Architecture of the tests for `ctsa`

The statistical library named `ctsa` has a very broad number of statistical algorithms for time series processing, carefully written in the high performance C programming language. Its maturity is defined as beta by its core developer, Rafat Hussain. It certainly deserves to have its maturity improved to achieve a broader audience.

One of the forms currently used to increase the maturity of a software is by applying systematic testing to it. This text considers an architecture of tests to that purpose.

The concept of software testing is almost as old as software development itself. There many libraries, frameworks and concepts that can surpass the limits of technology, science and get closer and closer to philosophy. This proposal is much more pragmatic and considers only a number of techniques that can be used to achieve results in a short time.

Other techniques can and will be applied to a software with a spectrum of usage as broad as `ctsa`. This architecture is designed with the aim of providing an ample basis upon which other testing techniques can be applied systematicly.

Definition of terms

First of all, it's important to remember the current generation of testing techniques favor what's called *automatic* or *self-testing*, meaning that the result of a test can be defined by the test itself, without the need of human evaluation.

That has two implications: 1st, the tests need to have a simple logic, and yield only the results of true or false, that are usual to computer programming and are said *pass* and *fail* in the context of testing.

2nd, the fact that testing is automatic allows the application of many and many tests, and therefore to cover a large part of the software being tested.

Another important point is that, even though tests will yield true or false, or in the test jargon, passing or failing, doesn't mean necessarily that the part of the code that was tested by it is buggy or has an error. It only means that this part of the code should be better examined, as the logic of the test can be wrong.

There many and many ways of classifying the tests, but for the sake of the simplicity of this text, let's consider just the simplistic point of view of only two types of tests: microscopic, or *unit tests*, and macroscopic, or *functional tests*.

The macroscopic, or functional tests focus on the broader results of the code, without taking into consideration the details of its inner working.

The microscopic, or unit tests are focused in details of the code, and is usually written by the programmers that are developing the software. They tend to be simpler than the macroscopic tests, as far less degrees of freedom are involved in them, and therefore to give results that can be interpreted more easily.

Unfortunately, they demand much more time to cover significant areas of the code.

Therefore, in this moment, we shall be focusing in macroscopic tests. Our simplistic definition will be presented in the next section.

Types of functional tests

Just as in the previous section, let's consider just two types of macroscopic or functional tests: the regression tests and the benchmarking tests.

By *regression testing* we mean a set of tests intended to make sure that the current version of the software being tested has the same good results as a previous version that is *in production*, is mature, was validated etc.

By *benchmarking testing* we mean to determine the quality of the software being tested by comparing its results against the results of another software for the same features.

The usual meaning of benchmark in computing is the execution speed or the consumption of computer resources of a given software, like memory use, network traffic, etc. In this case we borrow the term from the field of business administration where a company is compared to another in the same market.

We intend to design a test architecture that can be used to apply both regression tests and benchmark tests.

Similar softwares that will be considered

Two statistical softwares shall be considered to be compared against `ctsa`. They are:

- `forecast` is a library developed by a team led by Rob Hyndman, a notable teacher, researcher and practicioner of forecasting in general, and time series in particular.

  It is written in R, and many researchers consider `forecast` as the state of art in the field of time series forecasting;

- `statsmodels` is a very ambitions statistical library written in Python and generally considered one of the best in the statistical modelling in Python.

Features of `forecast` and `statsmodels` that are similar to the ones available in `ctsa` will be identified in order to create the testing databases that will be considered in the benchmarking tests.

Detailing the test software

The software modules in `ctsa` follow a *object-based* structure. That means they do not intend to simulate a strict object oriented structure, but they do have an object-like structure, with a state that's composed of parameters and execution results.