Quote and Trade Binning with a Vector Database

<u>HOMEWORK:</u> The purpose of this homework is for you to walk through each essential part of the code in taq_tools.q so that you understand it. You will use this code to prepare data for analysis in later parts of this course.

<u>KDB+ AQUISITION:</u> These directions will guide you through the download and installation of a free version of the kdb+ database software package. These notes are intended for a mac / linux installation, windows installers should read through the kx webpage.

1. DOWNLOAD: In a web browser navigate to `www.kx.com`−>Software−>Download kdb (32−bit). Select the download the corresponds to your OS, fill in your email, and click the download button where you agree to Kx's terms and conditions. The downloaded file is a simple zip file.

2. UNZIP/INSTALL: Make a directory where the downloaded software will be installed. For instance, /opt/sw/kdb_32. Copy the downloaded file to this path and unzip. Underneath the install path will appear a q/ path. The application appears in q/∗32/q where ∗ is a character that corresponds to your OS.

3. ENVIRONMENT VARIABLES: You need to augment your PATH and set the envvars QHOME and QLIC as follows:

   ```
   $ export PATH=/opt/sw/kdb_32/q/l32:$PATH
   $ export QHOME=/opt/sw/kdb_32/q
   $ export QLIC=/opt/sw/kdb_32/q
   ```

   where you need to substitute your install path instead of /opt/sw/kdb_32. These exports should be added to your ~/.bashrc file.

4. 32-BIT APPLICATIONS: On Ubuntu you need the package ia32−libs in order to run 32-bit applications. Install this package if you don't have it already.

5. FIRST START: To test your install, verify that the application path is correct by using `$ which q`. If this is correct, start q from the command line:

   ```
   $ q
   KDB+ 2.8 2011.12.02 Copyright (C) 1993-2011 Kx Systems
   l32/ 4()core 3761MB jaydamask [computer] 127.0.1.1 PLAY 2012.03.01
   q)
   ```

   To exit q use the \\ command:

   ```
   q)\\
   $
   ```

6. Wrapper: For linux and mac you need to wrap the command-line q interpreter with rlwrap to enable line-edit mode. Add an alias to your ~/.bashrc file such as

   ```
   alias q32='rlwrap -rc -l $HOME/.rlwrap/q.`date +%Y%m%d`.$RANDOM.log q'
   ```

   The −l option writes all standard input/output to the specified file. You may need to create the $HOME/.rlwrap path. From now on you should use the command q32 to start q.

7. Reference: There are two decent references for the q language. A pure keyword reference is found here: `http://code.kx.com/wiki/Reference`. A thorough language discussion, although not very pedagogical, is here: `http://code.kx.com/wiki/JB:QforMortals2/contents`.

8. Notes: The 32-bit trial kdb+ software that I've directed you to download exits execution after running for two hours. Its best that you write scripts in separate files to preserve your work, and, for linux and mac, use rlwrap −l to preserve your sessions.

   If the page `code.kx.com` requires a login just follow the instructions for anonymous logon.

   If you are using emacs as your editor then you can download and install a kdb+−mode lisp file. Search the web for kdbp−mode.el.

Homework Package: I included two q-scripts in this homework package: taq_tools.q and taq_import_examples.q. Place these files is a convenient directory. Loading the taq_tools.q script into q defines several functions but otherwise takes no action. Loading the taq_import_examples.q executes several commands that, taken together, generate quotes and trades on wall-clock bins. You will need to modify the taq_import_examples.q script to point to your specific paths.

To load a script and check that it is there, execute the following [1]:

```
q)\l <path>/taq_tools.q
q)key `
`q`Q`h`o`taq
```

Note that q doesn't expand the path ~/ (the home path) so you'll need to use full pathnames. The `taq context is where the functions from taq_tools.q reside. To execute a function you can either prepend the context name, or switch contexts outright. For instance,

```
q).taq.import_quote_file[<path>/<file>]
2012.01.02T16:47:41.877 taq | loaded file <path>/<file>
2012.01.02T16:47:41.906 taq |   there are <N> records
q)\d .taq
q.taq)import_quote_file[<path>/<file>]
```

[1] Note that \l cannot load a file when written into a script. Instead you must use the more general syntax: `system "l ", path_string, "/", file_string` .

```
2012.01.02T16:47:41.877 taq | loaded file <path>/<file>
2012.01.02T16:47:41.906 taq |  there are <N> records
```

For this course I recommend the former approach. This is because the loaded table, named quote in this case, is assigned to the main context (`.). Using the latter approach loads the quote table into the .taq context. The rest of the code I've written assumes that quote resides in the main context.

You will also need data. Prof. Richard Holowczak has generated TAQ data for this class. I have extracted a quote and trade file for the DOW 30 for date 2010.01.05. You can download these files by following these links:

```
https://www.dropbox.com/s/h32uawyvriew97a/taq_20100105_quotes_dow30.zip?dl=0
https://www.dropbox.com/s/068pb9ljnehmwf9/taq_all_20100105_trades.zip?dl=0
```

Unzip these files and place them in a convenient directory. You will need to point to these files in your scripts, as well as in the taq_import_examples.q script.

Problems:

1. **Loading a TAQ Quote File:** Using the function import_quote_file [] import a quote file. For each question provide the code and its output:
   (a) Explain the code `quote set ("SDTFFIIIC"; enlist ",") 0: hsym "S"$ file_; that appears in the function import_quote_file [].
   (b) How many records are there in the table quote?
   (c) What are the types of each column?
   (d) Extract a list of the column names
   (e) Extract a list of unique SYMBOLs.
   (f) Query for the number of records for each SYMBOL.
   (g) Query for the number of exchanges (EX) for the SYMBOL AA.
   (h) Query for the number of modes (MODE) for the SYMBOL AA.
   (i) Find on the internet the association of these taq exchange characters to the full name of the exchange. Also find, if you can, the explanation of the MODE column.
2. **Loading a TAQ Trade File:** Using the function import_trade_file [] import the trade file taq_ALL_20100105_trades.csv. For each question provide the code and its output:
   (a) What are the types of each column?
   (b) Extract a list of the column names
   (c) Query for the number of records for each SYMBOL.
   (d) Query for the number of records for each SYMBOL, EXCHANGE.

(e) Find on the internet the association of the COND fields with condition description.

3. **Uniform Wall-Clock Time Ruler:** Using the function make_time_ruler [] as a template, write the function make_time_sec_ruler [] to generate time points uniformly spaced in seconds. The argument list of this function should read: make_time_sec_ruler[start_; end_; dsec_]. Write this function into the .taq context.

4. **Prevailing Quote and Intensity Bins:** This problem is associated with the function:

```
/ Given a quote table, a time ruler, symbol and exchange
/  returns a table of most-recent quotes as of the times
/  on the time ruler and adds the CNT column which is a
/  count of the # of records between each time-point.
/ symbol_: type string
/ exch_: type string
/ time_ruler_: constructed from .taq.make_time_ruler[..]
.taq.make_quote_bars: {[symbol_; exch_; time_ruler_]

  / reorders the columns of the final table to that of quote
  ((cols quote), `CNT) xcols

    / joins time_ruler back into the result table
    / 'join' is the comma ,
    / 'each' is an adverb and is the single quote '
    / 'join-each' is ,'
    time_ruler_ ,'

      / take difference of CNT to get # quotes in each interval
      update CNT: deltas CNT from

        / asof join between selected quotes and the time_ruler
        /  the update adds the row index where the joins are
        /  made for the resulting table
        (update CNT:i from
          select from quote where SYMBOL="S"$ symbol_, EX=exch_, MODE=12
        )
        asof
        time_ruler_
  };
```

Commands in q are executed in right-to-left order, just as matrices are multiplied right-to-left. Parentheses allow for grouping, and special keywords such as select and update follow different parsing rules.

There are three arguments to this function: symbol_, exch_ and time_ruler_. You are free to work on the problems below as you like, but what I suggest is that you first load a quotes file so that you have a quote table in the main context; make a time ruler (with minute or second intervals, either way); and make direct assignments such as:

```
q)symbol_: "AA"
q)exch_: "T"
q)time_ruler_: ruler
```

(a) Execute the query, report and explain your result

```
update CNT:i from select from quote where SYMBOL="S"$ symbol_, EX=
    exch_, MODE=12
```

(b) Execute the query, report and explain your result

```
(update CNT:i from select from quote where SYMBOL="S"$ symbol_, EX=
    exch_, MODE=12) asof time_ruler_
```

Assign the result of this command to the table t1.

(c) Execute the query, report and explain your result

```
update CNT: deltas CNT from t1
```

Assign the result of this query to the table t2.

(d) Execute the query, report and explain your result

```
time_ruler_ ,' t2
```

(e) Execute the command, report and explain your result

```
(cols quote), `CNT
```

(f) The function .taq.make_quote_bars [...] returns a table that is the result of the preceding sequence of commands. Save an example of this table to a csv file using .taq.save_csv ["filename"; .taq.make_quote_bars [...]] and plot CNT vs. TIME. Add this plot to your homework.

5. **Trade-Intensity Bins:** This problem is associated with the function:

```
/ Given a trade table, a time ruler, symbol and exchange
/  returns a table of most-recent trades as of the times
/  on the time ruler and adds the CNT and VOL columns.
/  CNT, like that for make_quote_bars, is the number of trade
/  events in each interval.
/  VOL is the accumulated traded volume per interval.
/ symbol_: type string
/ exch_: type string
/ time_ruler_: constructed from .taq.make_time_ruler[..]

.taq.make_trade_bars: {[symbol_; exch_; time_ruler_]

  / constrained selection from trade
  T: select from trade where SYMBOL="S"$ symbol_, EXCHANGE=exch_, COND in
       (`;`$"F";`$"@";`$"@F");

  / I don't need a local variable but the code is less cryptic this way.
  / creation of t follows the form in make_quote_bars[]
  t: ((cols trade), `CNT) xcols

    time_ruler_ ,'

      / note: unlike quotes I don't take CNT differences yet

      (update CNT:i from T)
      asof
      time_ruler_;

  / vector cut:
  /   list_l _ list_r
  / cuts list_r at indices specified by list_l, giving a list of lists
  / sum each (list of lists) sums the values of each list, giving a list
  update VOL: sum each t[`CNT] _ T[`SIZE],
         CNT: deltas CNT
    from t
  };
```

There are some essential differences compared with the quotes function. We will walk through this code as before.

(a) Execute the query, report and explain your result

```
update CNT:i from select from trade where SYMBOL="S"$ symbol_,
     EXCHANGE=exch_, COND in (`;`$"@";`$"F";`$"@F")
```

(b) Execute the queries, report and explain your result

```
T: select from trade where SYMBOL="S"$ symbol_, EXCHANGE=exch_, COND
      in (`;`$"F";`$"@";`$"@F");
t: ((cols trade), `CNT) xcols
  time_ruler_ ,'
    (update CNT:i from T)
    asof
     time_ruler_ ;
```

(c) Execute the commands, report and explain your result

```
q)t[`CNT]
q)count t[`CNT]
q)T[`SIZE]
q)count T[`SIZE]
q)t[`CNT] _ T[`SIZE]
q)sum each t[`CNT] _ T[`SIZE]
```

(d) Execute the query, report and explain your result

```
update VOL: sum each t[`CNT] _ T[`SIZE],
       CNT: deltas CNT
  from t
```

(e) The function .taq.make_trade_bars [...] returns a table that is the result of the preceding sequence of commands. Save an example of this table to a csv file using .taq.save_csv ["filename"; .taq.make_trade_bars [...]] and plot VOL, CNT vs. TIME. Add this plot to your homework.