# Inference:
# A Python Package for Astrostatistics

## *A NASA AISR Project*

Tom Loredo, Alanna Connors, Travis Oliphant

Cornell/Eureka Scientific/BYU

# Motivation

Many advanced methods are conceptually simple but computationally complex.

Competing methods of very different levels of sophistication are often similar from end-user’s perspective.

Principal obstacle to understanding/use is the *art of statistical computing*.

*Eliminate this obstacle!*

# The Inference Project

- The `Inference` package
  - Library: Deep and broad collection of self-contained tools tailored to astronomers' needs; *multiple methods*
  - Parametric Inference Engine: Framework for parametric modeling; multiple methodologies ($\chi^2$, likelihood, Bayes) with unified interface
- Use of a modern VHL language: Python
  - Single implementation facilitates depth/breadth
  - VHL features speed development, facilitate testing
  - Easy access for new users and PyRAF users
- Outreach
  - Astrostatistics speakers and sessions at conferences
  - Selected methods from sessions in the package

# A Bit About Python

- A general purpose language with a rich standard library
- Sophisticated and fast scientific computing capability
- Simple syntax—resembles “pseudo code,” Matlab/IDL
- Use interactively, or via scripts/modules
- Object oriented, with a very simple object model—facilitates high level interfaces, modularity
- Practical rather than “pure”—Selected capabilities of various paradigms (e.g., functional programming, list comprehensions, metaclasses)
- Easily extendible/embeddable with C/C++/Fortran
- Open source, cross-platform, active & growing user community

# Scientific Computing With Python

- Array computations
  - Syntax inspired by Matlab/IDL/Fortran90
  - Performance near that of C/Fortran
  - `Numeric:` Developed by LLNL/MIT scientists
  - `numarray: Numeric`'s successor from STScI
- PyRAF — The IRAF command line in Python (STScI)
- SciPy (partly supported by Enthought, NASA)— special functions, linear algebra, FFTs, DSP, quadrature, ODE solvers, optimizers, basic stats
- Plotting: `matplotlib`, `Chaco`, various libraries

# Simple Example: The Rayleigh Statistic

Search for periodic signals in arrival time series, $\{t_i\}$.

$$R(\omega) = \frac{1}{N}\left[\left(\sum_i \sin \omega t_i\right)^2 + \left(\sum_i \cos \omega t_i\right)^2\right]$$

# Sample Source Code

**Python source code**

```
from Numeric import *

def Rayleigh (data, w):
    wd = w*data
    return (sum(sin(wd))**2 +
         sum(cos(wd))**2)/len(data)
```

**C source code**

```
#include <math.h>

double Rayleigh (int n, double *data,
                 double w) {
    double S, C, wt;
    int i;
    S = 0.;
    C = 0.;
    for (i=0; i<n; i++) {
        wt = w*data[i];
        S += sin(wt);
        C += cos(wt);
    }
    return (S*S + C*C)/n;
}
```

# Library: Tools for Continuous Data

*Sampled functions with additive noise,* $d_i = f(t_i) + e_i$

- Linear & nonlinear regression: Bershady/Isobe packages, Bayesian EVM, correlated errors
- Detection/measurement of periodic signals
  - Standard approaches: Power spectrum, Schuster periodogram, Lomb-Scargle
  - Bretthorst algorithm (Bayesian periodograms); Kepler periodogram
  - Bayesian piecewise-constant modeling (Gregory method)
- Nonperiodic time series analysis: ARMA, BB, long-mem.
- Robust estimation/outlier detection

# Library: Tools for Discrete Data

- Intervals and limits for rates and ratios from counting: Feldman-Cousins likelihood ordering, Bayes, ABC, profile likelihood
- Periodic point processes (period searching in arrival time data): Rayleigh statistic, $Z_N^2$, Bayes log-Fourier models, Gregory-Loredo, adaptive 1-d grid, accelerated ($P$,$\dot{P}$) searching, fractional transforms
- Inhomogeneous point process models for local event detection: Bayes blocks, Poisson “Haar” wavelets
- Survey analyses: Survival analysis (ASURV), Bayes point process $+$ noise
- Nonparametric methods: Adaptive splines, neural nets (interfaces to Max Planck PPI methods), mixture models

# Parametric Inference Engine

- Three methodologies: $\chi^2$, likelihood, Bayes
- Data types: Point samples, binned, folded; on/off; surveys
- Automate standard parameter exploration tasks
  - Exploration on equispaced & logarithmic grids
  - Optimization (unconstrained and with boundary constraints)
  - Exploration of subsets of parameter space (profiling/projection)
  - Hessian/information matrix calculation
- Bayesian computation
  - Marginalization and Bayes factors via adaptive quadrature & Laplace approximation
  - Calculation of 1-d, 2-d, 3-d credible region boundaries
  - Basic Markov chain Monte Carlo (MCMC) support
- Simulate data

*Build a model:*

```
class PowerLawModel(ParametricModel):
    A = RealParameter(1., 'Amplitude')
    alpha = RealParameter(range=(-5,-1), 'Index')

    def signal(self,E):
        return self.A*E**(self.alpha)
```

*Associate data with predictor:*

```
p1 = SampledChisqrPred(data1)
p2 = BinnedChisqrPred(data2)
```

*Make inferences:*

```
inf = ChisqrInference(PowerLawModel, p1, p2)

inf.A.logStep(1., 10., 50)

inf.alpha.vary()
grid = inf.opt()
```

Returns a grid object w/ projected $\chi^2(A)$