LCL a library for modelling continuous systems

Erik Arne Mathiesen
EuroSciPy 2013

Project Aim

Easy definition and simulation of various dynamical systems

- Deterministic/Stochastic
- Linear/Non-linear
- Simple/Composed
- Single/Multi-dimensional

Flexible distribution of computations
Computations can be distributed locally at operator level and remotely at sub-system level.

System definition is independant of underlying type
The definition and approximation steps are the same whether the system is deterministic or stochastic only the simulation step differs.

Several interesting extensions:

- Selection: parameter selection by optimisation
- Reasoning: proving properties about solutions/approximations
- Solving: mechanically finding solutions/approximations

Model selection

Builtin optimisation schemes can be used for selecting model parameters or boundary conditions for a given data set.
Several types of optimisation schemes are supported for selection.

- **Stochastic schemes:** Simulated Annealing or Differential Evolution
- **Gradient schemes:** Newton or Levenberg-Marquardt

Gradient schemes work very well in LCL.

What's next?

- **GA model exploration:** finding closed form solutions for dynamical systems using genetic algorithms
- **CAD-based reasoning:** the ability to prove or disprove properties about LCL defined systems

Basic Flow

- **Definition** The system is defined using a simple grammar

```
SYSTEM = equation(S,S)
     S = B | int(S,V) | diff(S,V) |
         add(S,S) | mult(S,S) | apply(S,V,S)
     B = V | constant(...) |
         function(...) | real(...)
     V = variable(...)
```

- **Approximation** Dimensions, free variables, constants and boundary conditions are defined and an approximation object is created

```
def approximate(system,
                constants=[],
                boundary=[],
                variables=[],
                functions=[],
                dimensions=[],
                base=None)
```

- **Simulation** A set of simulation points is defined and the simulation is performed using the approximation object

```
def simulate(approximation,
             points,
             function={},
             statistics=None)
```

- **Result analysis** Simulation results can be handled in several ways:
  - Access to raw data
  - Integrated visualisation using Matplotlib
  - Integration with Numpy
  - Integration with Pandas

Simplest Example: Exponential function

The exponential function is one of the simplest dynamical systems

$$\frac{\partial f}{\partial t}(t) = f(t)$$

Definition:

```
equation = lng.equation(lng.variable("f"),
                        lng.diff(lng.variable("f"),
                                 lng.variable("t")))
```

Approximation:

```
dimension_t = lcl.dimension("t", 10)

boundary_t = lcl.boundary("f", dimension_t, 1.0)

appr = lcl.approximate(equation,
                       boundary=boundary_t,
                       dimensions=dimension_t)
```

Simulation:

```
points = [lcl.point(dimension=dimension_t,
                    type=lcl.REAL,
                    value=value*0.1)
                for value in range(1, 10)]

result = lcl.simulate(appr, points)
```

Result analysis:

- Integrated visualisation

```
result.plot()
```

- Numpy integration

```
np = result.ndarray(stat=lcl.statistic.mean())
```

- Pandas integration

```
df = result.data_frame(stat=lcl.statistic.mean())
```

Stochastic process

Stochastics are in Stratonovich space so for a standard Ito process

$$\boldsymbol{X}(t, \boldsymbol{W}_t) = \int \boldsymbol{a}\boldsymbol{X}(t, \boldsymbol{W}_t)\partial t + \int \boldsymbol{b}\boldsymbol{X}(t, \boldsymbol{W}_t)\partial \boldsymbol{W}_t + \boldsymbol{X}(0, \boldsymbol{W}_0)$$

we must adjust the drift, which can be done by the use of a shortcut

```
import lcl.process as prc
equation = prc.stratonovich("X",
                            "W",
                            lng.c("A") * lng.v("X"),
                            lng.c("B") * lng.v("X"))
```

The underlying variable types are defined on a point-by-point basis.

```
point = lcl.point(value1 = {'dimension': dimension_t,
                            'type': lcl.REAL,
                            'value': 1.0},
                  value2 = {'dimension': dimension_W,
                            'type': lcl.GAUSSIAN,
                            'value': 'value1'},
                  iterations = 10000)
```