Modelling dynamical systems with a DSL

Erik Mathiesen

Overview

- Simple technique for simulating differential equations
- Easy to implement, Easy to distribute
- Can be used in many ways: simulation, optimisation, calibration, model selection, etc.
- Code examples of a Python interface for the implementation

Technique - 4 steps

Equation

Input in human readable format

DSL

Functional language describing the "execution" of the equation

Execution

Assign semantics to DSL and execute program

Interpretation

Interpret DSL execution as equation result

Technique - Equation

Simple input language

$S := S + S \mid S * S \mid \frac{\partial}{\partial V} S \mid \lambda(S, V, S) \mid \cdots \mid F \mid V \mid R$
$F := \exp \mid \sin \mid \surd \mid \cdots$
$V := x \mid y \mid z \mid \cdots$
$R \in \mathbb{R}$

Technique - Equation

Simple input language

$S := S + S \mid S * S \mid \frac{\partial}{\partial V} S \mid \lambda(S, V, S) \mid \cdots \mid F \mid V \mid R$
$F := \exp \mid \sin \mid \sqrt{} \mid \cdots$
$V := x \mid y \mid z \mid \cdots$
$R \in \mathbb{R}$

Equation

$y = x^2 + 2$

Technique - DSL

DSL functional language

$S := trace(S, V) \mid \otimes(S, S) \mid \circ(S, S) \mid A$

$A := sum \mid mult \mid conv(V) \mid \cdots \mid function(F) \mid var(V) \mid real(R)$

$F := \exp \mid \sin \mid \sqrt{} \mid \cdots$

$V := x \mid y \mid z$

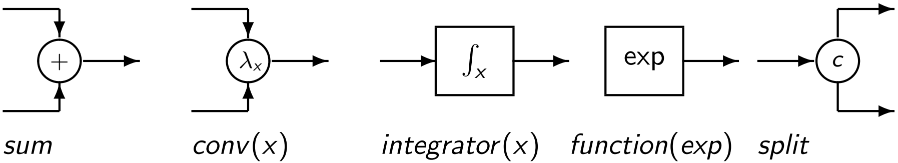
$R \in \mathbb{R}$

Equation

$y = x^2 + 2$

Technique - DSL

DSL graphical language

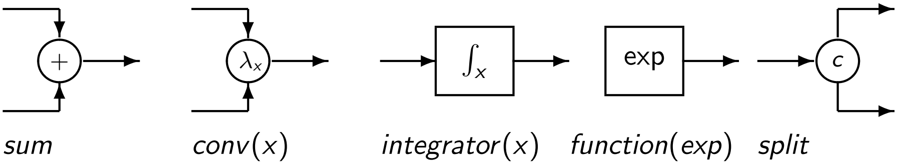


Equation

$y = x^2 + 2$

Technique - DSL

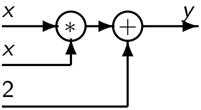
DSL graphical language



Equation

$y = x^2 + 2$

DSL



Technique - Execution

Stream circuits

$$Streams = \{\mathbb{N}^N \to \mathbb{R}\}$$

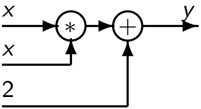
$$T : DSL \to (Streams \to Streams)$$

- $T(sum)\langle s, r\rangle(n) = s(n) + r(n)$
- $T(mult)\langle s, r\rangle(n) = \sum_{n_1, n_2 \in \mathbb{N}^N, n_1 + n_2 = n} s(n_1) * r(n_2)$

Equation

$y = x^2 + 2$

DSL



Technique - Execution

Stream circuits

$$Streams = \{\mathbb{N}^N \rightarrow \mathbb{R}\}$$

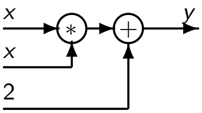
$$T : DSL \rightarrow (Streams \rightarrow Streams)$$

- $T(sum)\langle s, r\rangle(n) = s(n) + r(n)$
- $T(mult)\langle s, r\rangle(n) = \sum_{n_1, n_2 \in \mathbb{N}^N, n_1 + n_2 = n} s(n_1) * r(n_2)$

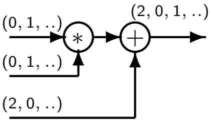
Equation

$y = x^2 + 2$

DSL



Execution



Technique - Interpretation

Polynomial interpretation

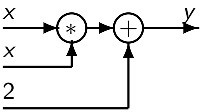
For precision $M \in \mathbb{N}$, point $x \in \mathbb{R}^N$ and $s \in Streams$

$$P_M(s, x) = \sum_{n \in [0,M]^N} s(n) x^n$$

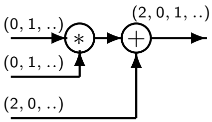
Equation

$y = x^2 + 2$

DSL



Execution



Technique - Interpretation

Polynomial interpretation

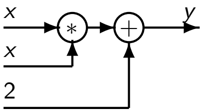
For precision $M \in \mathbb{N}$, point $x \in \mathbb{R}^N$ and $s \in \mathit{Streams}$

$$P_M(s, x) = \sum_{n \in [0,M]^N} s(n) x^n$$

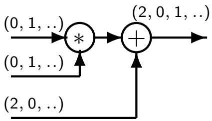
Equation

$y = x^2 + 2$

DSL



Execution



Interpretation

$y(x) = 2 + x^2$

Simple Example

Equation

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

$$f(0) = 1$$

Simple Example

Equation

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

$$f(0) = 1$$

Translation

$$f(t) = f(0) + \int f(t) \partial t$$

$$f(0) = 1$$

Simple Example

Equation

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

$$f(0) = 1$$

Translation

$$f(t) = f(0) + \int f(t) \partial t$$

$$f(0) = 1$$

DSL

$$(boundary(f) \otimes var(f)) \circ (id \otimes integrator(t)) \circ sum \circ var(f)$$

Simple Example

Equation

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

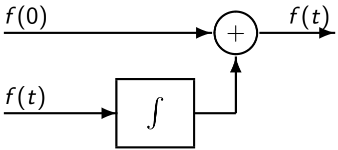
$$f(0) = 1$$

Translation

$$f(t) = f(0) + \int f(t) \partial t$$

$$f(0) = 1$$

DSL Graph



Simple Example

Equation

$$f(t) = \frac{\partial}{\partial t} f(t)$$
$$f(0) = 1$$

DSL Normalised

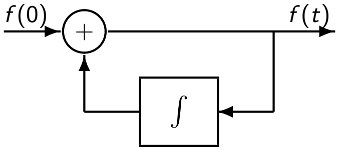
$$boundary(f) \circ trace((id \otimes integrator(t)) \circ sum \circ split) \circ var(f)$$

Simple Example

Equation

$$f(t) = \frac{\partial}{\partial t} f(t)$$
$$f(0) = 1$$

DSL Normalised Graph

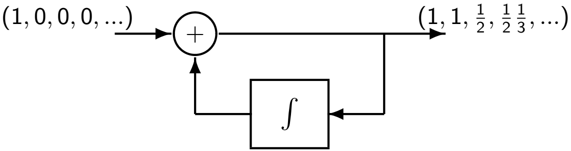


Simple Example

Equation

$$f(t) = \frac{\partial}{\partial t} f(t)$$
$$f(0) = 1$$

Execution



Simple Example

Equation

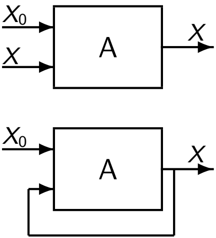
$$f(t) = \frac{\partial}{\partial t} f(t)$$
$$f(0) = 1$$

Interpretation

$$f(t) = 1 + t + \frac{1}{2}t^2 + \frac{1}{3!}t^3 + \cdots$$

DSL Normalisation

- Atomic elements uses normal group rules
- Constructs use categorical rules
- "Closing loops":



Simple Example - code

```
equation = Equation("f = diff(f,t)")

dim_t = Dimension("t", 10)

variables = []

boundary_t = Variable([dim_t], 1.0, "BOUNDARY_f_t")

constants = []

a = Approximate(equation,
                constants,
                [boundary_t],
                variables,
                [dim_t])

point = Point().add_value(dim_t, REAL, 1.0)

result = Simulate(a, 0, [point])
```

Stochastic systems

- In defining the system and executing the DSL there is no mentioning of the underlying types
- Types are defined at the point of interpretation

Another Example - Stochastic system

Equation

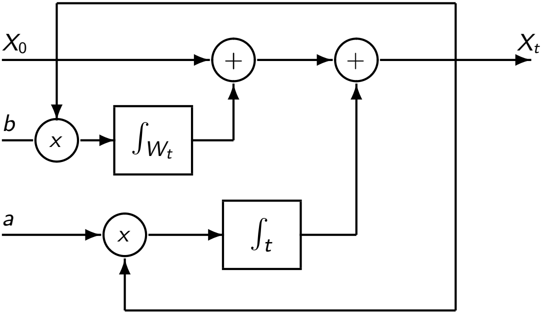
$$X(t, W_t) = \int aX(t, W_t)\partial t + \int bX(t, W_t)\partial W_t + X(0, W_0)$$

Another Example - Stochastic system

Equation

$$X(t, W_t) = \int aX(t, W_t)\partial t + \int bX(t, W_t)\partial W_t + X(0, W_0)$$

DSL

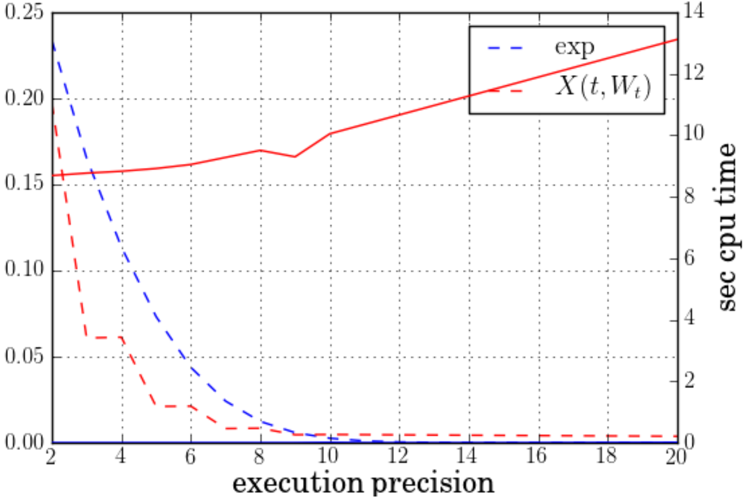


Another Example - code

```
equation = Stratonovich("X", "W", "mult(A,X)", "mult(B,X)")

dim_t = Dimension("t", 10)
dim_W = Dimension("W", 10)

boundary_X_t = Variable(dims, 1.0, "BOUNDARY_X_t")

constant_A = Variable(dims, 0.02, "A")
constant_B = Variable(dims, 0.3, "B")

a = Approximate(equation,
                [constant_A, constant_B],
                [boundary_X_t],
                Variables(),
                [dim_t, dim_W])

point = Point()
point.add_value(dim_t, REAL, 1.0)
point.add_value(dim_W, GAUSSIAN, 1.0)

result = Simulate(a, 0, [point])
```

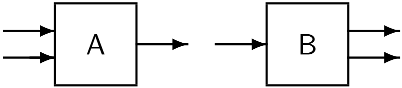
Stochastic systems

Precision



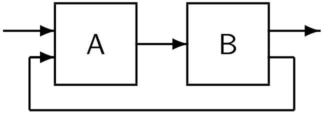
Complex systems

- In the DSL world systems/equations easily combine to create more complex systems



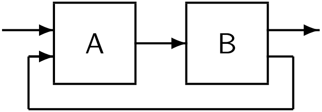
Complex systems

- In the DSL world systems/equations easily combine to create more complex systems



Complex systems

- In the DSL world systems/equations easily combine to create more complex systems

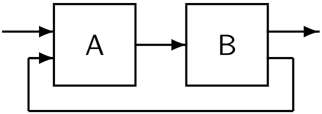


- Other underlying distributions can be used

```
point.add_value(dim_W, POISSON, 1.0)
```

Complex systems

- In the DSL world systems/equations easily combine to create more complex systems



- Other underlying distributions can be used

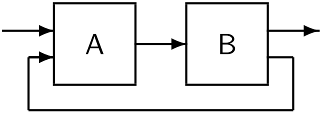
```
point.add_value(dim_W, POISSON, 1.0)
```

- Stochastic dimensions can be correlated

```
point.add_correlation(dim_W_X, dim_W_V, CONSTANT, 0.2)
point.add_correlation(dim_W_X, dim_W_V, VARIABLE, "Z")
```

Complex systems

- In the DSL world systems/equations easily combine to create more complex systems



- Other underlying distributions can be used

```
point.add_value(dim_W, POISSON, 1.0)
```

- Stochastic dimensions can be correlated

```
point.add_correlation(dim_W_X, dim_W_V, CONSTANT, 0.2)
point.add_correlation(dim_W_X, dim_W_V, VARIABLE, "Z")
```

- Language also has some special functions, such as $\sqrt{}$, sin, exp

And a last example

Equation

$$X(t, W_t^X) = \int aX(t, W_t^X)\partial t + \int \sqrt{V(t, W_t^X)} X(t, W_t)\partial W_t^X + X(0, W_0^X)$$

$$V(t, W_t^V) = \int \kappa(\theta - V(t, W_t^V))\partial t + \int \sigma \sqrt{V(t, W_t^V)} \partial W_t^V + V(0, W_0^V)$$

$$W_t^X \cdot W_t^V = 0$$

And a last example

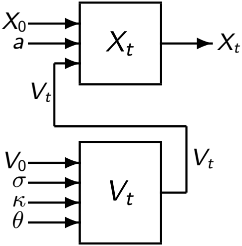
Equation

$$X(t, W_t^X) = \int aX(t, W_t^X)\partial t + \int \sqrt{V(t, W_t^X)} X(t, W_t)\partial W_t^X + X(0, W_0^X)$$

$$V(t, W_t^V) = \int \kappa(\theta - V(t, W_t^V))\partial t + \int \sigma\sqrt{V(t, W_t^V)}\partial W_t^V + V(0, W_0^V)$$

$$W_t^X \cdot W_t^V = 0$$

DSL



And a last example

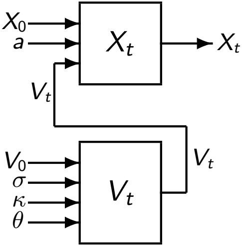
Equation

$$X(t, W_t^X) = \int aX(t, W_t^X)\partial t + \int \sqrt{V(t, W_t^X)}X(t, W_t)\partial W_t^X + X(0, W_0^X)$$

$$V(t, W_t^V) = \int \kappa(\theta - V(t, W_t^V))\partial t + \int \sigma\sqrt{V(t, W_t^V)}\partial W_t^V + V(0, W_0^V)$$

$$W_t^X \cdot W_t^V = 0$$

DSL



Code

```
system = Merge(equation1, equation2)
```

```
point = Point()
point.add_value(dim_t, REAL, 1.0)
point.add_value(dim_W_X, GAUSSIAN, 1.0)
point.add_value(dim_W_V, GAUSSIAN, 1.0)
point.add_correlation(dim_W_X, dim_W_V,
    CONSTANT, 0.0)
```

Distributed Computing

- Lowest level: splitting the computation of a single node
  - Works well with multiple threads

$$T(conv_M)\langle s, r\rangle(n) = \sum_{i=0}^{P(M)} s(n_{i\to M}) \sum_{v\in(\mathbb{N}^N)^i, \sum_j v_j = n} (\prod_{j=1}^{i} r(v_j))$$

Distributed Computing

- Lowest level: splitting the computation of a single node
  - Works well with multiple threads

$$T(conv_M)\langle s, r\rangle(n) = \sum_{i=0}^{P(M)} s(n_{i\to M}) \sum_{v\in(\mathbb{N}^N)^i, \sum_j v_j = n} (\prod_{j=1}^{i} r(v_j))$$

- Higher level: divide circuit into parallel sub-circuits
  - Works well across multiple processes/machines

Distributed Computing

- Lowest level: splitting the computation of a single node
  - Works well with multiple threads

$$T(conv_M)\langle s, r\rangle(n) = \sum_{i=0}^{P(M)} s(n_{i\to M}) \sum_{v\in(\mathbb{N}^N)^i, \sum_j v_j = n} (\prod_{j=1}^{i} r(v_j))$$

- Higher level: divide circuit into parallel sub-circuits
  - Works well across multiple processes/machines

Code

```
SetConfig(MULTITHREADING, True)
SetConfig(RECURSIVE_MULTITHREADING, True)
SetConfig(REMOTE_COMPUTATION, True)
```

Distributed Computing

Equation

$$\frac{\partial}{\partial x} f(x) = \sin(g(x))f(x)$$

$$\frac{\partial}{\partial x} g(x) = g(x)$$

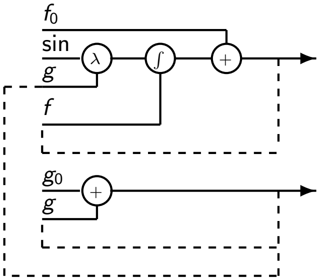
Distributed Computing

Equation

$$\frac{\partial}{\partial x} f(x) = \sin(g(x)) f(x)$$

$$\frac{\partial}{\partial x} g(x) = g(x)$$

DSL



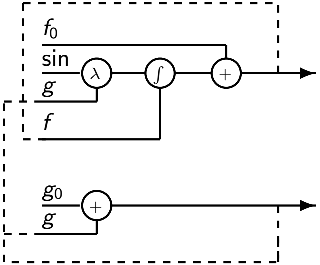
Distributed Computing

Equation

$$\frac{\partial}{\partial x} f(x) = \sin(g(x)) f(x)$$

$$\frac{\partial}{\partial x} g(x) = g(x)$$

DSL



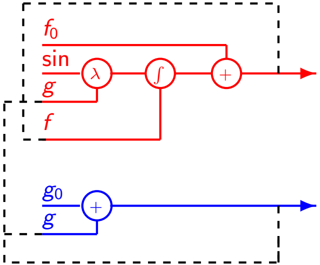
Distributed Computing

Equation

$$\frac{\partial}{\partial x} f(x) = \sin(g(x)) f(x)$$

$$\frac{\partial}{\partial x} g(x) = g(x)$$

DSL



Optimisation

Usage

- For prototyping we can use optimisation to fit observed data
- Either to construct the general model or to fit parameters

Benefits

- Parameters can be viewed as either parameters of the model or dimensions
- Different levels of precision can be used across solution space
- Easy derivatives for parameters

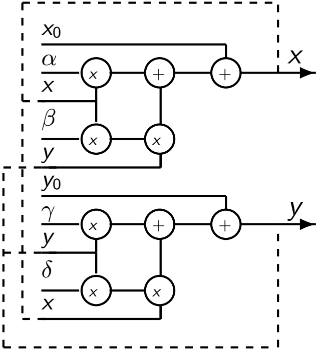
Optimisation - Example

Equation

$$\frac{\partial}{\partial t}x = \alpha x + \beta xy$$

$$\frac{\partial}{\partial t}y = \gamma y + \delta yx$$

Optimisation - Example



Optimisation - Example - code

Optimiser function

```
def eval_function(state, point):
...

def distance(points, values):
...

result = DifferentialEvolution(lower_bound,
                               upper_bound,
                               optimisation_bound,
                               mutation_scaling,
                               cross_over_rate,
                               combination_factor,
                               population_size,
                               points,
                               values,
                               eval_function,
                               10000,
                               distance)
```

Optimisation - Example - code

Evaluator function

```
def eval_function(state, point):
    alpha = state[0]
    beta = state[1]
    gamma = state[2]
    delta = state[3]
    t = point[0]

    ...

    result = Simulate(approximation, iterations, points)
    x_val = result.get_result(0, "x")
    y_val = result.get_result(0, "y")
    return [x_val, y_val]
```

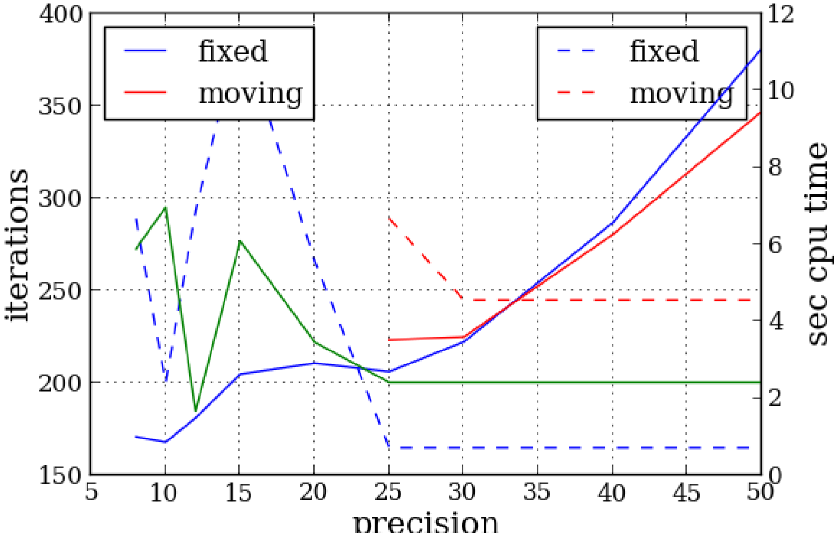
Optimisation - Example - code

Distance function

```
def eval_function(state, point):
  ...

def distance(points, values):
  ...
  for i in range(0, n):
    r1 = points[0][i] - values[0][i]
    r2 = points[1][i] - values[1][i]
    distance1 = distance1 + r1 * r1 * weights[i][0] *
        weights[i][0]
    distance2 = distance2 + r2 * r2 * weights[i][1] *
        weights[i][1]
  d = 0.5 * (math.sqrt(distance1) + math.sqrt(distance2))
  if (last_precision_distance > 2 * d):
    last_precision_distance = d
    if precision < max_precision:
      precision = precision+(end_precision-precision)/2
  return d
```

Optimisation - Example

Precision



The Future

Genetic Algo

Using GA to solve/approximate systems of differential equations.

Input System of differential equations

Output Solution or reasonable closed form approximation

The Future - Genetic Algorithm

Initial Population

Mutated versions of the input system

Population measure

Distance between approximations of input and population systems

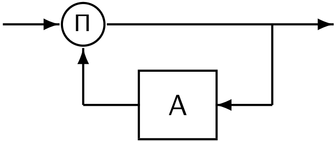
$$d(s,r) = \sqrt{\sum_{n \in [0,M]^N} w_n (s(n) - r(n))^2}$$

Stopping condition

Distance of closed form population smaller than given bound

The Future

Mutation: It is all in the loop



$$\frac{\{\langle X, Z\rangle\}A\{\langle Y, Z\rangle\}}{\{X\}trace(A)\{Y\}}$$

The Future

Mutation - Example

```
ga = GeneticAlgo(
        100,
        condition,
        random_circuit,
        optm.initial_member.initial_random_member(),
        optm.selector.selector_sum(),
        [optm.operator.operator_random_mutate(
            1.0,
            {"composition": 0.33,
             "monoidal": 0.33,
             "trace": 0.5})],
        optm.replacement.replacement_member(),
        optm.evaluator.evaluator_approximation(
            {"x": 10},
            {},
            {"BOUNDARY_x": 1.0},
            {})
       )
ga.run(circuit)
```

The Future

Mutation - Example

- Random path mutation

```
class operator_random_mutate(operator):
  def __init__(self, weight, probs):
  ...
  def arity(self):
    return 1
  def apply(self, members, algo):
    path = RandomPath(members[0].circuit, self.probs, 1)
    random = algo.GenerateRandomCircuit()
    return ReplacePath(members[0].circuit, path, random)
```

- Random rewrite rule

```
subgraph = TracePath(members[0].circuit, path)
rule = RewriteRule("tr(c(c(m(id,#[1]),sum),split))",
                   "conv(exp,t,diff(#[1],t))")
subgraph = Normalise(subgraph, rule)
return ReplacePath(members[0].circuit, path, subgraph)
```

Conclusion

- Nice intuitive, practical tool for prototyping and exploring models
- Shows some promise for future less practical applications