for a given location.

In [35]:
```
def CalculateLocalVariation(Filename,lat,lon,verbal=False): # takes KNMI filef
ormat. Q, (j/mm2), HH(hour of day)are needed at least.
    dates = []
    hourdatakt = [0.0 for i in range(366*24)]
    hourdata = [[] for i in range(366*24)]
    with open(Filename,'r') as csvfile:
          reader = csv.DictReader(csvfile)
          for row in reader:
              try:
                  h = int(row["HH"])
                  h = h-1
                  if h < 10:
                      h = "0%s"%h
                  date = "%s-%s"%(row['YYYYMMDD'],h)
                  date = datetime.datetime.strptime(date,"%Y%m%d-%H")
                  DOY = int(R.solar.GetDayOfYear(date+datetime.timedelta(minut
es=30))) #midpoint of hour is needed.
                  HOY = int(R.GetHourOfYear(date))

                  I = float((int(row["Q"])*100*100))/3600
                  alt,azi,Qsim,Qdir,Qdif = R.GetRad(lat,lon,date)
                  Qext = R.radiation.GetApparentExtraterrestrialFlux(DOY)
                  Ferror = False
                  if(I != 0 and Qsim != 0):
                    Ferror = True


                  if Ferror:
                      if hourdatakt[HOY] == 0: # init case
                          hourdatakt[HOY] = (I/Qext)
                      else:
                          hourdatakt[HOY] = ((I/Qext) + hourdatakt[HOY])/2
                      hourdata[HOY].append(I/Qsim)
                  if verbal and Ferror:
                      print "I: {} Sim: {} [I/Qsim]: {} date: {}, HOY:{}".form
at(I,Qsim,(I/Qext),date,HOY)
              except ValueError:
                  if verbal:
                      print "EOF"
    if verbal:
        print hourdata
    for i in hourdata:
        plt.plot(i)
    plt.ylabel('Hourly kt')
    plt.title('kt(hour) calculated according to Solar Engineering of thermal p
rocesses p.71')
    plt.grid(True)
    plt.savefig("hourly.png")
    plt.show()
    return hourdata,hourdatakt
```

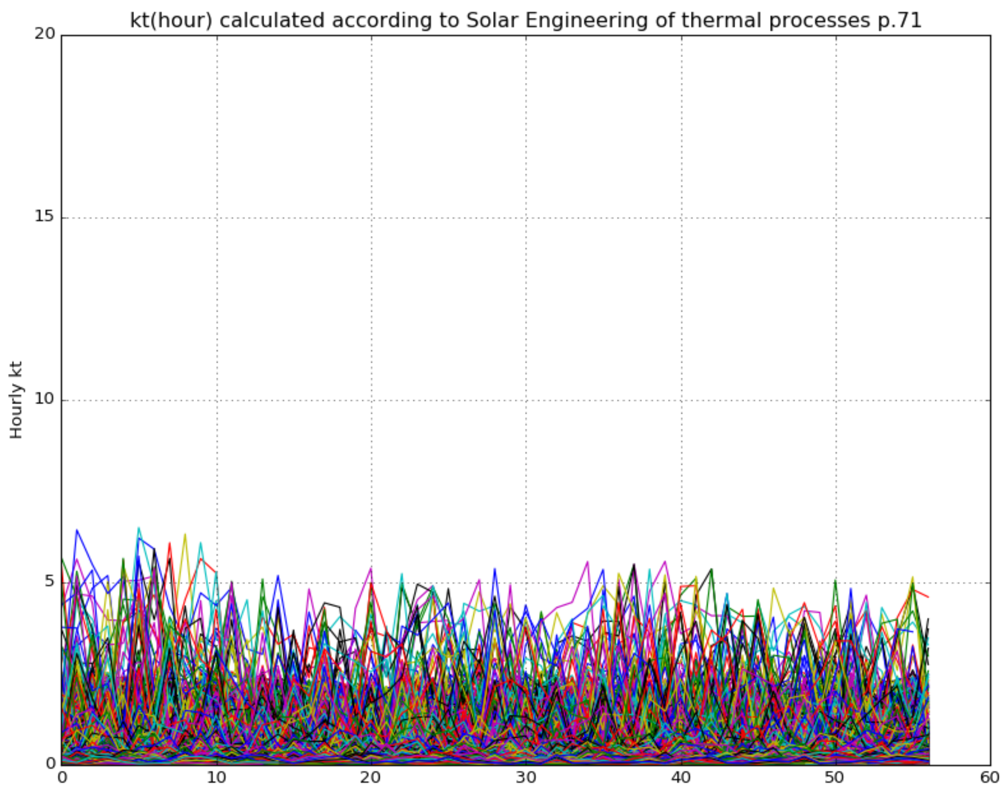
The above function does 2 things and is designed to take KNMI data;

1. calculate the mean kt value (only mean is needed here)
2. create an array for each hour with **Qglob/Qglob,pysolar**
   A. we need statistical data from these arrays; at least standard deviation and mean value

---

syntax: **HourData,HourDataKt = CalculateLocalKT("5-180_52-100.csv",5.18,52.1)** this would use the prepared file, which contains 1960 to 2016 in hour steps for 'De Bilt', the Netherlands (latitude 5.18, longitude 52.1). An extra argument can be passed, if set to True it wil return all data it calculates. *Warning: this can slow down the process.*

In [36]:
```
HourData,HourDataKt = R.CalculateLocalVariation("5-180_52-100.csv",5.18,52.1)
#HourData = CalculateLocalKT("KNMI_20171107_hourly.txt",5.18,52.1)
R.ToCSV("hourkt_1960-2016.csv", HourDataKt, fieldnames = ["kt"])
```



```
Done writing file with filename:
 hourkt_1960-2016.csv
```

If you are calculating new values, save *kt* with:

R.ToCSV("hourkt_1960-2016.csv", hourdataKt, fieldnames = ["Qr/Qext"])

"hourkt_1960-2016.csv" will be the output filename,

---

hourdataKt is created above, this contains the calculated Kt values.

If you save a matrix, also set the column names for the resulting file. fieldnames = ["Qr/Qext"]

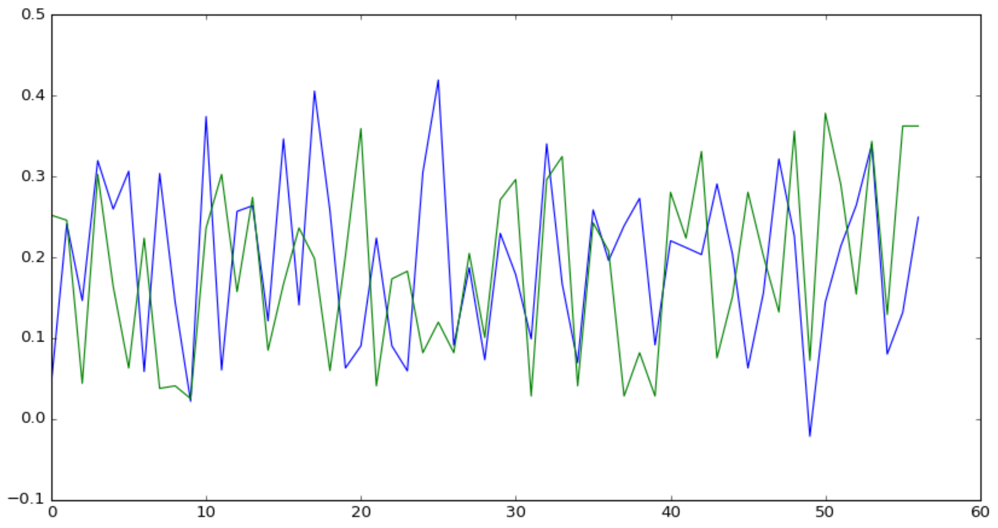
In [37]:
```
def CalcStats(Array,verbose = False):
    if len(Array) < 1:
        return 0,0,0,0,0,0
    else:
        stdev = np.std(Array)
        mean = np.mean(Array)
        median = np.median(Array)
        variance = np.var(Array)
        Min = np.min(Array)
        Max = np.max(Array)
        if verbose:
            NormDist = np.random.normal(mean,stdev,len(testhour))
            stdevErr = abs(stdev-np.std(NormDist))
            meanErr = abs(mean-np.mean(NormDist))
            medianErr = abs(median-np.median(NormDist))
            varianceErr = abs(variance-np.var(NormDist))
            print "standard deviation: {}, mean: {}, median: {}, variance: {},
 min: {}, max: {}".format(stdev,mean,median,variance,Min,Max)
            print "stdevErr: {}, meanErr: {}, medianErr: {}, varianceErr: {}".
format(stdevErr,meanErr,medianErr,varianceErr)
        return stdev,mean,median,variance, Min, Max
```

CalcStats

- This function calculates statistical properties for a given dataset.
  - **stdev,mean,median,variance, Min, Max= CalcStats(testhour,True)**
  - If the second argument is set to true, an error check is returned *may slow down process*

In [38]:
```
testhour = HourData[3006]
stdev,mean,median,variance, Min, Max= R.CalcStats(testhour,True)
bc = np.random.normal(mean,stdev,len(testhour))
plt.plot(bc)
plt.plot(testhour)
plt.show()
```

```
standard deviation: 0.107971380478, mean: 0.186531688278, median: 0.198473903
589, variance: 0.0116578190024, min: 0.0252025647972, max: 0.378063915371
stdevErr: 0.0209408620785, meanErr: 0.0014040459097, medianErr: 0.02686069712
91, varianceErr: 0.00408350786945
```



Error check

- Above block defines hour 3006 as testhour
  - the output shows: stdev,measurement - stdev,calculated. As for mean, median and variance.
  - **bc = np.random.normal(mean,stdev,len(testhour))** creates a normal distribution with midpoint:mean and scale:stdev.
    - The same is done in the CalcStats function when the second argument is True (for sanity check)
  - The generated plot shows the original Gglob,measurement/Gglob,Pysolar in blue and the generated version in green.

Note: The generated set is not the same, this is an estimation. The mean, median, variance and standard deviation are almost the same. This means that this could be an 'option' with equal likelihood.

In [39]:
```
def CalcStatsSet(HourData):
    HourStats = []
    for i in HourData:
        mu,sig,med,var,Min,Max = CalcStats(i)
        HourStats.append([mu,sig,med,var,Min,Max])
    return HourStats
HourStats = R.CalcStatsSet(HourData)
print len(HourStats)
print len(HourData)
R.ToCSV("Stats_DeBilt_1960-2016.csv", HourStats, fieldnames = ["mu","sigma","m
edian","variance","min","max"])
```

```
8784
8784
Done writing file with filename:
 Stats_DeBilt_1960-2016.csv
```

Calculating Statistical data

Because of leap years, 8784 hours are taken into account.

- The block above iterates through the [8784 x (No. years)] array
- applies the CalcStats function
- appends the result as an array [mu,sig,med,var,Min,Max] to HourStats.
  - As stated above, multiple rows are needed thus the ToCSV needs to know about this.

In [40]:
```
def GetStats(hourfile="Stats_DeBilt_1960-2016"):
    hour = []
    with open(hourfile,'r') as csvfile:
        reader = csv.DictReader(csvfile)
        for row in reader:
            hour.append([float(row["mu"]),float(row["sigma"]),float(row["media
n"]),float(row["variance"]),float(row["min"]),float(row["max"])])
    return hour

hour = R.GetKx()
RestoredStats = R.GetStats()
print len(RestoredStats)
testhour = RestoredStats[3006]
print testhour
```

```
8784
[0.10797138047834474, 0.1865316882777165, 0.19847390358855732, 0.011657819002
399483, 0.02520256479724264, 0.378063915371414]
```

This function restores the .csv file containing statistical data.

- The **RestoredStats** variable below contains exactly the same as before the save.
  - In this case, **HourStats**

In [41]:
```
sample = R.CalcRad("1960-1-1","2016-12-18",5.180,52.100,3600)
```

The R.Calcrad uses the previous result and some other arguments to create the final set

- The function will work with above arguments but more arguments are possible.
- **CalcRad(startDate,stopDate,latitude,longitude,TimeStep = 3600,KxFile="hourkt_1960-2016.csv",StatFile="Stats_DeBilt*1960-2016",TimeFormat = "%Y-%m-%d*%H:%M:%S")**
  - KxFile="hourkt_1960-2016.csv" refers to the created hourly kx calculation. You can redo this calculation with the given function and use the resulting data.
  - StatFile="Stats_DeBilt_1960-2016" refers to the created hourly stats.
  - Both files are supplied and are valid for De Bilt, The Netherlands (lat:5.18,lon:52.1)
  - TimeFormat = "%Y-%m-%d_%H:%M:%S" can be adjusted to return a different format. See this (https://docs.python.org/2/library/datetime.html#strftime-strptime-behavior) for formatting options.
  - The maximum timestep is 3600 seconds (1 hour), everything smaller can be chosen.
  - StartDate and stopDate are expected in format "2015-10-12" and are mandatory
  - latitude and longitude are mandatory
- Calculating new Kx and statistical values is advised if the location is different then De Bilt. However, within reasonal distance from De Bilt, calculations should still be accurate.

In [42]:
```
def PlotSample(sample):
    date = []
    glob = []
    Dir = []
    Dif = []
    for i in sample:
        date.append(i[0])
        glob.append(i[1])
        Dir.append(i[2])
        Dif.append(i[3])
    g = plt.plot(glob, label='global')
    gdir = plt.plot(Dir, label='direct')
    gdif = plt.plot(Dif, label='diffuse')
    plt.ylabel('Radiation [W/m^2]')
    plt.xlabel('Time dt: {} - {}'.format(date[0],date[1]))
    plt.title('Solar radiation for specific location')
    plt.grid(True)
    plt.legend()
    plt.show()
```

result

Plot below shows measured data from 1960 to 2016 and calculated data from 1960 to 2016. Some overestimations occur, but overall the approximation is relatively close.

In [43]:
```
measured = []
with open("5-180_52-100.csv",'r') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        measured.append(float((int(row["Q"])*100*100))/3600)
```

Statistics

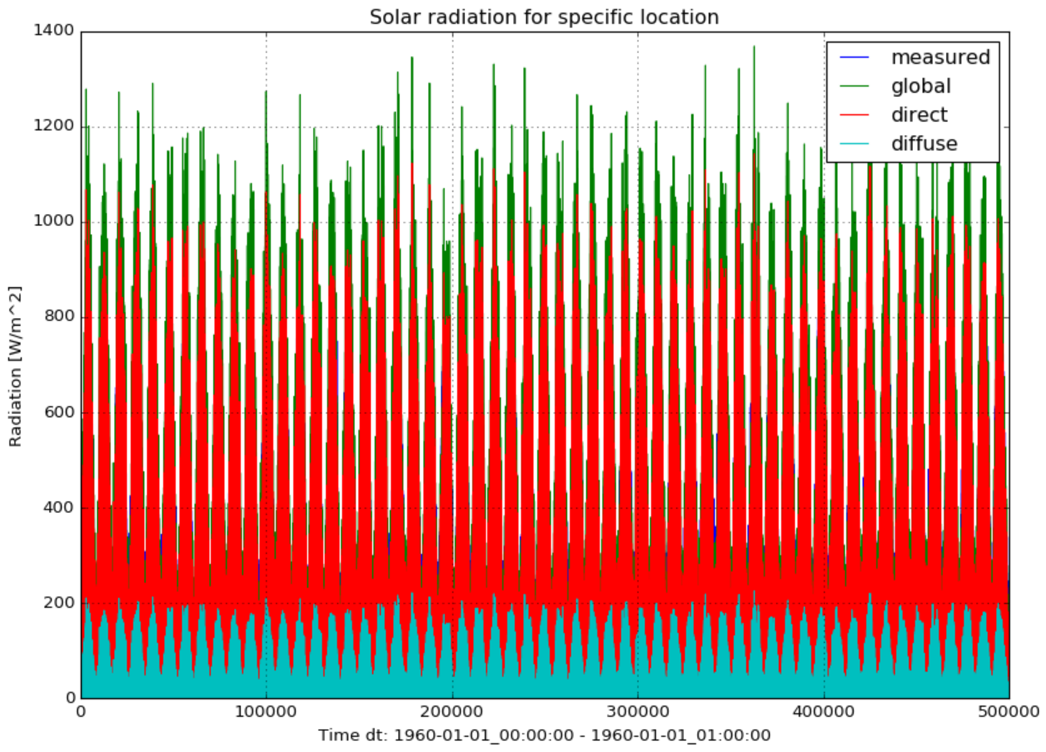
The error on standard deviation, mean and variance is small.

Median, minimum and maximum value can be inaccurate

In [44]:
```
Qgc = []
for i in sample:
    Qgc.append(i[1])
print "standard deviation:, mean:,       median:,    variance:,          min:,
 max:"
print CalcStats(Qgc)
print CalcStats(measured)
```

```
standard deviation:, mean:,       median:,    variance:,          min:, max:
(180.93034095998746, 96.514967472549287, 0.0, 32735.788279897315, 0.0, 1367.7
092375419288)
(184.55307841168047, 111.5830520421959, 2.7777777777777777, 34059.83875122788
6, 0.0, 983.33333333333337)
```

In [45]:
```
plt.plot(measured,label='measured')
PlotSample(sample)
```



Result

- All steps above are wrapped by the function below **CreateSet(FileName,Lat,Lon,dT=3600,start="1990-1-1",stop="2000-1-1",plot=False):**
- Remind that KNMI formatted data is expected;
- the needed format is:

STN,	YYYYMMDD,	HH,	T,	Q,	N
260,	19600101,	1,	80,	0,	8
260,	19600101,	2,	86,	0,	7
260,	19600101,	3,	83,	0,	8

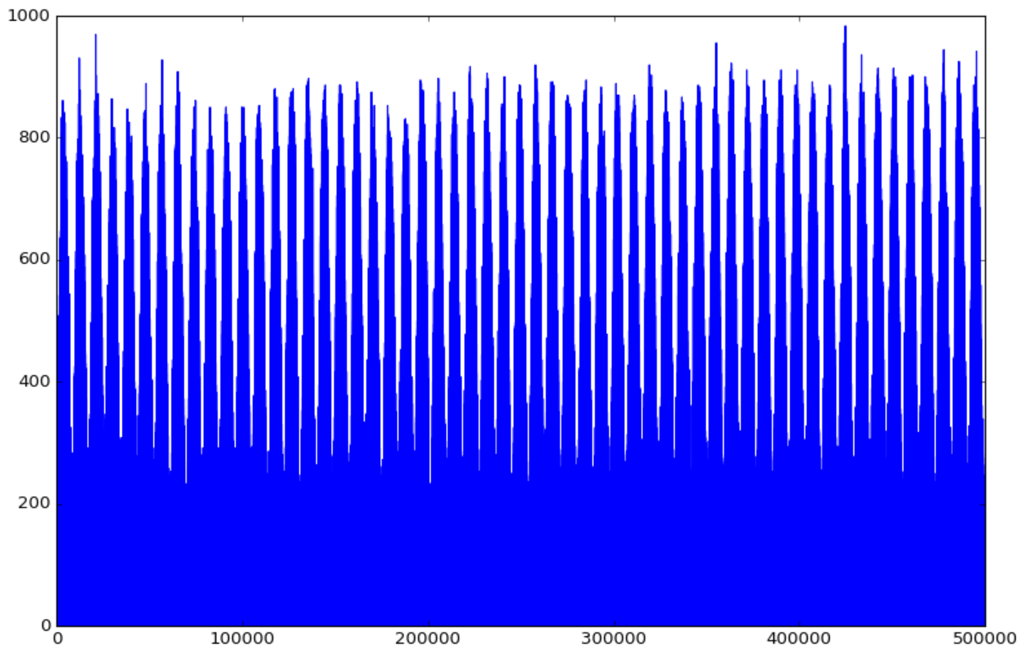
- where:
  - **STN** station number, not needed
  - **YYYYMMDD** self explanatory, *mandatory*
  - **HH** hour of day (1-24), *mandatory*
  - **T** Temperature, not needed
  - **Q** Average Radiation over hour [J/mm^2], *mandatory*
  - **N** cloud cover factor [unitless], not needed

In [46]:
```
def CreateSet(FileName,Lat,Lon,dT=3600,start="1990-1-1",stop="2000-1-1",plot=False):
    HourData,HourDataKt = CalculateLocalVariation(FileName,Lat,Lon)
    HourStats = CalcStatsSet(HourData)
    ToCSV("hourkt_%s"%FileName, HourDataKt, fieldnames = ["kt"])
    ToCSV("Stats_%s"%FileName, HourStats, fieldnames = ["mu","sigma","median","variance","min","max"])
    sample = CalcRad("1960-1-1","2016-12-18",5.180,52.100,3600)
    if plot:
        PlotSample(sample)   #[start.strftime(TimeFormat),rad,Dir,Dif,alt,azi]
    ToCSV("result_%s"%FileName, sample, fieldnames = ["date","Qg","Qdir","Qdif","alt","azi"])
    return sample
```

Plot below shows only measured data.

- for De bilt, 1960 untill 2016 in one-hour intervals.

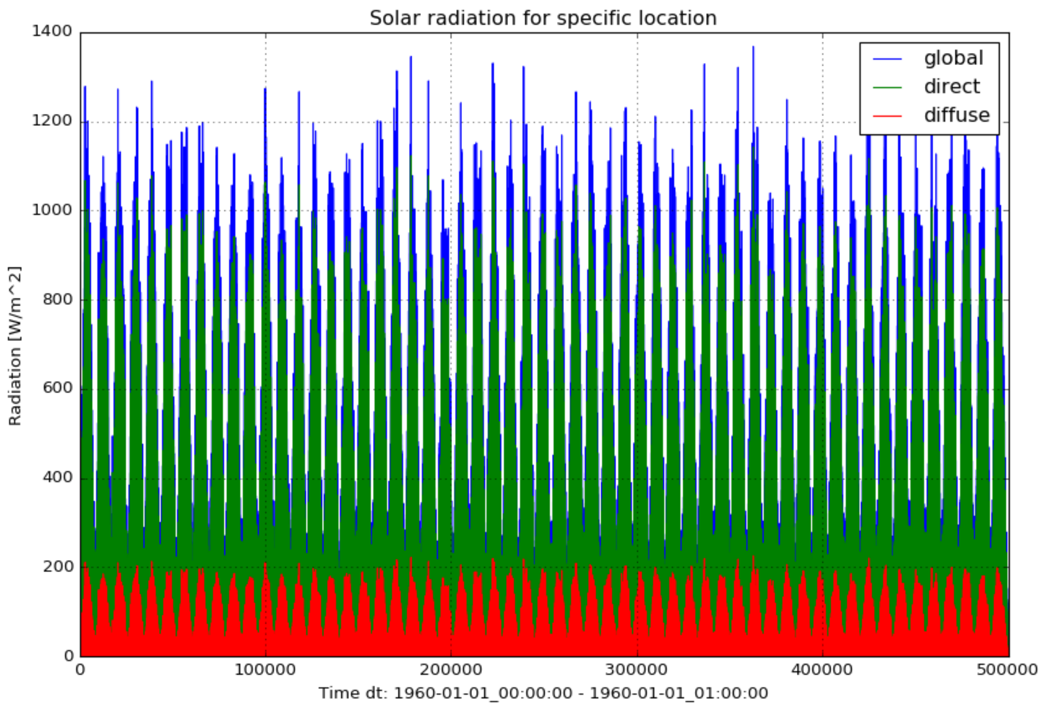
In [47]:
```
plt.plot(measured)
plt.show()
```



Plot below shows only calculated data.

- **global** is determined with PySolar module
    - **global** radiation is corrected with statistical data, based on measurements above.
- **Direct** and **Diffuse** are determined with empirical correlations which are described in *solar engineering for thermal processes*

```
In [48]: PlotSample(sample)
```


Solar radiation for specific location
Radiation [W/m^2]
Time dt: 1960-01-01_00:00:00 - 1960-01-01_01:00:00
global
direct
diffuse