Data Issues in Computational Finance

CS522 – Tibor Janosi

Looking Back

- HW2 was a sample of the real world.
- For most of you, it took a long time.

But you had it good:

- 62 MB of data is **not big** for a financial database.
- Your data spanned a very short period of time – it was homogeneous.

So, why was this homework hard? What have we learnt?

Financial Data Sets Are …

- Large - many transactions;
- Complex - data comes from several sources;
- Redundant – data is repeated, increasing the potential for errors;
- Inconsistent – different parts of the dataset might provide different views of reality;
- Incomplete – data is often missing;
- Formatted inconsistently – it is not rare to see file formats for the same underlying dataset to chance over time, sometime without notice.

Financial Data Sets Are … (2)

- Often made available as text files, primarily for portability across operating systems, file systems, and programming languages.
- Expensive to correct – in practice it is almost always too expensive to correct an incorrect value, or to retrieve a missing one.
- Just plain wrong – if one is a purist, almost no financial data is usable.

Should we give up? Of course not!

Now What?

- One needs to make reasonable compromises based on one's Finance and CS knowledge.
  Positive consequence: if data is bad to start with, we might get away with less precise, but fast numerical methods.
- The success of a Finance project critically depends on the quality of the data checks.
- Trivial, but true: "Garbage in, garbage out."
- One often spends more time on checking, fixing, and moving data around than on the computation itself.

Data Checks

- For a real project, test for all "sanity" conditions that you can think of; you will learn that there were many errors you have not anticipated anyway.
- Check for "impossible" conditions. See if a field defined as numeric contains alphabetic characters – you will be surprised of what you find.
- Impose strict conditions from the beginning; it is better to relax them later, if needed, than to proceed in the opposite direction. Why?

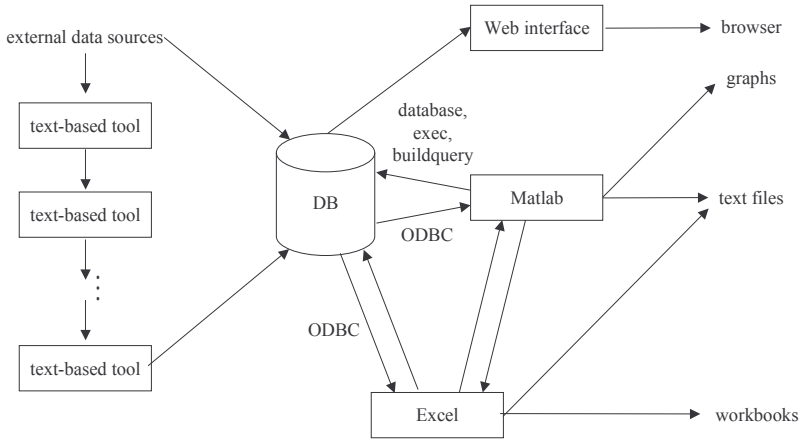
Know Your Data

- Design ways to represent your data so that you can get a qualitative and quantitative sense for the values that you handle.
- Make sure that you examine data
- If you can not infer “absolute” rules, use heuristics.
- Use data mining tools – they save a lot of time.

Tools

- Learn your text processing tools: (g)awk, cut, (e)grep, sort, uniq, sort, etc.
- Excel/Access, MySQL, PostgreSQL, Oracle.
- Know your tools’ limitations.
- Do not over-specialize; try to reuse your work (e.g. do not hardwire constants).
- Document your work thoroughly.
- Process data in stages; check it between stages. It is not rare to inadvertently introduce errors during data cleaning.

A Possible Structure



Matlab

- Needed for non-trivial computations (can also use Octave, Mathematica, etc.);
- Can interact with ODBC-compliant databases, Excel;
- Can read/write text files or binary information (see “save” and “load”);
- Can be compiled to generate stand-alone programs, libraries, COM objects. This speeds up things considerably.
- Matlab programming is not C/C++/C#/Java with different keywords, for speed, you need to gain efficiency.

Inverse Problems

- Typical in Finance (and many other areas of science).
- Theory tells us how to get from underlying reality (e.g. forward rates) to observables (e.g. Treasury prices).
- In reality, it is the other way around: prices must be used to determine underlying reality (parameters of forward rate).
- In most projects, one needs to solve a few inverse problems to get to the underlying parameters, then use these to produce theoretical values for observables that have not been used to infer the parameters.