Casa Framework for Global Fringe Fitting

Circulation: Bourke, Brisken, Kern, Moellenbrock, Stewart, van Langevelde

This document describes the framework proposed for research and development of global fringe fitting functionality in Casa.

The existing KJones class in Casa handles antenna based delay errors in the following manner:

- A reference antenna is specified, and designated to have a delay of 0.
- Baselines to this antenna are used to calculate the relative delays of the other antennas.
  - This is accomplished means of a Fourier transform along the frequency axis providing delay solutions (but not rates).
  - The input to the FFT is padded and the neighboring values of the peak in delay space are incorporated by means of parabolic interpolation, yielding the output peak, which is converted to nano-seconds and stored.
- Also provided by the KJones class are methods to convert these delay values to phase corrections which can be applied to the data, and a method to manually specify delay values.
- In addition to the KJones class, classes are implemented to solve the cross polarization delay offset (KcrossJones) and to correct for specified antenna position offsets (KAntPosJones).

An additional class KTest will be added, initially inheriting all its functionality from KJones. The aim of the class is to solve for antenna based delay, and initially rate but optionally higher order terms, taking into account all available data. The solve logic exists in the solveOneVB method. A multi stage determination of these parameters will be implemented. These stages will be represented as classes implementing a generic (in the context of fringe fitting) solve engine, which will be iterated over in the solveOneVB method.

A solve engine shall provide a solveDelay method which will take as input a VisBuffer optionally an initial estimate of the solution parameters. It will return a set of solution parameters. KTest may iterate over these engines to refine a solution using different methods. For efficiency, a method will be provided to request how many parameters the engine solves for, to allow a single set of solutions to propagate down the chain without being replaced or resized.

Currently two stages are defined:

- An FFT stage. Similar to that in KJones but using a 2D FFT to estimate rates as well as delays.
- A Least Squares based stage.

Baselines may be be stacked to achieve a higher baseline signal to noise by exploiting the phase closure relationship of visibilities. This will be implemented outside the KJones / KTest scope where it will be available to other routines in Casa if desired.

Baseline stacking will be provided as an option in the FFT stage and investigated as an alternative to a full global solution in the least squares stage.

A first implementation will be a refactoring of the existing KJones class using the above framework.

```
class FringeEngine {
        virtual int nTerms();
        virtual solveDelay(VisBuffer&, SolveParameters&);
};
```

(TBD, SolveParameters a place holder. What should the real class be?)

```
void baselineStack(VisBuffer&, VisBuffer&, Int=-1);
// The returned VisBuffer can contain all baselines or just those to a reference antenna.

class KTest : KJones {
public:
  // Constructor
  KTest(VisSet& vs);
  KTest(const Int& nAnt);

  virtual ~KTest();

  // Return the type enum
  virtual Type type() { return VisCal::K; };

  // Return type name as string
  virtual String typeName()     { return "K Test"; };
  virtual String longTypeName() { return "K Test (global delay,rate)"; };

  virtual void solveOneVB(const VisBuffer& vb);

private:
  Vector<FringeEngne> solvers;
};
```

(TBD: Would the sovers would be setup in setSolve? Not important immediately)