Unit testing: An audio research example

An audio-to-note method for solo vocal music recordings.
http://code.soundsoftware.ac.uk/projects/cepstral-pitchtracker

This method, implemented in C++, consists of:

1. Short-time Fourier transform using FFT library
2. Transform to cepstral domain
3. Peak finder and interpolator
4. Multi-agent method for identifying candidate notes

Although we have no idea what output to expect from the complete system when faced with real-world input, we can test the individual stages readily using small synthetic test data sets.

1. Calculate a 4-point FFT of various real-valued signals and check the outputs.
   Review your memory of Fourier transforms by working out the results by hand:
   a. Constant DC offset (1, 1, 1, 1)
   b. Cosine (1, 0, -1, 0)
   c. Sine (0, 1, 0, -1)
   d. Nyquist (1, -1, 1, -1)
   e. Dirac impulse (1, 0, 0, 0)
3. The cepstrum is a trickier proposition, but with a bit of experimentation to see how it actually functions we can come up with some sensible test cases (non-harmonic input, simple flat harmonic formation) and possibly improve our understanding of it as well.
   Even two or three simple tests are worth having. If something puzzling comes up later, we can write a test for it then.
4. Peak finding and interpolation has some details that are easily overlooked: we should check for example what happens when the peak is at one end or the other of our input array (so there is nothing to interpolate on one side).
   For our tests, we don't check exactly where the peak interpolator places the peak: we only check that it's within a reasonable range, for example closer to one sample than its neighbour. This allows us to change the interpolation strategy while continuing to test that we have a valid interpolator. We could make separate tests for the exact peak calculation.
5. Our agent code is implemented as a class with a method which accepts a new peak estimation value, reports whether it is consistent with the internal values already accepted, and updates its state appropriately.
   We provide it with a series of synthetic values with faked timestamps designed to test specific accept/reject behaviour.
   This works well in an exploratory research situation, because we can try out different accept/reject behaviour reliably by designing the acceptance criteria into the tests first, and then updating the implementation to make it pass the new tests.

There's still one omission: we have no tests for the code that creates, selects, and prunes the agents. Pulling that code out into its own class and testing it would be a good next step.

You can find all the code for these tests (using the Boost test framework) in the repository at https://code.soundsoftware.ac.uk/projects/cepstral-pitchtracker/repository/show/test .