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. Agent that takes a series of pitch peaks and tests to see if they form a plausible note
5. System that supplies the pitches to agents, keeps valid notes, and discards failures

Although we have no idea what output to expect from the complete system when given real-world input, we can test these individual stages 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)
2. For the cepstrum, but with a bit of experimentation we can come up with some sensible test cases (non-harmonic input, simple flat harmonic formation) and perhaps improve our understanding of it at the same time. Even two or three simple tests are worth having. If something puzzling comes up later, we can add a test for it then.
3. Peak finding and interpolation has some details that are easily overlooked: we should check for example what happens when the peak is at the end 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.
4. A note hypothesis agent is a class with a method which accepts a new peak estimate, reports whether it is consistent with estimates already accepted for this note, and updates internal state appropriately. We provide it with a series of synthetic values, with faked timestamps, designed to test specific accept/reject criteria.
   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.
5. The code that creates, selects, and prunes the note agents—the higher-level algorithm—can also be pulled out into its own class, and we can test it with series of faked pitch estimates designed to exercise a likely failure case (e.g. overlapping notes, simultaneous notes, perfectly adjoining notes, notes separated by silence, no notes).

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 .