Appendix 4

CODE LISTINGS

INCLUSIONS

A 4.1 TO GENERATE & PLOT NET RETURNS OF THE XAO DATASET (SEE §6.6.1)

THIS PAGE HAS INTENTIONALLY BEEN LEFT BLANK

```
# -*- coding: utf-8 -*-
# :::::::::::::::::::::::::::::::::::::::::: Start of returns.py ::::::::::::::::::::::::::::::::::::::::::::::::::
# ::::::::1:::::::::2:::::::::3:::::::::4:::::::::5:::::::::6:::::::::7:::::::::8:::::::::9:::::::::0:::::::::1:::::::::2
#:set printoptions=header:0,portrait:n,number:y,left:2pc,right:2pc,top:25mm,bottom:25mm
"""
#--------o---------o---------o---------o---------o---------o---------o---------o---------o---------o---------o
#--------o---------o-- Returns generation, plotting and soundfile generation --o---------o---------o
#--------o---------o---------o---------o---------o---------o---------o---------o---------o---------o---------o

- Converts trading or similar data to returns
- Produces correlated and decorrelated time series data
- Generates stats and graphs
- Outputs to AIFC, or perhaps WAV or direct audio streaming

history:
v0.00   20040902 initial draft
v0.01   20040922 minor bugs + rework w. better stats
v0.02   20041024 add toAIFC
v0.03   20041024 minor bugs + improve toAIFC
v0.20   20051209 objectify
v0.21   20070126 reorder for np, scipy and matplotlib
v0.22   20090110 redo plotting routines
v0.23   to sonification.com.au/audification for ICAD paper review
"""
__author__      = "David Worrall ¿ 2004++"
__version__     = "0.2.3"
__docformat__   = "restructuredtext en"

import  string, ctypes, struct
import  os.path as path                 # for file paths and universal separators.
from    datetime    import datetime
from    scipy       import stats
import  numpy   as np                   # hurray for arrays!

import  aifc, pyaudio                   # for file writing and rt audio streaming
import  pylab
import  matplotlib.pyplot   as plt      # interface to matplotlib plotting routines
import  matplotlib.ticker   as ticker   # for controlling the axis labelling
import  matplotlib.mlab     as mlab     # for plotting returns and histograms

EOF = ''                                # End of File ID for readline
BITS_15 = pow (2, 14)                   # for bit-shifting audio samples from -1 to 1 into 16 bit gamut

#--------o---------o---------o---------o---------o---------o---------o---------o---------o---------o---------o
#--------o---------o---------o-- Generic Functions --------o---------o---------o---------o---------o---------o
```

```
#--------o---------o---------o---------o---------o---------o---------o---------o---------o---------o---------o
def printStringList (listy, stringy=None, ):
    """
    Pretty-prints a list of strings.
    Good for looking for directory entries containing whatever stringy is assigned to.
    Use Eg.: printStringList (dir(), stringy = 'xxx') to find out if xxx is in the dir()
    """
    beg =''
    for i in dir(listy):
        if i[0] != beg:
            beg=i[0]
            print
        else:
            if stringy != None: print i ,
            else:
                if stringy in i:    print i

def marketTimeString (dtTuple):
    """
    Converts datetime tuple to a string in the form yyyymmdd.hhmmss.
    # Eg. datetime.datetime(2009, 1, 26, 18, 20, 5, 198934) ==> '20090126.182005'
    """
    res = str([dtTuple.year, dtTuple.month, dtTuple.day, dtTuple.hour, dtTuple.minute, \
                                        dtTuple.second])[1:-1] # dtTuple elts as a single string
    res=res.split(',')
    res = [res[0]] + [('0'+i.strip(' '))[-2:] for i in res[1:]]         # res = [yyyy,mm,dd,hh,ss]
    return string.join(res[:3], '')+'.'+ string.join(res[3:], '')       # returns "yyyymmdd.hhmmss"
# marketTime (now)

def scaleArray (nplarray, miny=-1, maxy=1, trace=False):
    """Scales a numpy array to between miny and maxy. Typically, npArray is a row of the returnsArray
    Why isn't this a numpy method?"""
    nplarray = np.array(nplarray)           # just in case! redundant if already a numpy array
    newrange = abs(float(maxy-miny))        # abs range of IP.The abs could be avoided -
                                            # it's there in case max and min are swapped on IP
    scaler = newrange/abs((nplarray.max() - nplarray.min()))
    minAdjust=(scaler * nplarray.min()) - miny
    nplarray = (scaler * nplarray) - minAdjust
    if trace:
        print "newrange:", newrange, "--- scaler:", scaler, "--- minAdjust:", minAdjust
        print "Xchecks. scaled returnsMinMax:", (nplarray.min(), nplarray.max())
    return nplarray

#--------o---------o---------o---------o---------o---------o---------o---------o---------o---------o---------o
#--------o---------o---------o- Returns class ---o---------o---------o---------o---------o---------o---------o
```

```
#--------o---------o---------o---------o---------o---------o---------o---------o---------o---------o---------o

class Returns:
    def __init__(self, IPdataFname, begDate=0, endDate=0, nr=None, field="close"):
        self.IPdatafile = IPdataFname   # the csv file of the market data. The shortname - extension
                                        # is used as part of the OPfilename. See returnsToAif method
        self.shortFname = self.IPdatafile.split(path.sep)[-1].split(path.extsep)[0]

        self.nrRecords = nr             # max number records to be read btwn begDate & endDate
        self.fieldName = field.lower()  # the dataField (column ) we want + drop all caps
        self.datefield = 'date'         # should be 0. asign to null to tell if it is present
        self.fieldNr =-1                # init for column nr of data to be pulled
        self.sep = ','                  # for csv files. Could be an IPvar, or more sophisticated

        self.returnsArray = None        # set later when size is known
        self.origDataIx         = 0     # self.returnsArray[0] Use named ix's for code readability
        self.returnsIx          = 1     # self.returnsArray[1] overwritten for clipped returns
        self.decorrReturnsIx    = 2     # self.returnsArray[2] for decorrelated version of [1]
        self.uniformIx          = 3     # self.returnsArray[3] for uniform distribution & returns
        self.normalIx           = 4     # self.returnsArray[4] for uniform distribution & returns

        self.returnsStats = None        # Stats for the returns. See self.calculateReturns()
        self.begDate = begDate          # default is 0, ergo first record if not set
        if endDate == 0:
            self.endDate = datetime.today() # default is current date, ergo  EOF if not set
            self.endDate = self.endDate.replace(hour=0, minute=0, second=0, microsecond=0)
        else:
            self.endDate = self.secDateToDT(endDate)
        try:
            self.IPfile = open(self.IPdatafile, 'r')
        except:
            print "Unable to locate ",IPfname,". If file is not in cur. dir, use full pathname."
            return
        self.getDataFormat()
        self.getDataFromFile()
        self.IPfile.close()

#--------o---------o---------o---------o---------o---------o---------o---------o---------o---------o---------o
    def getDataFormat (self, trace=False):
        """
        Reads first line of datafile for data format. Called on class instantiation.
        self.dataFormat is not referenced outside this method - so could be a local.
        """
        try:
            self.dataFormat = self.IPfile.readline()   # read first line of file
```

```
                        # should be in <date>,<open> etc format or
                        #string of the form "Date,Open,High,Low,Close,Volume,OpenInterest"

            # convert dataFormat into a simple csv string
            if self.sep in self.dataFormat:
                self.dataFormat.replace('<', "") # toss the <fieldnme> formatting, if present ...
                self.dataFormat.replace('>', "") # no harm done if not.¿check  for empty space?
            if self.fieldName in self.dataFormat.lower():
                                            # find field nr of the field (eg 'close) reqd
                self.fields = self.dataFormat.lower().split(self.sep) # drop caps, split to list

            # get the field numbers
            for i in range (len(self.fields)):
                if self.fields[i] == self.datefield:
                    self.datefield = i  # ugly type-change - but saves a var!
                if self.fields[i] == self.fieldName:
                    self.fieldNr = i
            if self.fieldNr == -1:
                print "Unable to locate ", self.fieldName, " in ", self.dataFormat, \
                      ". Check (non case-sensitive) spelling."
            if trace:
                print "datafields   :", self.dataFormat
                print "Datafield '", self.fieldName ,"' is col ", self.fieldNr, ". File is closed."
        except:
                print "Problem with data read."

#--------o---------o---------o---------o---------o---------o---------o---------o---------o---------o---------o
    def secDateToDT (self, dateString):
        """
        Converts a security date to eight-digit in the form yyyymmdd to a datedtime structure
        six-digit dates are common so we have to convert 86 to 1986 and 05 to 2005
        dateString is in eithe '861025' or '19861025' or '081025' or '20081025'
        """
        dateString = dateString.replace('/','')         # get rid of separators
        dateString = dateString [-6:]                   # toss  yyyy if present, we'll calc it.'
        if (2000 + int (dateString[:2]) )> datetime.today().year:   # it must be a 1900 date so
            dateString = int('19'+ dateString)
        else:
            dateString = int('20'+ dateString)
        tmp=divmod(dateString,10000)                    # leftmost 4 digits is the year
        m, d = divmod(tmp[1],100)
        return datetime (tmp[0],m,d)

    def getDataFromFile (self, trace=False):
        """
```

```
        Reads data from file, sorts out the date and keeps the required field.
        Uses secDateToDT() to put date in datetime format.
        """
        tbuff=self.IPfile.readline()
        count = 0
        datafromFile = []                       # list into which file data is read, elt by elt
        while tbuff != EOF and self.nrRecords < count:
            tbuff=tbuff.split(self.sep)                         # NB default for self.nrRecords = None
#            dayte = tbuff[self.datefield].replace('/',"")      # convert date string into an int
            dayte = self.secDateToDT(tbuff[self.datefield])
            datafromFile.append(string.atof(tbuff[self.fieldNr]))  # pull fielddata requird to array
            tbuff=self.IPfile.readline()                        # read 1 new line from file
            count += 1
            if trace:
                print "record Nr:", count, dayte
#       --------------- Make the np array for all the data --------------# row0 4 data, row1 4 Rnet,
        self.returnsArray=np.empty([5, len (datafromFile)], dtype=float) # row2 4 normalised
        self.nrRecords = len (datafromFile)
        if trace:
            print "Length of returnsArray = ", self.nrRecords       # len (datafromFile)
        self.returnsArray[self.origDataIx] = datafromFile           # transfer list to np array

    def calculateReturns (self):
        """
        Calculates the Net returns of in col0 and stores them in col1 of the np returnsArray.
        """
        self.returnsArray[self.returnsIx,0] = 0                     # set  return of zeroth sample
        for i in range (1,len(self.returnsArray[self.returnsIx])):
            self.returnsArray[self.returnsIx,i]=(self.returnsArray[self.origDataIx,i] \
                    - self.returnsArray[self.origDataIx,i-1])/self.returnsArray[self.origDataIx,i]
        self.doStats()          # set the records straight!
#        self.returnsArray[self.returnsIx] = scaleArray(self.returnsArray[self.returnsIx])

    def decorrelateReturns (self):
        """
        Stores a decorrelated form of the net returns in returnsArray[sourcecol]
        in returnsArray[targetcol].
        """
        self.returnsArray[self.decorrReturnsIx]= \
                                    np.random.permutation(self.returnsArray[self.returnsIx])
    def makeUniformReturns (self):
        """
        Generates a uniform random distribution over the interval [-1.0,1.0)
        Calcs the returns and stores in self.returnsArray[uniformIx]
        This is the base 'flat' distribution aganst which others canbe compared
        """
        self.returnsArray[self.uniformIx] = scaleArray(np.random.uniform(low=-1.0, high = 1.0,\
#                                                  size = self.nrRecords), trace=True)
        self.returnsArray[self.uniformIx] = np.random.uniform (low=-1.0, high = 1.0,\
                                                  size = self.nrRecords)
    def makeNormalReturns (self):
        """
        Generates a normal random distribution over the interval [-1.0,1.0)
        Calcs the returns and stores in self.returnsArray[normalIx]
        This is the distribution against which random (uncorrelated) walks can be compared
        """
        self.returnsArray[self.normalIx] = scaleArray(np.random.normal(size = self.nrRecords),\
                                                miny=-1.0, maxy = 1.0)

#--------o---------o---------o---------o---------o---------o---------o---------o---------o---------o
    def doStats (self, trace=False):
        """
        Calculates a number of common statistical properties of the returns data. Prints if nesess.
        """
        self.returnsStats = list(stats.stats.describe(self.returnsArray[self.returnsIx]))
        self.returnsStats[0] = len (self.returnsArray[self.returnsIx])
        self.returnsStats[1] = (np.min(self.returnsArray[self.returnsIx]) ,\
                                            np.max(self.returnsArray[self.returnsIx]))
        self.returnsStats[2] = np.mean(self.returnsArray[self.returnsIx])
        self.returnsStats[4] = np.var(self.returnsArray[self.returnsIx])
        self.returnsStats.append(np.median(self.returnsArray[self.returnsIx]))
        if trace:
            print "Description\n-----------"
            print" nr samples ", self.returnsStats[0]
            print" min sample ", self.returnsStats[1][0]
            print" max sample  ", self.returnsStats[1][1]
            print" arith mean  ", self.returnsStats[2]
            print " median      ", self.returnsStats[6]
            print " mode        ", stats.stats.mode(self.returnsArray[self.returnsIx])[0][0]
            print" variance    ", self.returnsStats[3]
            print" skewness    ", self.returnsStats[4]
            print" kurtosis   ", self.returnsStats[5]
#           print " kurtosos test", stats.stats.kurtosistest(self.returnsArray[self.returnsIx])
#           print " normal test", stats.stats.normaltest(self.returnsArray[self.returnsIx])

    def clipReturns (self, nrMin=0, nrMax=0, trace=False):
        """
        Clips all min values of the returnsArray to the size of the nr Min'th & nrMax'th vals.
        Used to 'limit' the returns to be audified. The default position is no clipping.
        Could be used for autoclip based on the 'significant' of SNR improvment for soing so.
```

```
        Eg clipReturns(nrMin=1,nrMax=0) limits the most neg. val to the 2nd most neg.
        Used to clip  outliers to improve SNR of sample space prior to sonification.
        If zeroing the samples is more appropriate, use threshold like this:
            tmp2array=stats.stats.threshold(tmp2array, negLimit,posLimit)
        In this usage, better to clip to nearest to maintain firmer recog. of ss.
        nrMin, nrMax are the number of samples on either end of a sorted ss to be clipped.
        """
        tmpArray = np.copy(self.returnsArray[self.returnsIx])
        tmpArray.sort()                                 # necess to make for independent sort
        negLimit = tmpArray[nrMin]                      # the value below which all are clipped
        posLimit = tmpArray[-nrMax-1]                   # the value above which all are clipped

        # fill outliers with closest inlier vals as dictated
        negIndicesToLimit = []                          # for indices of -ve vals to fix
        posIndicesToLimit = []                          # for indices of +ve vals to fix
        for index in range(len(self.returnsArray[self.returnsIx])):
            if self.returnsArray[self.returnsIx][index] < negLimit:
                negIndicesToLimit.append(index)         # find a -ve outlier
                self.returnsArray[self.returnsIx][index]=negLimit    # fill it with most -ve inlies
            if self.returnsArray[self.returnsIx][index] > posLimit:
                posIndicesToLimit.append(index)         # find a +ve outlier
                self.returnsArray[self.returnsIx][index]=posLimit    # fill it with most +ve inlies
        self.doStats()                                  # set the records straight!
        if trace:
            print "Neg limit", nrMin, " vals to", negLimit
            print "Pos limit", nrMax, " vals to", posLimit
            print "-ve indexes to fix with %f:" % (negLimit)
            for i in negIndicesToLimit:
                print "\treturns [%i] is currently %f" %(i, self.returnsArray[self.returnsIx][i])
            print "+ve indexes to fix with  %f:" % (posLimit)
            for i in posIndicesToLimit:
                print "\treturns [%i] is currently %f" %(i, self.returnsArray[self.returnsIx][i])

    def arrayToAIFC (self, np1array, OPfname,
                     nrchans    = 1,
                     SR         = 44100,
                     nrReps     = 0,
                     gapSecs     = 0,
                     buffsize   = 4096,
                     trace      = False):
        """
        Writes npArray to fname.AIFC. Filename contains original DataFilename datetime-stamp.
        buffsize is size of AIFC write buffer. File writes are in buffsize increments.
        Write the arran nrReps number of times, with gapSec seconds between
        """
```

```
        OPfileID = aifc.open(OPfname+'.aifc', 'wb')         # open the file and set up the header
        OPfileID.setnchannels(nrchans)                      # the number of channels (1=mono)
        OPfileID.setsampwidth(2)                            # the sample width in bytes(16bit fmt)
        OPfileID.setframerate(SR)                           # the frame rate. for mono, FR =SR
                                                            # scale data -1 to 1
                        # scale (-1,1) the np.array, shift it 15 bits and overwrite it as int16s
        npSarray=BITS_15*np1array                          # NB can be a np.array. Doesn't have 2B
        npSarray=npSarray.astype('int16')                   # cvt to 16 bit ints.
                                                            # Use nparray.dtype say what type it is
                        # to python list - has to be a better way!!!!! ##
                        # try to pack it directly from the np1aray, as per pyAudioEg.py
        tmp = npSarray.tolist()
        dataString = "".join(struct.pack('>h', i) for i in tmp) # pack the array data into a buffer
        OPfileID.writeframes(dataString)                    # write buffer to file
                        # if the sound it to be repeated,insert a gapsSec silence b4 writing again
        if nrReps > 0:
            tmp = np.array (np.zeros(gapSecs * SR * nrchans, dtype='int16'))
            silentString = "".join(struct.pack('>h', i) for i in tmp) # pack zeros into a buff
            for i in range (nrReps):
                OPfileID.writeframes(silentString)          # write the silence buffer
                OPfileID.writeframes(dataString)            # write the non-silent buffer again
        OPfileID.close()
        if trace:
            numsamps = int(len(np1array)) + (gapSecs * SR * nrchans)
            print "Wrote %s.aifc, nrChans: %d, nrSamps: %d." % (OPfname, nrchans, numsamps)

#--------o---------o---------o---------o---------o---------o---------o---------o---------o---------o
    def outputAudioFiles (self, OPdir=None, SR = 44100, gapSecs = 0, trace=False):
        """
        Write the corr and decorr rtns and  same size uniform and normal returns
        OPdir is a string of the directory to which the .aifc files is output.
        """
        # Make sure the OPdir given is actually a directory
        if OPdir:
            if OPdir[-1] != path.sep:                       # '/' on unix etc, '\' on windows
                OPdir += path.sep                           # in case it has been left off
        if not path.isdir (OPdir):
            print "The directory to write to cannot be found."

        # Construct the filenames
        shortFname = self.IPdatafile.split(path.sep)[-1].split(path.extsep)[0]
        now = marketTimeString(datetime.today())            # 'yyyymmdd.hhmmss'
#        now= string.split(now, '.')[0]                      # drop the mins & secs

        # Make the output file names
```

```
        rtnFname    = OPdir + shortFname+'-ReturnsRaw_'+ now    # for rtns
        decorrFname = OPdir + shortFname+'-RtnsDecorr_'+ now    # for decorrelated rtns
        uniformFname= OPdir + shortFname+'-RtnsUniform_'+ now   # for white random rtns
        normalFname = OPdir + shortFname+'-RtnsNormal_'+ now    # for normal rtns
        CandDFname  = OPdir + shortFname+'-CorrAndDecorr_'+ now # for alternate A/B comparison
        UNDRtnFname = OPdir + shortFname+'-uniNormDecorRtn_'+ now # all 4 in sequence

        ret1dc1Fname = OPdir + shortFname+'-ret+2dcorr1_'+ now   # 1 x ret + 2 x decorr nr2
        ret1dc2Fname = OPdir + shortFname+'-ret+2dcorr2_'+ now   # 1 x ret + 2 x decorr nr2
        ret1dc3Fname = OPdir + shortFname+'-ret+2dcorr3_'+ now   # 1 x ret + 2 x decorr nr3

        # Concatenate the composite buffers (raw and Decorrelated + UNDR
        b0 = np.zeros (int(gapSecs*SR))
        b1 = scaleArray(self.returnsArray[self.returnsIx], miny=-1.0, maxy = 1.0)
        b2 = scaleArray(self.returnsArray[self.decorrReturnsIx], miny=-1.0, maxy = 1.0)
        b3 = scaleArray(self.returnsArray[self.uniformIx], miny=-1.0, maxy = 1.0)
        b4 = scaleArray(self.returnsArray[self.normalIx], miny=-1.0, maxy = 1.0)

        retAndDecorrArray = np.concatenate((b1,b0,b2))
        UNDRtnArray = np.concatenate((b3, b0, b4, b0, b2, b0, b1))

        # de-correlate the decorrelation, and again
        b5 = np.concatenate((b1, b0, np.random.permutation(b2), b0, np.random.permutation(b2)))
        b6 = np.concatenate((np.random.permutation(b2), b0, b1, b0, np.random.permutation(b2)))
        b7 = np.concatenate((np.random.permutation(b2), b0, np.random.permutation(b2), b0, b1,))

        if trace:
            print "Range cross-check:"
            print "     Array           Min    Max"
            print " ____________________________________"
            print "     ReturnsRaw_    %.2f   %.2f" %  (b1.min(), b1.max())
            print "     RtnsDecorr_    %.2f   %.2f" %  (b2.min(), b2.max())
            print "    RtnsUniform_    %.2f   %.2f" %  (b3.min(), b3.max())
            print "     RtnsNormal_    %.2f   %.2f" %  (b4.min(), b4.max())
            print "  CorrAndDecorr_    %.2f   %.2f" %  (retAndDecorrArray.min(),
                                                       retAndDecorrArray.max())
            print "uniNormDecorRtn_    %.2f   %.2f" %  (UNDRtnArray.min(), UNDRtnArray.max())
            print "   ret+1xdcorr1_    %.2f   %.2f" %  (b5.min(), b5.max())
            print "   ret+1xdcorr2_    %.2f   %.2f" %  (b6.min(), b6.max())
            print "   ret+2xdcorr3_    %.2f   %.2f" %  (b7.min(), b7.max())

        # send multiples of the individual arrays  off to the audio OP routine
        self.arrayToAIFC (b1, rtnFname,     SR=SR, nrReps = 3, gapSecs = 1, trace=trace)
        self.arrayToAIFC (b2, decorrFname,  SR=SR, nrReps = 3, gapSecs = 1, trace=trace)
        self.arrayToAIFC (b3, uniformFname, SR=SR, nrReps = 3, gapSecs = 1, trace=trace)
```

```
        self.arrayToAIFC (b4, normalFname,  SR=SR, nrReps = 3, gapSecs = 1, trace=trace)
        # send combinations of the individual arrays   to the audio OP routine
        self.arrayToAIFC (retAndDecorrArray, CandDFname,    SR=SR, nrReps=3, gapSecs=2, trace=trace)
        self.arrayToAIFC (UNDRtnArray, UNDRtnFname,         SR=SR, nrReps=3, gapSecs=2, trace=trace)

        # send 2x different decorrelates + raw return to the audio OP routine
        self.arrayToAIFC (b5, ret1dc1Fname,  SR=SR, trace=trace)
        self.arrayToAIFC (b6, ret1dc2Fname,  SR=SR, trace=trace)
        self.arrayToAIFC (b7, ret1dc3Fname,  SR=SR, trace=trace)

        print "Finished writing audio files to", OPdir + shortFname
        # send multiples of the individual arrays  off to the audio OP routine
#        self.arrayToWAV (b1, rtnFname,     SR=SR, nrReps = 3, gapSecs = 1, trace=True)
#        self.arrayToWAV (b2, decorrFname,  SR=SR, nrReps = 3, gapSecs = 1, trace=True)
#        self.arrayToWAV (b3, uniformFname, SR=SR, nrReps = 3, gapSecs = 1, trace=True)
#        self.arrayToWAV (b4, normalFname,  SR=SR, nrReps = 3, gapSecs = 1, trace=True)
        # send combinations of the individual arrays   to the audio OP routine
#        self.arrayToWAV (retAndDecorrArray, CandDFname,    SR=SR, nrReps=3, gapSecs=2, trace=True)
#        self.arrayToWAV (UNDRtnArray, UNDRtnFname,         SR=SR, nrReps=3, gapSecs=2, trace=True)
        # send 2x different decorrelates + raw return to the audio OP routine

#--------o---------o---------o- End of Returns Class ------o---------o---------o---------o---------o

#--------o---------o---------o---------o---------o---------o---------o---------o---------o---------o
#--------o---------o---------o---- Play Audio Routines ----o---------o---------o---------o---------o
#--------o---------o---------o---------o---------o---------o---------o---------o---------o---------o

# mono test tone @ 44.1KHz
def sineTone (Fname="sine", freq=1000, dur=1,SR = 44100):
    abscissa = np.linspace(0,1,SR)                          # abscissa  is the x of y = f(x)
    data = np.sin(abscissa *2 * np.pi* freq)                # data      is the y of y = f(x)
    return data
    arrayToAIFC (data,OPdir+str(freq), trace=False)          # if Fname doesn't contain full path,
                                                            # file will be written to current dir

#--------o---------o---------o---------o---------o---------o---------o---------o---------o---------o
#--------o---------o---------o- Plotting Routines ---------o---------o---------o---------o---------o
#--------o---------o---------o---------o---------o---------o---------o---------o---------o---------o
# NB All plotting routines need a call to pylab.show() to display the plot
def plotXAO ():
    pylab.plot(xao.data, 'g.')                  # dots are '.'  ; pixels are ','
    pylab.xlabel('Contiguous Trading Days')
    pylab.ylabel('XAO Closing Value')
    pylab.grid(True)
#    pylab.savefig('XAOpix')                    # use if .png file is required
```

```
451
452 def plotXAOreturns (col=1):
453     """
454     Plots the returns. col=1 is the returns, col=2 is the decorrelated returns.
455     """
456     pylab.plot(xao.returnsArray[col],'g-')      # dots are '.'  ; pixels are ','
457     pylab.xlabel('Contiguous Trading Days')
458     pylab.ylabel('XAO Net Returns')
459     pylab.grid(True)
460 #     pylab.savefig('XAOnetRtns')               # use if .png file is required
461     pylab.savefig('XAOdecorrelatedReturns')     # use for decorrelated signal
462
463 def histMeanLine (color='#FF0000', linestyle='-', width = 1, yminmax=(0,100)):
464     """
465     Draws a vertical line t at or near the mean.
466     'Near to' is there for situations where it would improve visibility of dataplots at the mean.
467     """
468     fudge = xao.returnsStats[2]+ 0.0002         # add a small o'set, when necess.
469     meanLineX = [fudge,fudge]
470     meanLineY = yminmax
471     pylab.plot(meanLineX,meanLineY, color=color, linestyle=linestyle,\
472                                         linewidth=1,  label='mean') # ¿ '_nolegend_' ?
473
474 def histXAOreturns (col=1, nrbins=10, color='green'):
475     pylab.hist(xao.returnsArray[col], nrbins, facecolor=color, label='actual returns',alpha=0.25)
476 #     histMeanLine(fmat='r-', width=1, yminmax=(0,490))           # alternative to that below
477     histMeanLine(color='#ff0000', linestyle='-', width=1, yminmax=(-20,480))
478 #     pylab.axis([-0.35,0.07, -30,890])                       # alternative to that below
479     pylab.axis([-0.35,0.07, -20,500])                         # use this when there's more bins
480     pylab.savefig('XAOhist+NormHist')                         # use if .png file is required
481
482 def histXAOclippedReturns (col=1, nrbins=10, color='green'):
483     pylab.hist(xao.returnsArray[col], nrbins, facecolor=color, \
484                label='clipped returns', align='mid', alpha=1)
485     #histMeanLine(color='#FF0000', linestyle='-', width=1, yminmax=(-20,480))
486     pylab.axis([-0.1,0.07, -30,450])
487     pylab.savefig('XAOclipped+NormHist')              # use if .png file is required
488
489 #--------o---------o---------o---------o---------o---------o---------o---------o---------o---------o
490 def histNormal (color='000000', trace = False):
491     """
492     Plot a normal distribution with mu and sigma sa,e ar original or clipped dataset.
493     To place mu at the centre of the abscissa space, use:
494     #mu = xao.returnsStats[1][0]+ (0.5* (xao.returnsStats[1][1] - xao.returnsStats[1][0]))
495     """
```

```
496     mu=xao.returnsStats[2]
497     sigma = np.sqrt(xao.returnsStats[3])                          # sdev-[3] is variance
498     normSamps = np.random.normal(mu, sigma, xao.returnsStats[0])    # [0] is nr datapoints
499     dta , smallest, binsize, ignore = stats.histogram (normSamps, nrbins)
500     histy, binEdges = np.histogram (normSamps, bins=nrbins, normed=0, new=False)
501 #     pylab.plot(binEdges,histy, 'b-', linewidth=1, color='#888800', antialiased=True,\
502 #           label='normal distribution,\nwith same St.Dev.', alpha=1)
503     pylab.hist(normSamps, nrbins, facecolor=color, align='mid' ,bottom=None, \
504                label='simulated normal\ndistrib. with same StD.', alpha=1)
505     if trace:
506         print "histy:", np.sum(histy) # ,histy
507         print "binEdges", np.sum(binEdges) #, binEdges
508         print 'size of normSamps:', len(normSamps)
509         print ' sum of hist:', histy.sum()
510         print "over same abscissa:", np.min(normSamps), np.max(normSamps)
511         print "over SND histogram",  np.min(histy), np.max(histy)
512
513 def xAxis ():
514     ax=pylab.subplot (111)
515     majLoc = pylab.MultipleLocator(0.02)
516     majFmat = pylab.FormatStrFormatter('%f') # or some other format - this puts the numbers on
517     ax.xaxis.set_major_locator(majLoc)
518 #     ax.xaxis.set_major_formatter(majFmat)
519
520     minLoc = pylab.MultipleLocator(0.01)
521     ax.xaxis.set_minor_locator(minLoc) #for the minor ticks, use no labels; default NullFormatter
522
523 def yAxis ():
524     ay=pylab.subplot (111)
525     majLoc = pylab.MultipleLocator(50)
526     majFmat = pylab.FormatStrFormatter('%d') # or some other format - this puts the numbers on
527     ay.yaxis.set_major_locator(majLoc)
528     ay.yaxis.set_major_formatter(majFmat)
529
530     minLoc = pylab.MultipleLocator(10)
531     ay.yaxis.set_minor_locator(minLoc) #for the minor ticks, use no labels; default NullFormatter
532
533 #     pylab.axhline(y=-50, color='black') # draw a line - why not!
534
535 def legend ():
536     pylab.legend(loc='center left') #, size='medium')
537 #     pylab.legend(loc='top left') #, size='medium')
538
539 def labels ():
540     s = 'Decorrelated Net Returns (%d bins)' % (nrbins)
```

```
    pylab.xlabel(s)
    pylab.ylabel('Frequency of occurrence')

#--------o---------o---------o- End of Plotting Routines --o---------o---------o---------o---------o

#--------o---------o---------o---------o---------o---------o---------o---------o---------o---------o
#--------o---------o---------o- __main__() code -o---------o---------o---------o---------o---------o
#--------o---------o---------o---------o---------o---------o---------o---------o---------o---------o
xao = Returns ("XAO_5725.csv", field ='close') # get original dataset from file etc
#plotXAO ()                                    # plots the original dataset

xao.calculateReturns ()                       # calcs the Prob. Density Fn (pdf)
#plotXAOreturns (col=1)                       # plots the pdf of the correlated returns
xao.clipReturns (1,0)                         # the dataset with the LH datapoint removed
xao.doStats (trace=True)                      # stats have changed, so reset the record!
xao.decorrelateReturns ()                     # decorr rtns[returnsIx] into retns[decorreatedRtnsIx]
xao.makeUniformReturns()
xao.makeNormalReturns()

#xao.outputAudioFiles (OPdir='/Users/drw/Desktop', trace=True) # output all returns to AIF files
xao.outputAudioFiles ('/Users/drw/Desktop/XAO-5725audiofiles/',
                                          SR = 16000,
                                          gapSecs = 1.0,
                                          trace = False)  # output all returns to AIF files

"""
The works and is used - just commented out for the mo. should be part of doit()
# uncomment the commands below to generate what's needed
plotXAOreturns (col=2)                          # plots the pdf of the correlated returns

    # _____________________________- MAKE THE HISTOGRAMS - _____________________________o
                    #calc for space just of most -ve sample:
                    # total s.sppce = 0.333204213427 + 0.0588620748375 = 0.39206628826450002
                    # diff btw least two: 0.33320421 - 0.08807546 = 0.24512875000000001
                    # proport. of total space = 0.24512875/0.39206628826450002= 0.6252227170182727

nrbins=200                                      # nr is somewhat arbitrary,easily changed!
#nrbins = int (200/(1- 0.625))                   # use this to keep the bin size the same for
#histXAOreturns(nrbins, color='#888888')         # compute the returns from the original dataset
#histXAOreturns(col=2,nrbins=nrbins, color='#00ff00')
                                                # the raw and the clipped.
#histXAOclippedReturns(col=1, nrbins=nrbins, color='#00ff00')
#histMeanLine(fmat='g-')
#histNormal(color='#000000')
legend()
```

```
labels()
yAxis()
#xAxis()
#pylab.grid(True)
#pylab.axhline(y=-50, color='black') # draw a line - why not!

#pylab.axis('off')
#pylab.savefig('XAOhist+NormHist')             # use if .png file is required
#pylab.savefig('XAOdecorrelatedReturns')
pylab.show()

"""
# results of the stats analysis = for the record
"""
==================================

Description b4 clip
-----------
 nr samples  5725
 min sample   -0.333204213427
 max sample   0.0588620748375
 arith mean   0.000217488453128
 variance     9.56858813276e-05
 skewness    -7.64911826438
 kurtosis   241.729886112
---
calc for space just of most -ve sample:
total s.sppce = 0.333204213427 + 0.0588620748375 = 0.39206628826450002
diff btw least two: 0.33320421 - 0.08807546 = 0.24512875000000001
proport. of total space = 0.24512875/0.39206628826450002  =   0.6252227170182727
---
Description after clip
-----------
 nr samples  5725
 min sample  -0.088075456212
 max sample   0.0588620748375
 arith mean   0.000260305703297
median       0.000236144211932
mode         0.0
 variance     7.76242316164e-05
 skewness    7.76106727987e-05
 kurtosis   12.2617257637
"""
```

THIS PAGE HAS INTENTIONALLY BEEN LEFT BLANK

A 4.2 TO CLEAN HIGH–FREQUENCY TRADING–ENGINE DATASET (SEE §6.8.1)

THIS PAGE HAS INTENTIONALLY BEEN LEFT BLANK

```
# -*- coding: utf-8 -*-
# :::::::::::::::::::::::::::::::::::::::::: CMdataCleaner:.py ::::::::::::::::::::::::::::::::::::::::::::::::::
# ::::::::1:::::::::2:::::::::3:::::::::4:::::::::5:::::::::6:::::::::7:::::::::8:::::::::9:::::::::0:::::::::1:::::::::2
#:set printoptions=header:0,portrait:n,number:y,left:2pc,right:2pc,top:25mm,bottom:25mm
#for options in vim use :help popt¿option
"""
CMDataCleaner.py cleans the original 65 day files of CMdata, saving the cleaned files to a different directory.
"""
import glob, string

def trunc(x,n=0):
        """ truncates x, to n decimal places ... works for +ve and -ve floats and ints """
        if x> 0:
                return  round( x -0.5 * 10**-n,n)
        else:
                return -round( abs(x) -0.5 * 10**-n,n)

def fdata_to_list(fid):
        """Reads file by line into a list."""
        fid.seek(0,0)
        fdata=fid.readlines()
        return fdata

def printdata(fdata):
        """Prints file data list line by line."""
        count = 0
        for line in fdata:
                print count, line
                count = count + 1

IPdir="/Users/drw/Documents/R_DATA/CMdataOrig/"   # IP dir of sourcefile
OPdir="/Users/drw/Documents/R_DATA/CMdataClean/"  # OP dir of cleaned file

orderTypes={
                'ENTER': 11,
                'DELET': 5,
                'AMEND': 15,
                'TRADE':  10,
                'OFFTR': 11,
                'CANTR': 11,
                'FIELD': 2, #actually needs 3 but cludge here to trigger a fix for field 3
                'CONTL': 6,
                'FINIS': 3
                }
#print orderTypes['FINIS'] , orderTypes.has_key('FINIS') # for testing
```

```

def priceCheck (order, priceField, volField, valueField):
        """ Checks whether priceField * volField = valueField to .... 2 decimal places"""
        try:
                not (round(string.atof(order[priceField][1:]) \
                                * string.atof(order[volField]),2) \
                                == round(string.atof(order[valueField]),2))
        except:
                print "PriceCheckError:", order

global NrErrors ; NrErrors = 0
def fixOrder(order):
        """ corrects known errors in various error types """
        global NrErrors
        NrErrors +=1
        if order[1]=='ENTER':
                priceCheck(order,5,6,7)
        elif order[1]=='DELET': pass
        elif order[1]=='AMEND':
                try:
                        order=order[:6] + order[7:11] + order[14:]      # delete last field
                        #fields are now  <timeStamp>,AMEND,<securityID>,Bid|Ask,<oldOrderID>,
                        #<newOrderID>,<newPrice>,<newDisclosedVolume>,<newDisclosedValue>
                except:
                        print "AMEND conversion error", order
                # X-check that <newPrice>*<newDisclosedVolume>=<newDisclosedValue>
                priceCheck (order,6,7,8)
        elif order[1]=='TRADE':
                try:
                        order=order[:5] + [string.joinfields(order[5:-4],"")[1:]] +order[-4:]
                        #fields are now <timeStamp>,TRADE,<securityID>,<tradeID>,
                        #<Price>,<Volume>,<bidOrderID>,<askOrderID>
                except:
                        print "TRADE conversion error", order
                # X-check that <newPrice> * <newDisclosedVolume> = <newDisclosedValue>
                priceCheck (order,4,5,6)
        elif order[1]=='OFFTR':
                priceCheck (order,5,6,7)
        elif order[1]=='CANTR':
                priceCheck (order,5,6,7)
        elif order[1]=='FIELD':
                try:                                        #NB assumes the GN= a single digit numeral
                        temp=string.splitfields(string.strip(order[2]),' ')
                        order=order[:2]+[temp[0]]+[temp[-1][-1]]
                except:
```

```
                        print "FIELD conversion error:", order
        elif order[1]=='CONTL':
                try:                                #turn last field into 3 fields
                        temp=string.split(order[-1])
                        order=order[:-1]+[temp[0]]  + [string.join(temp[1:])]
                except:
                        print "CTRL conversion error:", order
        elif order[1]=='FINIS':
                try:
                        order=order[:2]                 # just timeStamp and 'FINIS'
                except:
                        print "FINIS conversion error:", order
        else: print " can't fix order of type", order[1]
        return order

def fixNewline(order):
        """ ensures that the last item in the order list is a \n and that it is the only one """
        if order[-1] != '\n':   # if id the last item in the order list is not already only a \n
                try:
                        if order[-1][-1]=='\n':          #if the last char in the last item is a \n
                                order[-1]=order[-1][:-1] # drop the \n
                        order += ['\n']                  # add a \n to the list
                except:
                        "fixNewline error", order
        return order

def cleanFiles():
        """
        Opens all the files in the IPdir and writes clean files of the same name into the OPdir.
        Reads & writes files line-at-a-time so can handle large files.
        """
        filelist=glob.glob(IPdir+"*.csv")
#       print filelist
        print
        print "!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"
        filecount=0
        for filename in filelist:
                try:
                        IPfileID=open(filename,'r')
                        shortFname=filename[string.rfind(filename,"/")+1:] # just the fname not the dir
                        print "Processing:", shortFname
                except:
                        "Can't open", filename
                        break
                OPfileID=open(OPdir+shortFname,'w',0)
```

```
                orderCount = errorCount = 0
                while 1:
                        order=IPfileID.readline()
                        if not order:
                                break                           # stop and EOF
                        order = string.splitfields(order,',')   # orders are 1 per line, of diff types
                        if orderTypes.has_key(order[1]):
                                order=fixOrder(order)
                                order=fixNewline(order)
                                OPfileID.write(string.joinfields(order,',')) # order to OPfile as a.csv
                                orderCount += 1
                        else:
                                print " orderType not recognised:", order
                                errorCount +=1
                print " Number of orders processed: ", orderCount,
                print " ...... Number of errors:", errorCount
                print "-------"
                IPfileID.close()
                OPfileID.close()
                filecount = filecount + 1
        print "Number of files processed: " , filecount
        print "FIN"

cleanFiles()    # Just do it!
```

A 4.3 TO CREATE HDF5 DATABASE OF ALL HDTF DATA (SEE §6.8)

THIS PAGE HAS INTENTIONALLY BEEN LEFT BLANK

```
# -*- coding: utf-8 -*-
# ::::::::::::::::::::::::::::::::::::::::: CMpyTables.py ::::::::::::::::::::::::::::::::::::::::::::::::::::::
# :::::::1:::::::::2:::::::::3:::::::::4:::::::::5:::::::::6:::::::::7:::::::::8:::::::::9:::::::::0:::::::::1:::::::::2
#:set printoptions=header:0,portrait:n,number:y,left:2pc,right:2pc,top:25mm,bottom:25mm
#for options in vim use :help popt¿option
#
# pyTables file and group classes for CM65 order data in HDF5 format
# last revision: drw 20070811
"""
Defines a H5file class which is a superstructure around a HDF5 file created with PyTables for the CM65 data.
The pyTable leaf classes for orderbook entries are also defined here.
orderbook structure: two techniques:
 - a 3D structure which represents the state of the orderbook at each point of change
       need to check whether pyTables/numpy can handle 3D array w. 1D blank
       day time (price vol) x n
    or can it handle a Dict?

The overall structure of the HDF5 datafile
------------------------------------------
root/MessageTexts                       # A Node with MessageText leaves
root/securitiesInfo/SecID12346Leaf      # The SecuritiesInfo node is created both as a h5 file node
                   /SecID22346Leaf      # and as a python a RAM Dict called self.SecurityNodesDict
.....                                   # This may eventually be eliminated.
root/securityID/OnTrades/OnTrade_Leaf1
                        /OnTrade_Leaf2
                        /.....
                /OffTrade/OffTrade_Leaf1
                        /OffTrade_Leaf2
                        /.....
                /CancelledTrades/CanceTrade_Leaf1
                        /CanceTrade_Leaf2
                        /.....
                /EnteredOrder/EnterOrder_Leaf1
                             /EnterOrder_Leaf2
                             /.....
                /DeletedOrder/DeleteOrder_Leaf1
                             /DeleteOrder_Leaf2
                             /.....
                /AmendedOrder/AmendOrder_Leaf1
                             /AmendOrder_Leaf2
                             /.....
                /Messages/MessageReference_Leaf1
                             /MessageReference_Leaf2
                             /.....

root/securityID/ (etc)
"""
import string, tables, numpy        # for tables

# :::::::1:::::::::2:::::::::3:::::::::4:::::::::5:::::::::6:::::::::7:::::::::8:::::::::9:::::::::0:::::::::1:::::::::2
#-------------- HDF5 file class definition --------------------
class H5file:
    def __init__(self, filename):
        self.fileName = filename    # the name string of the H5 file. It's usual for it to have a .h5 extension.
        self.shortFileName = self.fileName[string.rfind(self.fileName,"/")+1:] # just filename.ext not the directory
        self.fileID = None          # returned by the opening definitions. Used to access the file.
        self.openError = "Can't open ", self.fileName   # for the HDF5 file
        self.mode = None            # the mode of the the file opening. 'w' = write, 'a' = append, 'r' = read
        self.cntrlMessages = {}     # a dictionary whose keys() are CNTRL and FIELD messages found across all dayfiles.
        self._cntrlMessageNr = 0    # the incremental count of a number of unique CNTRL and FIELD flags.
                                    # Internal use only. See getFlagNumber().
        self.securitiesDict = {}    # A dict of tuples for each Node/Group in the root directory.
                                    # each tuple is in the form (CMgroupID, firstDay, lastDay,  ....)
                                    # should this Dict be held in RAM and written to the h5 file before close?

    def wOpen(self):
        """ Opens the HDF5 format file, if closed.
        Makes new one if filename doesn't exist. Returns the file descriptor."""
        if not self.fileID:     # no point trying to open it once it's already open
            try:                # assume it exists and we're appending
                self.fileID = tables.openFile(self.fileName, mode = "a", title = self.fileName)
                self.mode = self.fileID.mode    # do we need to keep a record of this?
            except:
                try:            # it didn't exist, so create it and open in write mode.
                    self.hdf5fileID = tables.openFile(self.fileName, mode = "w", title = self.fileName)
                    node0   = self.fileID.createGroup("/", "MessageTexts", "CM65 data for"+ nameString)

                    self.mode = self.fileID.mode        # do we need to keep a record of this?
                except:
                    print "Problem opening HDF5 file", self.fileName, " for writing."

    def rOpen(self):
        """ Opens the HDF5 format file  for reading, if closed. Returns the file descriptor."""
        if not self.fileID:
            try:
                self.fileID = tables.openFile(self.fileName, mode = "r", title = "CM65 data") # assume it exist
                self.mode = self.fileID.mode            # do we need to keep a record of this?
            except:
                print "Problem opening HDF5 file", self.fileName, " for reading."
```

```
    def close(self):
        if self.fileID:
            try:
                self.fileID.close()     # assumnes all tables have been flushed
                self.fileID = None      # reset
                self.mode = None        # reset NB instance data persists w. some (filename etc) data
            except:
                    "Trouble in closing HDF5 file", self.fileName

# :::::::1:::::::::2:::::::::3:::::::::4:::::::::5:::::::::6:::::::::7:::::::::8:::::::::9:::::::::0:::::::::1:::::::::2
# This message stuff should probably be methods of the DayFile class,
                                            #?? except for the writing of the Dict to the H5 file.
    def getMessageNumber(self, message):
        """Updates the self.cntrlMessages dictionary. Keys are unique counting numbers. These (_cntrlFlagNr) numbers
        are incremented during the loading routines. NB the number is the message and the Dict key is the number.
        The complete MessagesDict is written to a MessagesNode in the H5 file and the number is included in
        the individual security H5 node."""
        if not message in self._cntrlMessages.keys():   # add a new message
            self._cntrlMessageNr += 1
            self._cntrlMessages[message] = self._cntrlMessageNr
        return self._cntrlMessages[message]      # return the number pointed to by the Dict index (the message)

#    def makeNewRootNodeDict(self):
#        """ Makes a new NodeList of the root directory."""
#        self.rootNodesDict = {}     # zero the list
#        if self.fileID:         # i.e. there is an hdf5 file open
#            for node in self.fileID.root:
#                self.rootNodes[node._v_name]=()

    def printNodeInfo(self):
        """ Prints the entire node structure of the file."""
        for node in self.fileID.walkNodes():
            print node

    def appendSecurityBranch(self, nameString="s07007", description="Jame Bonds", CMgroup=0):
        """ Appends a new security branch to the root directory if it doesn't exist.
        Adds the sub-branch nodes to the securityID node to accommmodate order book info.
        Updates rootNodes dict."""
        if self.fileID.isopen:
            if not nameString in self.securitiesDict.keys():       # If it's already at self.fileID, leave it alone.
                try:
                    node0   = self.fileID.createGroup("/", nameString)
                    #self.securitiesDict[node0._v_name]=(CMgroup,)
                                        # add it to the self.SercuritiesDict !!!!NB for generating a summary
                    # print "securitiesDict: Node: ", node0._v_name, self.securitiesDict
```

```
                except:
                    print "File:", self.fileName, ": Can't create a node for " , nameString
                    self.printNodeInfo()
                try:    # Append the Orderbook nodes to the securityID branch.
                                                # "nodeX" assignment to prevent printing. Alternative?
                    node1   = self.fileID.createGroup("/"+nameString, "OnTrade")      # for OnTrade leaves
                    table1  = self.fileID.createTable(node1, "OnTradeTable", OnTrade)
                    node2   = self.fileID.createGroup("/"+nameString, "OffTrade")     # for OffTrade leaves
                    table2  = self.fileID.createTable(node2, "OffTradeTable", OffTrade)
                    node3   = self.fileID.createGroup("/"+nameString, "CancelTrade")  # for CancelTrade leaves
                    table3  = self.fileID.createTable(node3, "CancelTradeTable", CancelTrade)
                    node4   = self.fileID.createGroup("/"+nameString, "Bid")          # for EnterBidOrder leaves
                    table4  = self.fileID.createTable(node4, "BidTable", EnterOrder)
                    node5   = self.fileID.createGroup("/"+nameString, "Ask")          # for EnterAskOrder leaves
                    table5  = self.fileID.createTable(node5, "AskTable", EnterOrder)
                    node6   = self.fileID.createGroup("/"+nameString, "BidDelete")    # for DeleteBidOrder leaves
                    table6  = self.fileID.createTable(node6, "BidDeleteTable", DeleteOrder)
                    node7   = self.fileID.createGroup("/"+nameString, "AskDelete")    # for DeleteAskOrder leaves
                    table7  = self.fileID.createTable(node7, "AskDeleteTable", DeleteOrder)
                    node8   = self.fileID.createGroup("/"+nameString, "BidAmend")     # for AmendBidOrder leaves
                    table8  = self.fileID.createTable(node8, "BidAmendTable", AmendOrder)
                    node9   = self.fileID.createGroup("/"+nameString, "AskAmend")     # for AmendAskOrder leaves
                    table9  = self.fileID.createTable(node9, "AskAmendTable", AmendOrder)
                    node10  = self.fileID.createGroup("/"+nameString, "Message")      # for Message leaves
                    table10 = self.fileID.createTable(node10, "MessageTable", OnTrade)
                except:
                        print "File:", self.fileID.fileName, ": Can't create a sub-node of " , nameString
                        self.printNodeInfo()
            else:
                print nameString, "is already in self.SecurityNodesDict."
        else:
            print self.fileID.fileName, "is not open."    # this might not work

    #def (self,
    def deleteNodes(self, nodeList):
       pass

#--------------end of  HDF5 file class stuff --------------------

""" test.......
H5filename="CM65datafile.h5"
#H5filename="/Users/drw/databases/sixpence/CM65datafile.h5"
h5file=H5file(H5filename)     # instantiate the h5 file object
h5file.wOpen()                # open for writing/appending
h5file.MakeSecuritiesTable()  # for dstoring the list of securities and their summary data.
```

```
print h5file.fileID
h5file.closeHDF5()
"""
# :::::::1:::::::::2:::::::::3:::::::::4:::::::::5:::::::::6:::::::::7:::::::::8:::::::::9:::::::::0:::::::::1:::::::::2
#
#---------- Begin pyTable Leaf classes for pyTable Groups. (Each securityID is a Node in the root ). ----------
#
# ----------------- Trade leaf classes:  -----------------
# an on-market trade order ("TRADE")
class OnTrade(tables.IsDescription):
    day         = tables.UInt8Col()         # unsigned short int for day number
    time        = tables.UInt16Col()        # seconds from midight (=0 and 86400). 10h00 = 36000, 16h00 = 57600
    tradeID     = tables.UInt32Col()        # trade identifier. A unique natural number
    secPrice    = tables.Float32Col()       # price of 1 unit of the security
    secVolume   = tables.UInt16Col()        # (disclosed) volume of the security
    secValue    = tables.Float32Col()       # (disclosed) volume of the security
    flagsMask   = tables.UInt64Col()        # 64 bit mask of the order's flags (see flags.py)
    bidOrderID  = tables.StringCol(16)      # Order ID: A number string prepended by '@'
    askOrderID  = tables.StringCol(16)      # Order ID: A number string prepended by '@'

# an off-market trade order ("OFFTR")   NB no ask broker
class OffTrade(tables.IsDescription):
    day         = tables.UInt8Col()         # unsigned short int for day number
    time        = tables.UInt16Col()        # seconds from midight (=0 and 86400). 10h00 = 36000, 16h00 = 57600
    executeTime = tables.UInt16Col()        # time that trade was effected (as reported by the trading engine)
    tradeID     = tables.UInt32Col()        # order identifier. A unique natural number
    secPrice    = tables.Float32Col()       # price of 1 unit of the security
    secVolume   = tables.UInt16Col()        # (disclosed) volume of the security
    secValue    = tables.Float32Col()       # (disclosed) value of the security
    flagsMask   = tables.UInt64Col()        # 64 bit mask of the order's flags (see flags.py)
    bidBrokerRefs = tables.StringCol(16)    # Bid Broker Trader Refs: A nr string prepended by '@#'
    askBrokerRefs = tables.StringCol(16)    # Ask Broker Trader Refs: 2 nr strings prepended by a '@#' and a '&#'

# a cancel-trade order ("CANTR") NB need to check whether these orders are on trades already in trade register
class CancelTrade(tables.IsDescription):
    day         = tables.UInt8Col()        # unsigned short int for day number
    time        = tables.UInt16Col()       # seconds from midight (=0 and 86400). 10h00 = 36000, 16h00 = 57600
    executeTime = tables.UInt16Col()       # time that trade was effected (as reported by the trading engine)
    tradeID     = tables.UInt32Col()       # order identifier. A unique natural number
    secPrice    = tables.Float32Col()      # price of 1 unit of the security
    secVolume   = tables.UInt16Col()       # (disclosed) volume of the security
    secValue    = tables.Float32Col()      # (disclosed) value of the security
    flagsMask   = tables.UInt64Col()       # 64 bit mask of the order's flags (see flags.py)
    bidBrokerRefs = tables.StringCol(16)    # Bid Broker Trader Refs: A nr string prepended by '@#'
    askBrokerRefs = tables.StringCol(16)    # Ask Broker Trader Refs: 2 nr strings prepended by a '@#' and a '&#'
```

```

# ----------------- Order Book leaf classes: Leaves are used for both Bid and Ask nodes. -----------------

# :::::::1:::::::::2:::::::::3:::::::::4:::::::::5:::::::::6:::::::::7:::::::::8:::::::::9:::::::::0:::::::::1:::::::::2
# an enter-order order ("ENTER")
class EnterOrder(tables.IsDescription):
    day         = tables.UInt8Col()         # unsigned short int for day number
    time        = tables.UInt16Col()        # seconds from midight (=0 and 86400). 10h00 = 36000, 16h00 = 57600
    orderID     = tables.UInt32Col()        # order identifier. A unique natural number
    secPrice    = tables.Float32Col()       # price sought of 1 unit of the security
    secVolume   = tables.UInt16Col()        # (disclosed) volume of the security
    secValue    = tables.Float32Col()       # (disclosed) value of the security
    flagsMask   = tables.UInt64Col()        # 64 bit mask of the order's flags (see flags.py)
    BrokerID    = tables.StringCol(16)       # Broker ID string: A number string prepended by '@'
    TraderID    = tables.StringCol(16)      # Trader ID string. A number string prepended by '&'

# a delete-order order ("DELET")
class DeleteOrder(tables.IsDescription):
    day         = tables.UInt8Col()         # unsigned short int for day number
    time        = tables.UInt16Col()        # seconds from midight (=0 and 86400). 10h00 = 36000, 16h00 = 57600
    orderID     = tables.UInt32Col()        # order identifier. A unique natural number

# an amend order order ("AMEND")
class AmendOrder(tables.IsDescription):
    day         = tables.UInt8Col()         # unsigned short int for day number
    time        = tables.UInt16Col()        # seconds from midight (=0 and 86400). 10h00 = 36000, 16h00 = 57600
#    oldOrderID  = tables.UInt32Col()        # order identifier. A unique natural number
    newOrderID  = tables.UInt32Col()        # old order identifier. A unique natural number
    secPrice    = tables.Float32Col()       # new price sought of 1 unit of the security
    secVolume   = tables.UInt16Col()        # (disclosed) volume of the security
    secValue    = tables.Float32Col()       # (disclosed) value of the security
    flagsMask   = tables.UInt64Col()        # 64 bit mask of the order's flags (see flags.py)
    TraderID    = tables.StringCol(16)       # Trader ID string. A number string prepended by '&'

# Market Engine controls ("CONTL" and "FIELD") The leaf structure for each message in the root Messages node.
class MessageText (tables.IsDescription):
#    day         = tables.UInt8Col()         # unsigned short int for day number
#    time        = tables.UInt16Col()        # seconds from midight (=0 and 86400). 10h00 = 36000, 16h00 = 57600
#    ordertype   = tables.StringCol(8)       # Order type  (either 'CONTL' or 'FIELD')
    number      = tables.UInt8Col()         # A counting number derived from getMessageNumber()
    message     = tables.StringCol(32)      # for FIELD, a group number. For CONTL, a string from the following list:
                    # 'PreOpen system' , 'Suspended system' , 'Opening system' , 'Open system' ,
                    # 'Opening group' , 'ENQUIRE system' , 'ENQUIRE' , 'Adjust-Period system' , 'Suspended group'

# The leaf structure for the Messages node in each security MessageIndex node
```

```
271 class ControlOrder (tables.IsDescription):
272     day         = tables.UInt8Col()         # unsigned short int for day number
273     time        = tables.UInt16Col()        # seconds from midight (=0 and 86400). 10h00 = 36000, 16h00 = 57600
274     messageNr   = tables.UInt8Col()         # A counting number derived from getMessageNumber()
275
276 # _____________________________________ FIN: CMpyTables.py _________________
```

A 4.4 TO CONVERT HFTD FROM FLAT-FILE TO HDF5 FORMAT (SEE §6.8)

THIS PAGE HAS INTENTIONALLY BEEN LEFT BLANK

```
# -*- coding: utf-8 -*-
# :::::::::::::::::::::::::::::::::::::::::: CMdataCleaner:.py ::::::::::::::::::::::::::::::::::::::::::::::::::
# ::::::::1:::::::::2:::::::::3:::::::::4:::::::::5:::::::::6:::::::::7:::::::::8:::::::::9:::::::::0:::::::::1:::::::::2
#:set printoptions=header:0,portrait:n,number:y,left:2pc,right:2pc,top:25mm,bottom:25mm
#for options in vim use :help popt¿option
"""
CMDataCleaner.py cleans the original 65 day files of CMdata, saving the cleaned files to a different directory.
"""
import glob, string

def trunc(x,n=0):
        """ truncates x, to n decimal places ... works for +ve and -ve floats and ints """
        if x> 0:
                return  round( x -0.5 * 10**-n,n)
        else:
                return -round( abs(x) -0.5 * 10**-n,n)

def fdata_to_list(fid):
        """Reads file by line into a list."""
        fid.seek(0,0)
        fdata=fid.readlines()
        return fdata

def printdata(fdata):
        """Prints file data list line by line."""
        count = 0
        for line in fdata:
                print count, line
                count = count + 1

IPdir="/Users/drw/Documents/R_DATA/CMdataOrig/"   # IP dir of sourcefile
OPdir="/Users/drw/Documents/R_DATA/CMdataClean/"  # OP dir of cleaned file

orderTypes={
                'ENTER': 11,
                'DELET': 5,
                'AMEND': 15,
                'TRADE':  10,
                'OFFTR': 11,
                'CANTR': 11,
                'FIELD': 2, #actually needs 3 but cludge here to trigger a fix for field 3
                'CONTL': 6,
                'FINIS': 3
                }
#print orderTypes['FINIS'] , orderTypes.has_key('FINIS') # for testing
```

```

def priceCheck (order, priceField, volField, valueField):
        """ Checks whether priceField * volField = valueField to .... 2 decimal places"""
        try:
                not (round(string.atof(order[priceField][1:]) \
                                * string.atof(order[volField]),2) \
                                == round(string.atof(order[valueField]),2))
        except:
                print "PriceCheckError:", order

global NrErrors ; NrErrors = 0
def fixOrder(order):
        """ corrects known errors in various error types """
        global NrErrors
        NrErrors +=1
        if order[1]=='ENTER':
                priceCheck(order,5,6,7)
        elif order[1]=='DELET': pass
        elif order[1]=='AMEND':
                try:
                        order=order[:6] + order[7:11] + order[14:]      # delete last field
                        #fields are now  <timeStamp>,AMEND,<securityID>,Bid|Ask,<oldOrderID>,
                        #<newOrderID>,<newPrice>,<newDisclosedVolume>,<newDisclosedValue>
                except:
                        print "AMEND conversion error", order
                # X-check that <newPrice>*<newDisclosedVolume>=<newDisclosedValue>
                priceCheck (order,6,7,8)
        elif order[1]=='TRADE':
                try:
                        order=order[:5] + [string.joinfields(order[5:-4],"")[1:]] +order[-4:]
                        #fields are now <timeStamp>,TRADE,<securityID>,<tradeID>,
                        #<Price>,<Volume>,<bidOrderID>,<askOrderID>
                except:
                        print "TRADE conversion error", order
                # X-check that <newPrice> * <newDisclosedVolume> = <newDisclosedValue>
                priceCheck (order,4,5,6)
        elif order[1]=='OFFTR':
                priceCheck (order,5,6,7)
        elif order[1]=='CANTR':
                priceCheck (order,5,6,7)
        elif order[1]=='FIELD':
                try:                                        #NB assumes the GN= a single digit numeral
                        temp=string.splitfields(string.strip(order[2]),' ')
                        order=order[:2]+[temp[0]]+[temp[-1][-1]]
                except:
```

```
                print "FIELD conversion error:", order
    elif order[1]=='CONTL':
            try:                                #turn last field into 3 fields
                temp=string.split(order[-1])
                order=order[:-1]+[temp[0]]  + [string.join(temp[1:])]
            except:
                print "CTRL conversion error:", order
    elif order[1]=='FINIS':
            try:
                order=order[:2]                 # just timeStamp and 'FINIS'
            except:
                print "FINIS conversion error:", order
    else: print " can't fix order of type", order[1]
    return order

def fixNewline(order):
    """ ensures that the last item in the order list is a \n and that it is the only one """
    if order[-1] != '\n':   # if id the last item in the order list is not already only a \n
            try:
                if order[-1][-1]=='\n':          #if the last char in the last item is a \n
                        order[-1]=order[-1][:-1] # drop the \n
                order += ['\n']                  # add a \n to the list
            except:
                "fixNewline error", order
    return order

def cleanFiles():
    """
    Opens all the files in the IPdir and writes clean files of the same name into the OPdir.
    Reads & writes files line-at-a-time so can handle large files.
    """
    filelist=glob.glob(IPdir+"*.csv")
#   print filelist
    print
    print "!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"
    filecount=0
    for filename in filelist:
            try:
                IPfileID=open(filename,'r')
                shortFname=filename[string.rfind(filename,"/")+1:] # just the fname not the dir
                print "Processing:", shortFname
            except:
                "Can't open", filename
                break
            OPfileID=open(OPdir+shortFname,'w',0)
```

```
            orderCount = errorCount = 0
            while 1:
                order=IPfileID.readline()
                if not order:
                        break                       # stop and EOF
                order = string.splitfields(order,',')   # orders are 1 per line, of diff types
                if orderTypes.has_key(order[1]):
                        order=fixOrder(order)
                        order=fixNewline(order)
                        OPfileID.write(string.joinfields(order,',')) # order to OPfile as a.csv
                        orderCount += 1
                else:
                        print " orderType not recognised:", order
                        errorCount +=1
            print " Number of orders processed: ", orderCount,
            print " ...... Number of errors:", errorCount
            print "-------"
            IPfileID.close()
            OPfileID.close()
            filecount = filecount + 1
    print "Number of files processed: " , filecount
    print "FIN"

cleanFiles()    # Just do it!
```

A 4.5 TO EXTRACT HFTD TRADES DATA FOR ANALYSIS & SYNTHESIS (SEE §6.8.2.1)

THIS PAGE HAS INTENTIONALLY BEEN LEFT BLANK

```python
# -*- coding: utf-8 -*-
# :::::::::::::::::::::::::::::::::::::::::::::::: Start of CM_IDTradeAnalysis.py :::::::::::::::::::::::::::::::::::::::::::::::
# :::::::1:::::::::2:::::::::3:::::::::4:::::::::5:::::::::6:::::::::7:::::::::8:::::::::9:::::::::0:::::::::1:::::::::2
# :set printoptions=header:0,portrait:n,number:y,left:2pc,right:2pc,top:25mm,bottom:25mm  #for options in vim use :help popt-o
ption
"""
=======> CMdataAnalysis.py <=======
This code analyses the (cleaned) data in the 65 CMdatafiles, extracting various statistics.
NB ID = Itraday     ED = Extraday , except in the case of secID which is a security indentfier.
Main functions are analyseTrades(),  printAnalysisFile()
last update 20070811
"""
# ::::::::::::::::::::::::::::::::::::::::::::::::::::::::: begin code here  :::::::::::::::::::::::::::::::::::::::::::::::
# :::::::1:::::::::2:::::::::3:::::::::4:::::::::5:::::::::6:::::::::7:::::::::8:::::::::9:::::::::0:::::::::1:::::::::2
from drwGeneric import *
import datetime, glob, string

def datetime2list(dttm):
        """
        Converts an object in the date, time or datetime format to a list of ints.
        Time format is hh:mm:ss.microsecs ; date format is yyyy-mm-dd
        Datetime is yyyy-mm-dd hh:mm:ss.microsecs
        By a quirk of fate, this function can also be used to convert a timestamp string "hh:mm:ss[.microsec]"
        into a list of ints which can then, by indirection, be used to create a datetime instance.
        Eg. t= datetime.datetime(*datetime2List("23:45:33.16789"))
        """
        dttm=str(dttm)
        for sep in ['-','.',' ']:
                dttm=dttm.replace(sep,':')      # replace all ws and other seps with ':'
        return map(int,dttm.split(':'))         # convert the split chars to ints b4 returning


# ::::::::::::::::::::::::::::::::::::::::::::::: analyse ID Trade orders ::::::::::::::::::::::::::::::::::::::::::::::::
# :::::::1:::::::::2:::::::::3:::::::::4:::::::::5:::::::::6:::::::::7:::::::::8:::::::::9:::::::::0:::::::::1:::::::::2
#IPdir="/Users/drw/Documents/R_DATA/CMdataClean/"
#OPdir="/Users/drw/Documents/R_DATA/CMdataAnalysis/"
IPdir="/Users/drw/Documents/R_DATA/CMdataCleaned/dates/"
OPdir="/Users/drw/Documents/R_DATA/CMdataAnalysis/"
oneD=datetime.timedelta(1)
oneH=datetime.timedelta(0,60*60)
oneM=datetime.timedelta(0,60)
oneS=datetime.timedelta(0,1)
securitiesDict={}       # for holding a daily data summary for each security. This data is written to analysis files
                        # key is securityIDs into most of <TRADE> data
densityProfile = {}     # a dictionary of timestamp keys into a list of [secID,tradeID]s
```

```python
#DstatsList = []        # for holding daily stats, mostly for ascertaining ID synth and timing variable quotents
#TstatsList = []        # for holding total stats, mostly for ascertaining ED synth and timing variable quotents

def analyseTrades():
        """
        Opens all the files in the IPdir and compiles general statistics on the ID trade data of the Securities.
        Focus is on     trade information not the order-book/market depth data.
        Reads & writes files line-at-a-time so can handle large files.
        Fills two distionaries: securitiesDict and densityProfile
        """
        global securitiesDict
        filelist=glob.glob(IPdir+"*.csv")
#       print filelist
#       filelist=filelist[:1]           # determine how many, for testing purposes
        print
        print "!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"
        print "%i files to process ..." % len(filelist)

        filecount=0
        for filename in filelist:
                print filename
                DtotNrTrades      = 0           # Day total number of trades
                DminTradePrice  = 0             # Day minimum trade price
                DmaxTradePrice = 0              # Day maximum trade price
                DminTradeVal    = [0,0]         # [Day minimum trade value, timeStamp]
                DmaxTradeVal   = [0,0]          # [Day maximum trade value, timeStamp]
#               tempData =[]                    # test
                try:
                        IPfileID=open(filename,'r')
                        shortFname=filename[string.rfind(filename,"/")+1:] # just the filename not the directory
                        print "Processing:", shortFname
                        securitiesDict={}
                except:
                        "Can't open", filename
                        break
                while 1:
                        order=IPfileID.readline()
                        if not order:
                                break                           # stop and EOF
                        order = string.splitfields(order,',')           # orders are 1 per line, of different types
                        if order [1] == 'TRADE':
                                #               0       1       2       3           4      5       6        ....
                                #fields are<timeStamp>,TRADE,<ticker>,<tradeID>,<price>,<volume>,<value>  <...>
                                # convert price and volume strings to numerics
# :::::::1:::::::::2:::::::::3:::::::::4:::::::::5:::::::::6:::::::::7:::::::::8:::::::::9:::::::::0:::::::::1:::::::::2
```

```
                                order[4] = round(float(order[4][1:]),3) # 3 d.p. round of price
                                order[5] = long(order[5])               # volume as long int
                                order[6] = round(float(order[6]),3)     # 3 d.p.round.Unlike price it doesn't have a '$'
                                tradeTime=datetime.time(*datetime2list(order[0])) # convert 'hh:mm:ss' to a datetime val
                                                                        # not an error! see comment in datetime2list()
                                if securitiesDict.has_key(order[2]):    # is already in the list, so update info
                                        tempData =  securitiesDict[order[2]] # get existing record from dict
                                        tempData[1]=tradeTime           # the last trade timestamp, in datetime format
                                        if  order[4] > tempData[3]:     # if new H for this SecID,
                                                tempData[3] =  order[4] # update H for this SecID
                                        elif order[4] < tempData[4]:    # if new low for this SecID,
                                                tempData[4] = order[4]  # update L 4 this SecID NBcan't be newH & newL!
                                        tempData[5]=order[4]            # update close/last price for this SecID
                                        tempData[6]+=order[5]           # increment vol for this SecID
                                        tempData[7]+=1                  # increment Nr of Trades for this SecID
                                        try:
                                            tempData[8]+= order[6]      # increment cumVol for this SecID
                                        except:
                                            print "tempData: ",  tempData
                                            print "Order    : " , order
                                            print float(tempData[8]) + order [6]
                                else:
                                        # add the security and trade info to dictionary
                                        #       0       1     2     3     4     5     6        7          8
                                        #firstTrade lastTrade   O     H     L     C     cumVol  nrTrades    value
                                        tempData= [tradeTime]*2+[float(order[4]) ]*4+[order[5],1,float(order[6])]
#                                       print tempData
#                               print tempData
                                securitiesDict[order[2]]=tempData
                                DtotNrTrades += 1
                                updateDensityProfile(order)
                IPfileID.close()
                writeIDanalysis(shortFname) # write the analysis data to file

                # now collate some _daily_ stats:
                # fname, DtotNrTrades, DminTradePrice, DmaxTradePrice, [DminTradeVal,when], [DmaxTradeVal,when]
                print " Number of securites:", len(securitiesDict)
                print " Number of intraday trades: ", DtotNrTrades
                filecount += 1
        print " ---------------------------------\nNumber of files processed:", filecount
        print "FIN"

# :::::::1:::::::::2:::::::::3:::::::::4:::::::::5:::::::::6:::::::::7:::::::::8:::::::::9:::::::::0:::::::::1:::::::::2
def writeIDanalysis(shortFname):
        OPfileID=open(OPdir+"IDTradeAnal"+shortFname,'w',0)
```

```
        OPfileID.writelines("<secID>,<Otrade>,<Ctrade>,<OpenP>,<HighP>,< LowP>,<ClosP>,<vol>,<NrTds>,<·value>\n")
        for ticker in securitiesDict:
                x= str(securitiesDict[ticker][-1]) # fix value to ensure it has 2 decimal places (boring!)
                securitiesDict[ticker][-1] = x + ((string.rfind(x,'.')+2) - (len(str(x))-1))*"0"
                OPfileID.writelines(ticker+","+list2string(securitiesDict[ticker],',')+'\n')
        OPfileID.close()

def  updateDensityProfile(order):
        """
        Adds [ticker,trade id] ( i.e. order[2:4] ) to the densityProfile dict at tradetime ( order[0] ).
        Creates a new entry if none exists.
        NB tradetime/timestamp is a string in the form "hh:mm:ss", not a datetime value.
        Because densityProfile is a dict,  a print will not necessarily be in time-order.
        See printTimeProfile()
        Adds [bidorderID,askorderID] for further analysis later
        """
        if densityProfile.has_key(order[0]):
                densityProfile[order[0]] += [order[2:4]+order[-3:-1]]
        else:                   # add  [secID, tradeID, bidOrderID, askOrderID] to densityProfile dictionary
                densityProfile[order[0]]   = [order[2:4]+order[-3:-1]]

def  printDensityProfile(fname=0):
        """
         NB timestamp is a string in the form "hh:mm:ss", not a datetime value.
        Because densityProfile is a dict,  the print is not in time-order
        """
        count = 0
        maxdensity=0
        if fname==0:
                for timestamp in densityProfile:
                        den=len (densityProfile[timestamp])
                        if den>maxdensity: maxdensity = den
        #               print count, ":",timestamp,"density=",den, densityProfile[timestamp]
                        print count,  ","   ,  den,   "," ,  timestamp
                        count +=1
        else:
                try:
                        fname = OPdir + fname
                        fid=open(fname, 'w',0)
                        for timestamp in densityProfile:
                                den=len (densityProfile[timestamp])
                                value = 0
                                for trade in densityProfile[timestamp]:
                                        print trade
                                        value += int([trade][1])
```

```
                                temp = timestamp,  ',', count, ',', den, ',', value, ',', '\n'
                                print temp
                                fid.writelines(temp)
                        count += 1
                except: "Can't write to ", fname
                fid.close()
        print "max density:", maxdensity

# :::::::1:::::::::2:::::::::3:::::::::4:::::::::5:::::::::6:::::::::7:::::::::8:::::::::9:::::::::0:::::::::1:::::::::2
def printAnalysisFile(shortFname, colWidth=0):
        """
        Pretty-prints an analysis file generated by analyseTrades()
        """
        fid = open(OPdir+"IDTradeAnal"+shortFname+".csv",'r')
        x= fdata_to_list(fid)
        printdata( fdata_to_list(fid),colWidth) # print items Radjusted in a colWidth chars field

# now we need to create a segence of trading times
def totalValue(densityList):
        """
        Returns the total value of all trades at a timestamp.
        densityList is a timestamp entry in the densityProfile dict,
        ie. a list of ['secID', 'value'] lists.
        'value' is a char form of a Value int.
        """
        value=long(0)                           # not necessary to cast, but good to make the point
        for item in densityList:
                value += long(item[1])
        return value

def chartimeToSecs(char):
        """
        converts a time in the form "hh:mm:ss" to seconds
        """
        res = datetime2list(char)
        return 3600 * res[0] + 60 * res[1] + res[2]
#print  chartimeToSecs("16:01:05")

def totalTimestampValue(adict):                 # , eventRate, sampleRate):
        """
        Generates a time-ordered list of total Value for each timestamp in adict in the form [secTime,totalVal]
        adjusts timeoffst to 0 - need to rework to make it either offset or fixed
        adict is densityProfile{}
        """
        timeStamps = adict.keys()
        timeStamps.sort()                       # cronological order. It works even though they are chars!
        timeOffset=chartimeToSecs(timeStamps[0]) # get the first timeStamp to use as offset
        synthList = []                                  # eventually, a list of [timeOffset, ValToBeMapped]
        for timeStamp in timeStamps:
#               print chartimeToSecs(key), timeOffset   # add all values at timeStamp together
                synthList += [[chartimeToSecs(timeStamp)-timeOffset, totalValue(adict[timeStamp])]]
        return synthList

def valueToExpFreq(minFreq, value, valueRange, radix, nrFurcations):
        """ converts a Value to an exonential of 1/pitchFrequency """
        exponent=nrFurcations*(1-(float(value)/valueRange))
        return minFreq * pow(radix,exponent)
#print valueToFreq(100, 6000, 6000, 2, 4)

# :::::::1:::::::::2:::::::::3:::::::::4:::::::::5:::::::::6:::::::::7:::::::::8:::::::::9:::::::::0:::::::::1:::::::::2
def sonifyValues(adict):
#       adict is densityProfile{}
        valuesNM=totalTimestampValue(adict) # calculate density profile of entire dictionary
        mnmx= colMinMax(valuesNM, 1)    # find min and max totalVal for all trades at any timestamp in the dict
        range = mnmx[1]-mnmx[0]         # range of totalValues in all timestamps in adict # print minV, maxV
        for item in valuesNM:
#               exponent=furcations*(1-(float(item[1])/range))
#               item[1] = minF * pow(radix,exponent)            # item=[timestamp, value]
                item[1]=valueToExpFreq(80,item[1],range,2,4)
        # This is a simple example of how a sine synth is done.
        ENV01=((0,0),(50,1),(99,0))   # tuple of (x,y)=(%duration,amplitude) pairs
        env01= makeEnvArray(ENV01)  # an envelope array
        #print env01
#       SR=44100        # sample rate. for CD writes, set SR=44100 and setnchannels(2)
        OPdirectory="/Volumes/DRWSONUSB2/R_CMdata/CMaudio/"
#       OPdirectory="/Users/ceuser/audio/"
        OPfilename="testCM.aiff"
        fname=OPdirectory+OPfilename
        OPfileID = aifc.open(fname, 'w')
        OPfileID.setnchannels(1)      # the number of channels (1=mono)
        OPfileID.setsampwidth(2)      # the sample width in bytes (2=16bit format)
        OPfileID.setframerate(SR)     # the frame rate. For mono, FR =SR
        for item in valuesNM[:200]:
#               print item
                sinAIF(item[1], 0.125, env01, OPfileID) # an S(1) fn
        OPfileID.close()
# :::::::1:::::::::2:::::::::3:::::::::4:::::::::5:::::::::6:::::::::7:::::::::8:::::::::9:::::::::0:::::::::1:::::::::2

def cSoundScore(adict,fund,radix,furcats, OPfilename):
        """ produces a Csound score for instr 1 (see EO this F for declaration)"""
```

```
        timeStamps = adict.keys()               # get the keys (they're in string form)
        timeStamps.sort()                       # crono order. This works even though they are in string form
        timeOffset=chartimeToSecs(timeStamps[0]) # the 0th item will be earliest; value is in secs after midnight
        valuesNM=totalTimestampValue(adict)     # NM = density profile of adict as a list of [timestamp, value]'s
        mnmx= colMinMax(valuesNM, 1)    # find min and max totalVal for all trades at all timestamp in adict
        range = mnmx[1]-mnmx[0]         # range of totalValues in all timestamps in adict # print minV, maxV
#       Fmin=100
#       Fmax=5100
#       cLin= (Fmax-Fmin)/float(range)          # constant for linear mapping to frequency

#           __________open .sco file for csound score_____________
        OPdirectory="/Volumes/DRWSONUSB2/R_CMdata/CMcsound/"
#       OPdirectory="/Users/ceuser/audio/"
        fname=OPdirectory+OPfilename
        OPfileID = open(fname, 'w')
        dur = 1.0
        amp =1.0
        tempoTable='t   0       3600    ; 60 x time compression: 1 sec = 1 minute'
        waveform1='f    1       0       16384   10      1         ; sinw'
        waveform2='f    2       0       16384   9.5     1       0 ; 1/2 sine for envelope'
        OPfileID.write(tempoTable +'\n')
        OPfileID.write(waveform1 +'\n')
        OPfileID.write(waveform2 +'\n')
# ------------begin: for sonifying individual trades ------------------------------------------------------------
#       for stamp in timeStamps:
#               stampSec=chartimeToSecs(stamp)# print stamp, adict[stamp]
#               for trade in adict[stamp]:#     print stampSec-timeOffset, cLin/float(trade[1]) #
#                       OPfileID.write('i\t1\t' +str(stampSec-timeOffset) + '\t' + str(dur) \
#                                       + '\t' + str(amp)  + '\t' \
#                                       + str(valueToExpFreq(fund,trade[1],range,radix,furcats)) + '\n')
# ------------end:  for sonifying individual trades ------------------------------------------------------------
# ------------begin: for sonifying value of all trades at  timestamp ----------------------------------------
        for item in valuesNM:
                freq = valueToExpFreq(fund,item[1],range,radix,furcats) # item=[timestamp, value]
#               print freq
                OPfileID.write('i\t1\t' +str(item[0])+'\t'+ str(dur)+'\t'+str(amp)+'\t'+str(freq) + '\n')
# ------------end: for sonifying value of all trades at  timestamp ----------------------------------------
        OPfileID.write('e\n')
        OPfileID.close()
        print "FIN: wrote cSound score", OPfilename
# :::::::1:::::::::2:::::::::3:::::::::4:::::::::5:::::::::6:::::::::7:::::::::8:::::::::9:::::::::0:::::::::1:::::::::2

# ::::::::::::::::::::::  Major executables here ::::::::::::::::::::::::::::::::::::::::::::::::::::::
print "::::::::::::::::::::::::::"
SR=44100
```

```
analyseTrades()                         # main from raw daily files into securitiesDict{} & densityProfile{}

#printAnalysisFile("001.csv",11)        # or use, say, printAnalysisFile("001.csv")
#printDensityProfile("001denPlot.csv")
#printDensityProfile()
#print dictToSynthList(densityProfile)
#
#sonifyValues(densityProfile)           # use this for single total value at timestamp son.
#cSoundScore(densityProfile, 2,2,12, "001_020212.sco")
#
#cSoundScore(densityProfile, 64,2,7, "001_totVal.sco")
#secIDs=securitiesDict.keys()
#secIDs.sort()
#for id in secIDs:
#       print securitiesDict[id]
courseOfSalesDict = {}     # a dictionary of timestamp keys into a list of [secID,tradeID]s

"""
def courseOfSales():
        # Generates a Course of Sales dictionary for data in densityProfile{}
        timestamps=densityProfile.keys()
        timestamps.sort()
        for stamp in timestamps:
                # register the first trade price
                # check whether each trade is in the [price, volume] dictionary
                for each <TRADE> :
                if first trade of <securityUD>
                        Record price, nrUnits
                else (<TRADE> has already been recorded):
                        if price is same as previous, add nrUnits
                        else price is different, so sonify previous (value + gliss:
up if price rose
down if price dropped

"""
"""
working code to sort out the multiple trade at same bid/askIDs
timestamp=densityProfile.keys()
timestamp.sort()
for stamp in timestamp:
        bidIDlist=askIDlist=[]
#       if len(densityProfile[stamp])>1:  # there is more than one trade at that time
#               # NB no need to check for same securityID because different securityID's
#               # will automatically have different bidID & askID
        for order in densityProfile[stamp]:
```

```
#               print order[2], order[3]
                bidIDlist+=[order[2]]
                askIDlist+=[order[3]]
        print "bids:", bidIDlist
        print "::::::::::"
        print askIDlist
        print "%%%%%%%%%%"

        for bid in bidIDlist:
                if bidIDlist.count(bid)>1:
                        print "multiple trades with same bidID", bid
                else:
                        print "-----------------------only 1 trade for  bidID", bid

for key in densityProfile.keys():
        print key , densityProfile[key]
"""
# :::::::::::::::::::::::::::::::::::::::::::: End of CM_IDTradeAnalysis.py ::::::::::::::::::::::::::::::::::::::::
# :::::::1:::::::::2:::::::::3:::::::::4:::::::::5:::::::::6:::::::::7:::::::::8:::::::::9:::::::::0:::::::::1:::::::::2
"""
; csound instrument for playing score produced by cSoundScore(adict)
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 1
        ifreq = 2048
        p5=p5/4
        kmaxamp = 120 - (110*p5/ifreq)  ; p5 is frequency
        p3 = (p3*0.9) + (0.1*ifreq/p5)                  ; p3 is dur
        idur = 1/p3                             ; oscil freq = 1/dur
        kenv    linseg 0, 0.01,1, P3-.01, 0
;       kenv    oscil 1, idur, 2                ; use f2 (1st 1/2 sine
        asig    oscili kenv, p5, 1      ; use f1 (sin)
        out asig * kmaxamp
endin

; SIMPLE csound instrument for playing score produced by cSoundScore(adict)
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 1
```

```
        kmaxamp = 100 - (100*p3/4096)  ; 4096 is max freq for 12 octaves on fund of 1 Hz
        ifreq = p5 ; * 440
        kenv    oscil 1, 1/p3, 2        ; use f2 (1/2 sine)
        asig    oscili kenv, ifreq, 1 ; use f1 (sin)
        out asig * kmaxamp
endin
;
; csound instrument for playing score produced by cSoundScore(adict)
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

instr 1
        kmaxamp = 100
        ifreq = p5 ; * 440
        iamp = 1
        iamp1= 0.68
        p3=p3*1.2

        idur1 = p3 * 0.38       ;2/5 dur
        idur2 = p3 * 0.5        ;1/2 dur
        idur3 = 1-idur1 ;3/5 dur

        kenv    linseg 0, p3*.25, 1, p3*.75, 0
        asig    oscili kenv, ifreq, 1 ; use f1 (sin)
        outs1 asig * kmaxamp
        outs2 asig * (1-kmaxamp)
endin
==========
"""
```

```
# -*- coding: utf-8 -*-
# :::::::::::::::::::::::::::::::::::::::::: CM65summaries.py ::::::::::::::::::::::::::::::::::::::::::::::::::::
# ::::::::1:::::::::2:::::::::3:::::::::4:::::::::5:::::::::6:::::::::7:::::::::8:::::::::9:::::::::0:::::::::1:::::::::2
# :set printoptions=header:0,portrait:n,number:y,left:2pc,right:2pc,top:25mm,bottom:25mm
#for options in vim use :help popt-option

"""
Summaries of the 65 dayfiles are stored in a pickle DB for faster referencing of HDF5 file
"""
import glob, string, cPickle

class MarketDay:
    """ Reads summary data for specified day from .csv file. Optional auto print of summary."""
    def __init__(self, day=0, fprint=False):
        self.day            = day
        self.IPdir          = "/Users/drw/Documents/R_DATA/CMdataAnalysis/"
        self.OPdir          = "/Users/drw/Documents/R_DATA/CMdataAnalysis/"
        self.dayFID         = None
        self.fileList       = glob.glob(self.IPdir+"*.csv")
        self.dayFileName    = None
#        self.dayList        = []+ daylist # this permits a single number 'n' in input rather than '[n]'
        self.dayData        = None  #self.fileToList([]
        self.closeValues    = []    # a list of (key, value) tuples for a day. See self.getValues()
        try:
            if self.day > 0:        # do it automatically
                self.dayToFilename()
                self.fileToList()
                if fprint:
                    self.printSummary()
        except:
            print "EXCEPTION:: No summary data available for day", self.day, "in", self.IPdir

    def stringToNumeric(self, val):
        """ Converts a string to a numeric: an integer if point is present, else a real
        If string cannot be converted to a numberic, input is returned unchanged.
        CHECK : what happens for longs? """
        try:
            res=string.atoi(val)
        except:
            try:
                res=string.atof(val)
            except:
                res=val
        return res

# :::::::1:::::::::2:::::::::3:::::::::4:::::::::5:::::::::6:::::::::7:::::::::8:::::::::9:::::::::0:::::::::1:::::::::2
    def dayToFilename(self):
        dayString="IDTradeAnal"+("00"+str(self.day))[-3:]+".csv" # a 3 char numeric string in the form "001" to "065"
        self.dayFileName=self.IPdir+dayString

    def fileToList(self):
        """ Converts a numeric string to a number. Returns contents of file fname in a list """
        self.dayFID=open(self.dayFileName,'r')     # file = IPdirectory+IPfilename ie full path
        self.dayFID.seek(0,0)
        self.dayData = self.dayFID.readlines()  # Reads file into a list of lines
        self.dayFID.close()

    def stringToList(self, stringy):
        """ Converts a string to a list of numerics if possible, else strings."""
        res=[]
        tmp=string.split (stringy,",")
        for i in range (len(tmp)):
            tmp[i] = self.stringToNumeric(tmp[i]) # convert all numberic string to numerics
        res += tmp
        return res

    def printSummary(self, fieldsize=0):
        """ Prints each line of a file in a list R adjusted in a field fieldsize chars wide.
        fdata is a list of strings, each representing a line. NB list of strings, not ints etc.
        If fieldsize=0, the data is scanned to determine the maximum fieldsize necessary or each field
        and that fieldsze is used to  R adjust each chars wide column. """
        count = 0
        if fieldsize > 0:
            for line in self.dayData:
                print  ' '* (fieldsize-len(str(count))),  count,":",
                for y in string.split(line,','):
                    print  ' '* (fieldsize-len(y)), y,  # "\t",
                count += 1
        else:   # fieldsize=0 so calculate them variably
            fieldsizeList={}
            for line in self.dayData:                   # scan the lines
                lineaslist=string.split(line,',')
                for col in range(0,len(lineaslist)):    # find and store the max fieldsize
                    try:
                        if fieldsizeList[col] < len(lineaslist[col]):
                            fieldsizeList[col] =  len(lineaslist[col])
                    except:     # gets here when a column's fieldsize hasn't been stored b4
                        fieldsizeList[col] =  len(lineaslist[col])
            # make up the format string
            fmt = ""
```

```
            for col in fieldsizeList: fmt += ' %' + str(fieldsizeList[col]) + 's'
            # now do the print
            pad = len(str(len(self.dayData)))           # the char size of the max line len
            for line in self.dayData:
                print  ' '* (pad-len(str(count))),  count,":",
                print fmt  % tuple(string.split(line,',')),
                count += 1

#d1=DaySummary(1 fprint=True)

class MarketSummary(MarketDay):
    def __init__(self, dazeList=[]):
        MarketDay.__init__(self)
        self.dayList =[]
        if dazeList:
            self.dayList = dazeList
#        self.day = self.dayList[0]
        self.securitiesDict     = {}    # key is securityID, data is list of day summaries
        self.summaryDict        = {}    # summaries of the trading data of a securities in self.securitiesDict
        if not self.dayList:
            self.dayList = [ i for i in xrange(1,66)]   # a bit crude. better to get them from the filelist
            print "Making a dictionary of all securities in all files....."
        self.doDays()
        self.summariseSecurityData()

    def getDayData(self, dayNr):
        # Gets one day's data from file and adds it to self.securitiesDict as a list.
        # Each security is thus a list of lists.
        try:
            self.day = dayNr
            if self.day > 0:                                    # do it automatically
                self.dayToFilename()
                self.fileToList()                               # data is now in self.dayData
            else:
                print "EXCEPTION:: No summary data available for day", self.day, "in", self.IPdir
                return
            newDayData = []
            for line in self.dayData:
                newline=self.stringToList(line)
                newDayData.append([newline[0]]+newline[3:])
            del newDayData[0]                                   # toss the headings away
            for sec in newDayData:
                if sec[0] in self.securitiesDict.keys():
                    self.securitiesDict[sec[0]].append(sec[1:])
                else:
```

```
                    self.securitiesDict[sec[0]] = [sec[1:]]
#               if sec[0] == '-':           # NB This was a test to discover that on day65 one secID ='-'
#                   print "Day:", self.day, ":", line  # drw changed it to (unique) '99999' (in the analysis file only)
        except:
            print "EXCEPTION:: Can't add", dayNr, "to the dictionary."

# :::::::1:::::::::2:::::::::3:::::::::4:::::::::5:::::::::6:::::::::7:::::::::8:::::::::9:::::::::0:::::::::1:::::::::2
    def doDays(self):
        for day in self.dayList:
            self.getDayData(day)

    def summariseSecurityData(self):
        # Summarises the day data for each security trading in self.dayList
        print "Summarising securitiesDict data...."
        if not self.securitiesDict:     # if the securitiesDict has not been generated
            for d in self.dayList:      # generate it
                self.getDayData(d)      # adds a list for each day to the self.securitiesDict
        secs = self.securitiesDict.keys()
#       # Make self.summaryDict                                 0  1  2  3   4      5       6       7
#                                                               O, H, L, C, Vol, #Trades, $Value, NrDays
        for sec in self.securitiesDict.keys():
            secSummary = [0]*8
            for d in xrange(len(self.securitiesDict[sec])):
                secSummary[7]+=1                                # increment Nr of days summarised
                secSummary[3] = self.securitiesDict[sec][d][3]  # new Close price
                secSummary[4] += self.securitiesDict[sec][d][4] # increment trading Volume
                secSummary[5] += self.securitiesDict[sec][d][5] # increment #trades
                secSummary[6] += self.securitiesDict[sec][d][6] # increment $Value
                if secSummary[0] == 0:                          # if this is the first day
                    secSummary[0] = self.securitiesDict[sec][d][0]  # Open price
                    secSummary[2] = self.securitiesDict[sec][d][0]  # initial Low price
                if self.securitiesDict[sec][d][1] > secSummary[1]:
                    secSummary[1] = self.securitiesDict[sec][d][1]  # new High price
                if self.securitiesDict[sec][d][2] < secSummary[2]:
                    secSummary[2] = self.securitiesDict[sec][d][2]  # new Low price
            self.summaryDict[sec]=secSummary                    # add it to the summary dictionary

    def sortByValue(self, dict):
        val = [(dict.itervalues(), values[6]) for values in dict.itervalues()]
        return val

    def dictColSort(self, dictionary, col=0, descend=False ):
        """ returns a list of (value, key) tuples of sorted col column of dictionary values, ascending by default
        # example:  # s          = MarketSummary()
                    # valHL      = s.dictColSort(s.summaryDict, 6, descend=True)
```

```
                    # top100keys = valHL[:100][1]    """
        res=[( v[col], k) for k, v in dictionary.iteritems()]   # NB (value, key) not (key, value)
        res.sort(reverse=descend)
        return res

# :::::::1:::::::::2:::::::::3:::::::::4:::::::::5:::::::::6:::::::::7:::::::::8:::::::::9:::::::::0:::::::::1:::::::::2
def doit():
    s = MarketSummary()
    kees=s.summaryDict.keys()
    OPfileID = open(s.OPdir+"summaryDict_all.pickle", 'w')
    cPickle.dump (s.summaryDict, OPfileID)                      # pickles a 65 day summary of all securities
    OPfileID.close()
    keysSortedByValue = s.dictColSort(s.summaryDict, col=6, descend=True)
    keysSortedByPrice = s.dictColSort(s.summaryDict, col=3, descend=True)   # uses Closing Price
    keysSortedByVolume = s.dictColSort(s.summaryDict, col=5, descend=True)
    OPfileID = open(s.OPdir+"summary65sortedLists.pickle", 'w')
    cPickle.dump (keysSortedByValue, OPfileID)  # pickles list of (value, secID) tupes  (sorted by value over 65 days)
    cPickle.dump (keysSortedByPrice, OPfileID)  # pickles list of (price, secID) tupes  (sorted by price over 65 days)
    cPickle.dump (keysSortedByVolume, OPfileID) # pickles list of (volume, secID) tupes (sorted by vol over 65 days)
    OPfileID.close()

def getTopPriceSecs(n=1):
    IPdir       = "/Users/drw/Documents/R_DATA/CMdataAnalysis/"
    filename    = IPdir+"summary65sortedLists.pickle"
    IPfileID = open(filename, 'r')
    # how do we get the 2nd item without reading the first? IPfileID.next() doesn't work.
    listy=cPickle.load(IPfileID)     # how do we get the 2nd pickled item without reading the first?
#    listy=cPickle.load(IPfileID)    # IPfileID.next() doesn't work.
#    listy=cPickle.load(IPfileID)    # IPfileID.next() doesn't work.
    lst = []
    for i in xrange(n):
        lst.append(listy[i][1])     # the 2nd of the tuple is the secID
    IPfileID.close()
    return lst
#------------------EOF - FIN --------- pickled onions 4T!

```

```
# -*- coding: utf-8 -*-
# ::::::::::::::::::::::::::::::::::::::::: getH5security.py ::::::::::::::::::::::::::::::::::::::::::::::::::
# :::::::1:::::::::2:::::::::3:::::::::4:::::::::5:::::::::6:::::::::7:::::::::8:::::::::9:::::::::0:::::::::1:::::::::2
#:set printoptions=header:0,portrait:n,number:y,left:2pc,right:2pc,top:25mm,bottom:25mm
#for options in vim use :help popt¿option

# class definitions for fetching, analysing and plotting of a given securityID from an HDF5 file.
# NB python days are zero-based. h5 days are 1-based. All days are by default python days. h5 days are called 'h5days'
# last edit: drw 20071011

from CMpyTables import *    # HDF5 classes for the CM data
from CMflags import *       # used to compress the <flags> field in order book entries into a single 64bit mask
import glob, string, time, datetime, os, tables, cPickle
import numpy
import pylab                # matplotloib routines for plotting


class h5security (H5file):              # inherits the h5file class (defined in CMpyTables.py)
    def __init__(self, filename):
        self.secID      = ''        # the Id (name) of the security data to be instanced
        self.fileName   = filename  # the name string of the H5 file. It's usual for it to have a .h5 extension.
        self.h5fileError    = "Unable to access HDF5 file "
        self.shortFileName  = self.fileName[string.rfind(self.fileName,"/")+1:] # just filename, not the path
        self.fileID = None          # returned by the opening definitions. Used to access the file.
        self.mode = None            # the mode of the the file opening. 'w' = write, 'a' = append, 'r' = read
                                    # Internal use only. See getFlagNumber().
        self.securitiesDict = {}    # A dict of tuples for each Node/Group in the root directory.
                                    # each tuple is in the form (CMgroupID, firstDay, lastDay,  ....)
                                    # should this Dict be held in RAM and written to the h5 file before close?
        self.h5data      = []       # into which h5 file data is extracted, one list per order
        self.h5dataBuff1 = []       # into which h5 file data can be transferred for temporary or analytic purposes
        self.h5dataBuff2 = []       # into which h5 file data can be transferred for temporary or analytic purposes
        self.dayRange    = (1,66)   # range of days requested/required      - default values are auto assigned
        self.hourRange   = (0,24)   # range of hours requested/required     - ditto
        self.minuteRange = (0,60)   # range of minutes requested/required   - ditto
        self.secondRange = (0,60)   # range of seconds requested/required   - ditto
        self.timePeriod  = (0,0)    # (min,max) of time period required, in seconds from midnight
        self.nrDays      = 0        # set to self.dayRange[1] - self.dayRange[0].
                                    # Indexes of the 1st of 3 dimensions of the self.bins array

        self.dayDuration =  0       # Nr of secs of a day's data to be analysied.
                                    # Set to self.timePeriod[1] - self.timePeriod[0]. Used for squeezing data to bins
        self.nrBins      = 100      # the number of bins into which each day data is to be sorted   - ditto
        self.bins        = None     # a numpy array of data bins
        self.breakpoints = None     # placeholder for a numpy1Darray of self.nrBins break points (in secs)
```

```
                                    # into which orders are binned.
        self.nrVariates  = 1        # the nr of independ. vars to be extracted from h5data or +ed for interum results
#        self.barData     = []      # a list of day lists of summary data for bar/histogram plotting
        self._getAllSecurityNodes() # open the bugger!

# :::::::1:::::::::2:::::::::3:::::::::4:::::::::5:::::::::6:::::::::7Y:::::::::8:::::::::9:::::::::0:::::::::1:::::::::2
    def _getAllSecurityNodes(self):
        """makes a dictionary of all the securities in the h5 file.
        Nominally a private method/function because our focus is on single securties."""
        try:
            self.rOpen()            # open the h5 file for reading.  inherited from h5file class.
        except: print self.h5fileError
        for sec in self.fileID.getNode("/"):
            self.securitiesDict[sec._v_name] = []           # make the string of the secID a key in the securitiesDict
#            print "sub groups:------------------:"
            for subgp in self.fileID.getNode("/"+sec._v_name):
                self.securitiesDict[sec._v_name] += [subgp]     # add to the list of group name strings
        print self.securitiesDict.keys()

    def _securityExistsQ(self):
        res=False
        if self.secID in self.securitiesDict.keys():
            return True
        else:
            print self.secID, "not available. Securitites available are:"
            print self.securitiesDict.keys()
            return False

    def setTimePeriods (self, days = (1,66), hours = (0,24), minutes = (0,60), seconds = (0,60)):
        """Time data is the time period from which data is to be pulled from the h5 file for this security.
        Transfers the time data to class variables; fixing common range error if in evidence."""
        self.timePeriod = ( hours[0]*3600+minutes[0]*60+seconds[0],\
                                    hours[1]*3600+minutes[1]*60+seconds[1]) # (min_sec, max_sec)
        if  days[0] < 1: days[0] = 1                    # just in case. The h5 day is 1-based.
        if days[1] == days[0]: days[1] = days[0]+1      # ditto ASSIGNMENT DON'T WORK - days is a () i.e. immutable
        self.dayRange = days
        if hours[1] == hours [0]: hours = (hours[0],hours[0]+1) ; self.hourRange = hours
        if minutes[1] == minutes [0]: minutes = (minutes[0],minutes[0]+1) ; self.minuteRange = minutes
        if seconds[1] == seconds [0]: seconds = (seconds[0], seconds[0]+1) ; self.secondRange = seconds
        self.nrDays = self.dayRange[1] - self.dayRange[0]           # for indexing 1st of 3 Ds of the array
        self.dayDuration =  self.timePeriod[1] - self.timePeriod[0] # nr of secs of a day's data to be analysed.

    def getOnTradeData (self, secID):
        """Pulls the relevant leaf data from the h5file secID branch & stores it in self.bins according to time data.
        Assumes self.bins has been appropriately made to accomodate the data. See self.makeBins()
```

```
        If multiple days, assumes all days are for the same time period."""
        self.secID = secID
        if self._securityExistsQ():                         # fetch requested trade data from h5 file
            nodey = "/" + self.secID + "/OnTrade/OnTradeTable"
            print "getOnTrades: ", nodey
            tab = self.fileID.getNode(nodey)
            print "getOnTrades: tradeTable columns:" , tab.colnames
            print "getOnTrades: MaxNrrows available:",tab.nrows,"\tdays:",self.dayRange ,"\ttime range:",self.timePeriod

            for row in tab.iterrows():
                if int(row['day']) in range(*self.dayRange):
                    if int(row['time']) in range (*self.timePeriod):                        # PULL THE h5 DATA
                        rowsie = [ row['day'], row['time'], row['secPrice'], row['secVolume'] ]
#                        print rowsie
                        self.h5data += [rowsie]                                 # NB the 0th item is for self.dayRange[0]
#                    else:
#                        break   # assume we can do this ... because a day's timestamps stored incrementally
#                else:
#                    break   # assume we can do this ... because day numbers are stored incrementally?
        else:
            print "getOnTrades: Can't get OnTrades for", self.secID
            print "getOnTrades: Securities in open file:", self.securitiesDict.keys()

# :::::::1:::::::::2:::::::::3:::::::::4:::::::::5:::::::::6:::::::::7:::::::::8:::::::::9:::::::::0:::::::::1:::::::::2
    def getAskData (self, secID) :
        """ Pulls the relevent (leaf) data from the h5file. A leaf is the data table at the end of a node.
        If multiple days, assumes all days are for the same time period."""
        self.secID = secID
        if self._securityExistsQ():                         # fetch requested trade data from h5 file
            nodey = "/" + self.secID + "/Ask/AskTable"
            print "getAskOrders: ", nodey
            tab = self.fileID.getNode(nodey)
            print "getAskOrders: AskTable columns:" , tab.colnames
            print "getAskOrders: NrRowsOfData:",tab.nrows,"\tdays:",self.dayRange ,"\ttime range:",self.timePeriod
            for row in tab.iterrows():
                if int(row['day']) in range(self.dayRange[0]+1,  self.dayRange[1]+1):       # because day0 == h5day1
                    if int(row['time']) in range (*self.timePeriod):                        # PULL THE h5 DATA
                        self.h5data += [[ row['day'],row['time'],row['secPrice'],row['secVolume'], row['flagsMask'] ]]
        else:
            print "getAskOrders: Can't get Ask Orders for", self.secID
            print "getAskOrders: Securities in open file:", self.securitiesDict.keys()

    def getBidData (self, secID):
        """ Pulls the relevent (leaf) data from the h5file. A leaf is the data table at the end of a node.
        If multiple days, assumes all days are for the same time period."""
```

```
        self.secID = secID
        if self._securityExistsQ():                         # fetch requested trade data from h5 file
            nodey = "/" + self.secID + "/Bid/BidTable"
            print "getBidOrders: ", nodey
            tab = self.fileID.getNode(nodey)
            print "getBidOrders: BidTable columns:" , tab.colnames
            print "getBidOrders: Nr rows of data:",tab.nrows,"\tdays range:",self.dayRange,"\ttime range:",self.timePeriod
            for row in tab.iterrows():
                if int(row['day']) in range(self.dayRange[0]+1,  self.dayRange[1]+1):    # because day0 == h5day1
                    if int(row['time']) in range (*self.timePeriod):                     # PULL THE h5 DATA
                        self.h5data += [[ row['day'], row['time'], row['secPrice'], row['secVolume'], row['flagsMask'] ]]
        else:
            print "getBidOrders: Can't get Bid Orders for", self.secID
            print "getBidOrders: Securities in open file:", self.securitiesDict.keys()

# :::::::1:::::::::2:::::::::3:::::::::4:::::::::5:::::::::6:::::::::7:::::::::8:::::::::9:::::::::0:::::::::1:::::::::2
    def makeBins(self, nrBins=100, nrVariates= 2):
        """Makes an list lists: self.nrDays X nrBins X nrVariates+1.
        self.bins is in the form [day [breakpoint [variate1, variate2, ...]]] """
        self.nrBins     = nrBins        # the number of bins (rows) into which each day data is to be sorted
        self.nrVariates = nrVariates    # the number of variable data (columns) to be analysed, or needed for analysis
        self.breakpoints = numpy.arange(nrBins, dtype=numpy.int16)  # a 1D-array of self.nrBins break points (in secs)
                                                                    # into which orders are binned.
        self.breakpoints = numpy.multiply(self.breakpoints, self.dayDuration/self.nrBins)
                # each elt in self.breakpoints in now a rel start time of the bin NB this arg has to be a list or tuple
        self.breakpoints = numpy.add(self.breakpoints, [self.timePeriod[0]])
                                        # each elt in self.breakpoints in now an absolute start time of the bin
        self.bins = []                  # a list of day lists of summary data for bar/histogram plotting
        for day in range(*self.dayRange):       #  make a list of day lists of breakpoints. The day Nr is not important.
            self.bins.append([nrVariates*[0,] for i in xrange(len(self.breakpoints))])

    def dataToBins (self):
        """Massages the H5 data in self.hData into nrBin time clumps, aggregating values in the variates.
         Transfers the timer data to class variables; fixing common range error if in evidence.
        The type and amount of data is set in getH5securityData()."""
        if len(self.h5data)==0:
            print "dataToBins: self.h5data array is empty."
            print "Fill using self.getOnTradeData() etc before calling self.dataToBins()"
            return 0
        for order in self.h5data:                           # find the right bin Nr based on breakpoint time
            binNr = 0                                       # temp var to hold which bin the order is in
            for i in range (len(self.breakpoints)):         # find the appropriate time bin for this order
                if order[1] >= self.breakpoints[binNr+1]:   # if order time >= bin breakpoint time
                    binNr += 1                              # increment the binNr
            try:
```

```
                            # array index is -vely offset from the min day number
                    dayIndex=order[0]-self.dayRange[0]-1        # min day index is 0 despite which actual day it is.
                    self.bins[dayIndex][binNr][-2] += order[-1] * order[-2]    # ++ the value (price X vol)
                    self.bins[dayIndex][binNr][-1] += 1  # ++ the number of trades in this bin. res index is 1 < dayNr
                except:
                    print "order:", order, "binNr:", binNr, order[0]
                    return 0

# :::::::1:::::::::2:::::::::3:::::::::4:::::::::5:::::::::6:::::::::7:::::::::8:::::::::9:::::::::0:::::::::1:::::::::2
    def _plotLayoutCodes(self, nrCols=5):
        """ returns a list of location codes for nrplots in nrCols columns."""
        nrPlots=len(range(*self.dayRange))
        codes=[]           # a list of tuples, 1 for each plot e.g. (5,2,3) is the 3rd plot in a 5x2 subplot drawing
        nrRows= divmod(nrPlots+nrCols-1, nrCols)[0]   # the old rounding trick!
        stem=(nrRows,nrCols)
        for plotNr in range(nrPlots):
            codes.append(stem+(plotNr+1,))
        return codes

    def getMaxArrayVal(self, array, index):
        """Returns the maximum value of the array."""
        maxie = 0
        for day in array:
            for bin in day:
                if bin[index] > maxie:
                    maxie = bin[index]
        return maxie

    def drawBinBars(self, index=-2):
        """Draws bar charts of value data in self.bins, which is a list of day lists, each of nrBins divisions"""
        codes=self._plotLayoutCodes(5)     # in weekly (5-day) columns
#        print "plot layout codes:", codes
        maxVal = self.getMaxArrayVal(self.bins, 0)
        print "drawValueBars: max array value = ", maxVal
        for day in range(*self.dayRange):
            try:
                print "drawValueBars: processing data for h5day", day
                pylab.subplot(*codes[day-self.dayRange[0]])
                                    # eg: int('5'+'10')+1 = 510+2 = 512 ie 2nd plot in 5 x 1 array of plots

                for i in range(self.nrBins):
                    pylab.bar (i, self.bins[day-self.dayRange[0]][i][index], color='#22cc55')   #[-2]= $value
                pylab.axis([-10, 100, 0, maxVal])               # NB this must come after the previous line
                titl = 'Day '+repr(day)                         # display h5day
                pylab.title (titl, fontsize='small')
```

```
                if (day-self.dayRange[0]) % 5 == 0:
                    pylab.ylabel ('$value', fontsize='small')
#                pylab.xticks(pylab.arange(1,100,1), xt )
#                pylab.xlabel(titl, fontsize='small')
#                pylab.xlabel(' 100 periods', fontsize='small')
                pylab.grid (True)
            except:
                print "drawBinBars: Problem making plot data for a h5day", day+1
                return 0
#        pylab.title('A tale of 2 subplots')
        heading = "On-Market trades for security " + self.secID
        pylab.title(heading, bbox={'facecolor':'0.8', 'pad':5})
#        pylab.plot("Value of Trades in equal time daily periods")
        pylab.show()

fname = "/Users/drw/Documents/R_DATA/CM65top20.h5"
s40 = h5security(fname)
s40.setTimePeriods (days = (62,65), hours = (10,16), minutes = (0,29), seconds = (0,60)) #
s40.getOnTradeData('s40') # raw data is stored in the self.h5data list
s40.makeBins(nrBins=100, nrVariates= 2)
s40.dataToBins()
s40.drawBinBars(-2)
#for item in s40.bins:   print item

print "closing h5file" ; s40.close()    # inherited from h5file class defn

#s40.h5dataBuff1 = s40.h5data[:] # copy the data over
#print "s40.h5dataBuff1", len(s40.h5dataBuff1)
#s40.h5data=[]
#s40.dataToBins(nrBins=100, nrVariates=3)     # v1 = self.nrDays, v2 = self.nrBins, v3 = nrTrades
#s40.drawValueBars((0,10))

"""
seccy = h5security(fname)                           # instantiate for securityID nr 20
seccy.setTimePeriods (days = (0,65), hours = (10,16), minutes = (0,29), seconds = (0,60)) #
for sec in seccy.securitiesDict.keys():
    acount, bcount = 0, 0
    seccy.getBidOrders(sec) # get raw data from the self.h5data list
    for b in seccy.h5data:
        if flagInMaskQ('t', b[4]):
            print "B::", b
            bcount += 1
#        printFlagsInMask(a[4])

    seccy.h5data=[]
```

```
    seccy.getAskOrders(sec)
    for a in seccy.h5data:
        if flagInMaskQ('t', a[4]):
            print "A::", a
            acount += 1
#        printFlagsInMask(a[4])
seccy.close()
"""
"""
acount = 0
for a in s20.h5data:
    if flagInMaskQ('t', a[4]):
#        print "A::", a
#        printFlagsInMask(a[4])
        acount += 1
print "-----------------------------"
bcount = 0
for b in s20.h5dataBuff1:
    if flagInMaskQ('t', b[4]):
#        print "B::", b
#        printFlagsInMask(b[4])
        bcount += 1
print "NrAsks::", acount,  "NrBids::", bcount,

    def _securityExistsQ(self):
        res=False
        if self.secID not in self.securitiesDict.keys():
            if len (self.securitiesDict.keys()) > 0:
                print secID, "not in h5 file."
#                return 0
        else:
            try:
                print " Loading security Names from ", self.shortFileName
                self._getAllSecurityNodes()
                res = True
            except self.h5fileError, self.fileName:
                pass
#                    return 0
        return res
"""
```