Effects of Functional and Declarative Modeling Frameworks on System Simulation

John Morris, Gregory Mocko, John Wagner

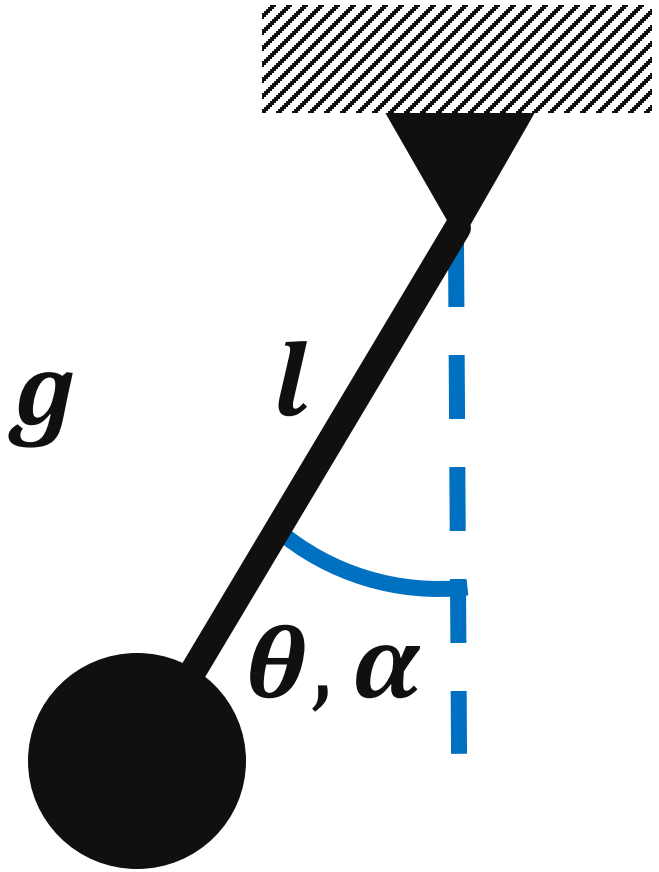
Dept. of Mechanical Engineering
Clemson University
7 Oct 2025

PLMC

CEDAR
Clemson Engineering Design Applications and Research

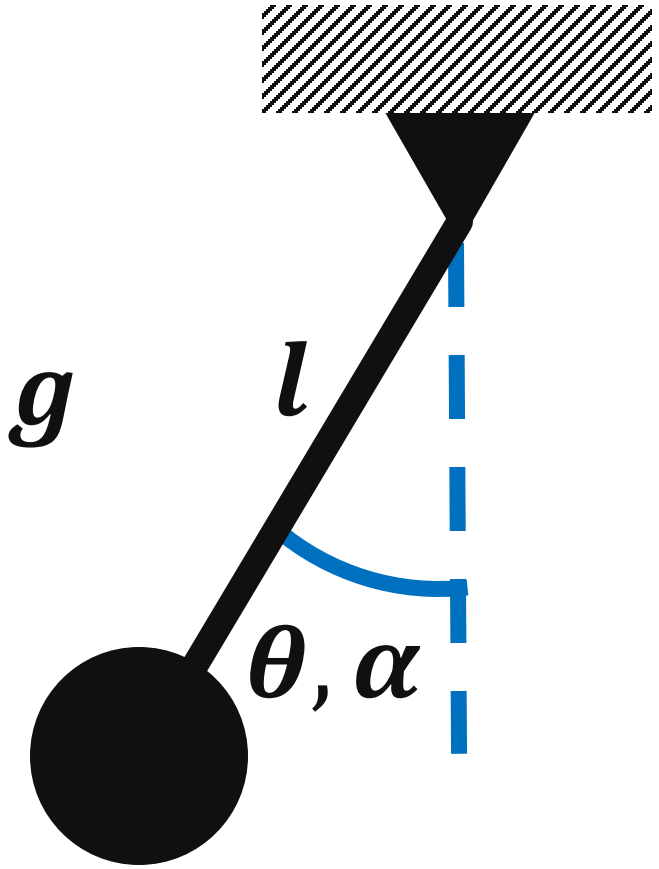
CLEMSON

“A system is a collection of variables” —Ross Ashby



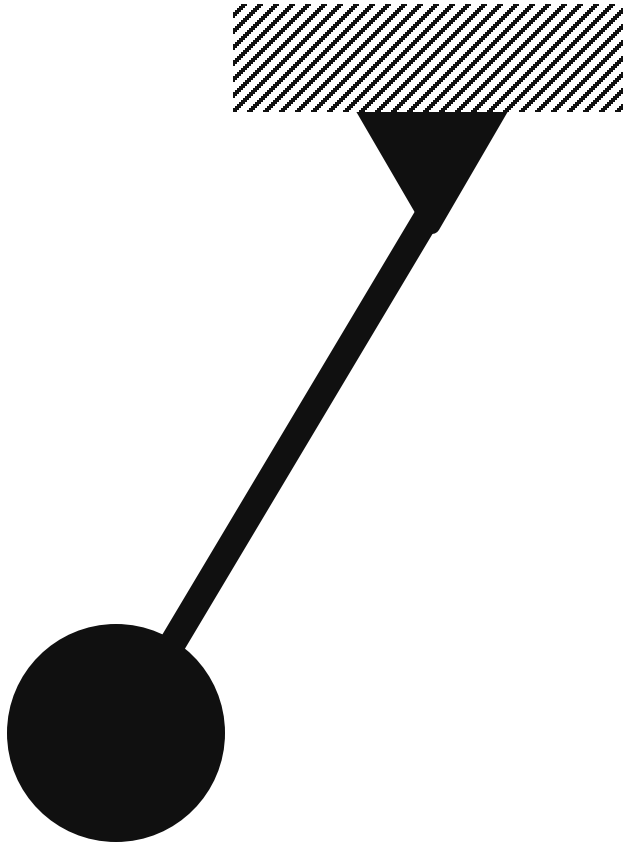
Ashby, W. R. (1956). An Introduction to Cybernetics. Chapman & Hall.

System behavior is given by functions relating these variables



$$\alpha = -\frac{g}{l}\sin(\theta)$$

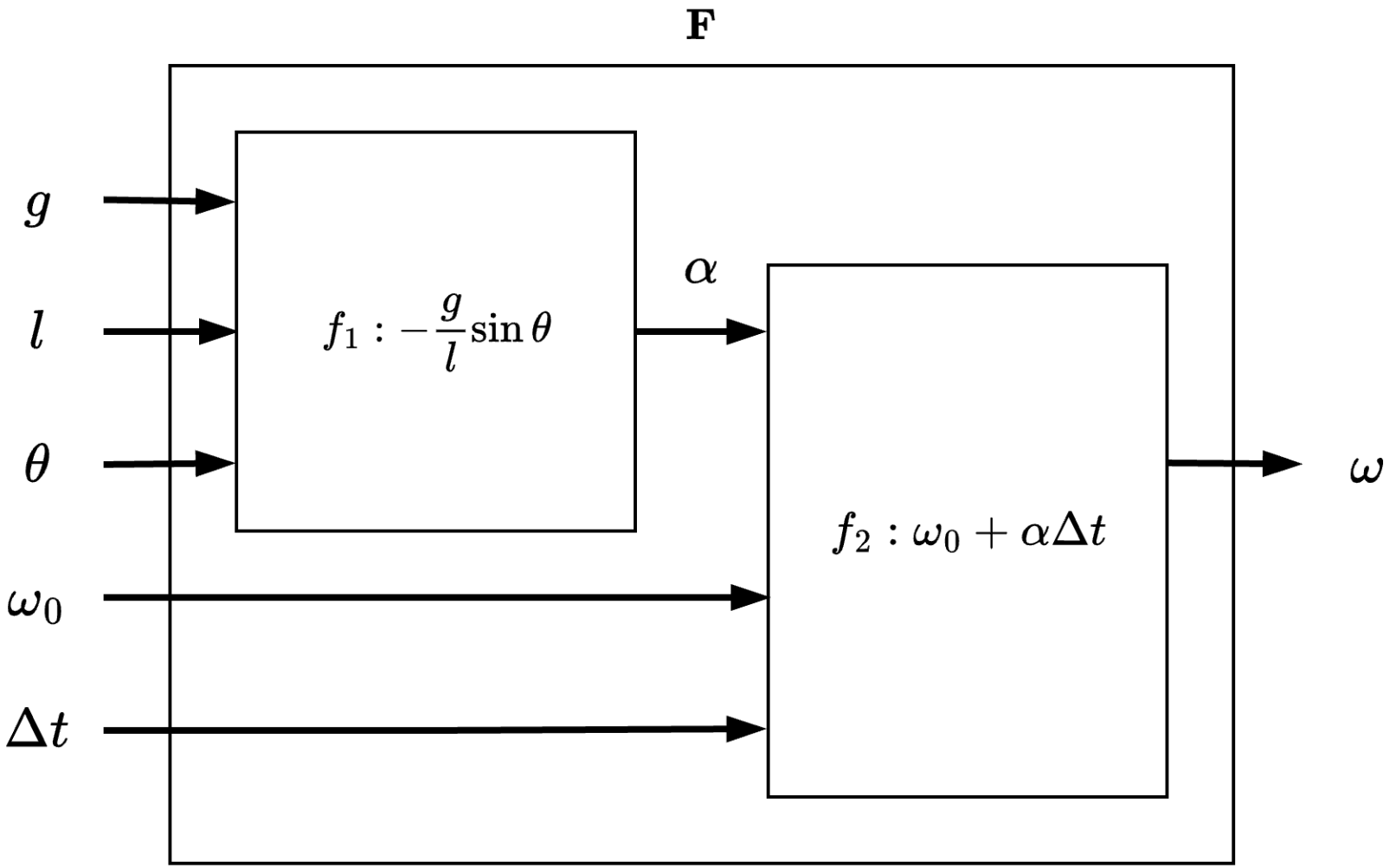
Simulating a model is wiring functions together



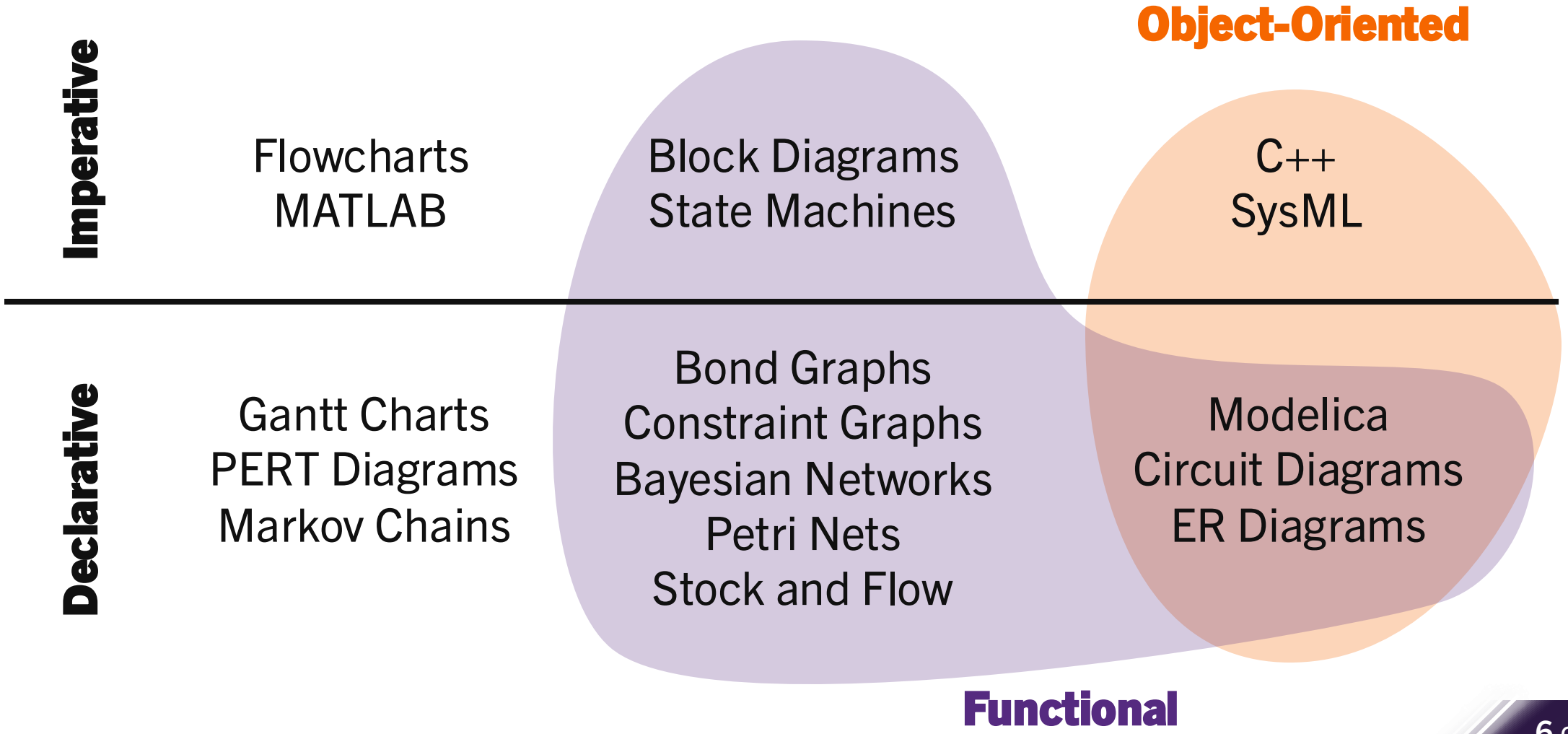
$$\alpha = -\frac{g}{l}\sin(\theta)$$

$$\omega = \int_{\Delta t} \alpha \, dt$$

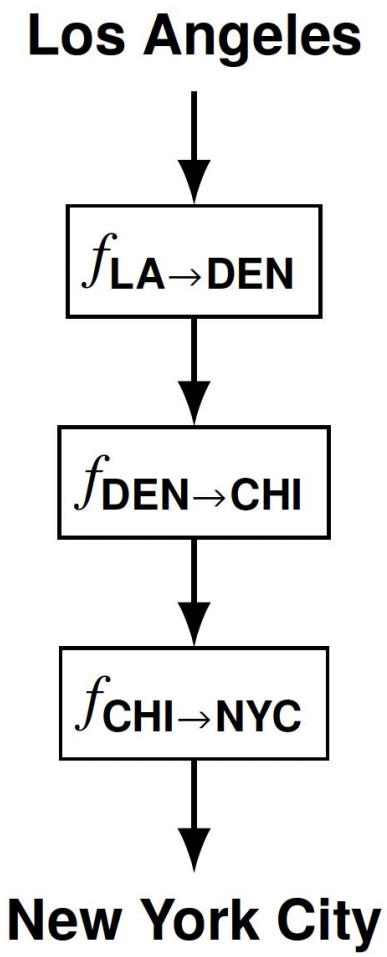
Every program is composed of a series of functions

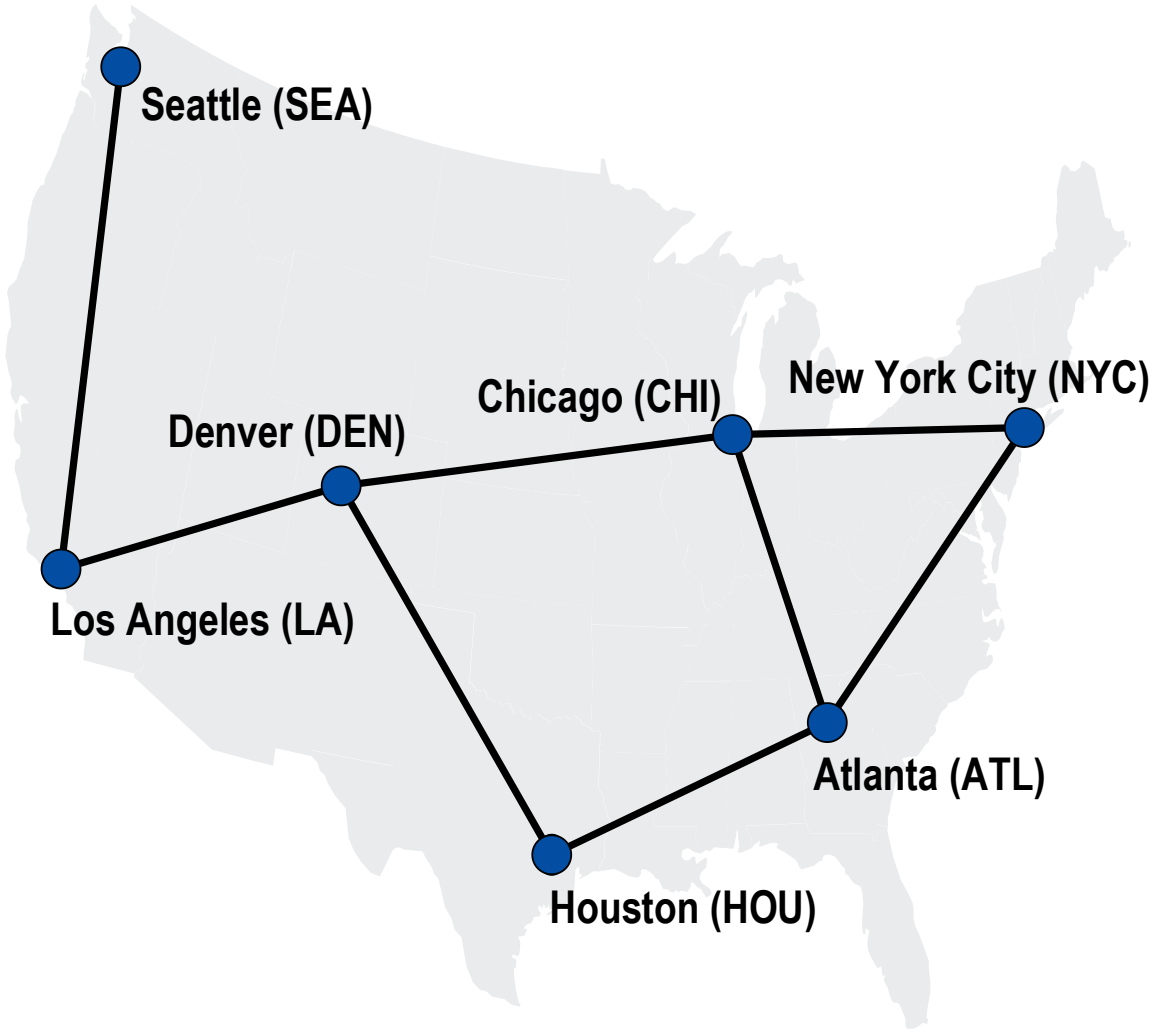


The way functions composed is described by a modeling paradigm



Imperative frameworks are processes, declarative are structures





Paradigm effects compared in a driven, double pendulum

$$f_{\alpha_A} : \{\omega_A, \omega_{A_0}, \Delta t\} \rightarrow \alpha_A = \frac{\omega_A - \omega_{A_0}}{\Delta t}$$

$$f_{\alpha_B} : \{\theta_A, \theta_B, \omega_A, \alpha_A, g\} \rightarrow \alpha_B$$
$$= -\ddot{x}\cos\theta_B - \sin\theta_B\,(\ddot{y} + g)$$

where:

$$\ddot{x} = \alpha_A \cos\theta_A - \omega_A^2 \sin\theta_A \text{and}$$

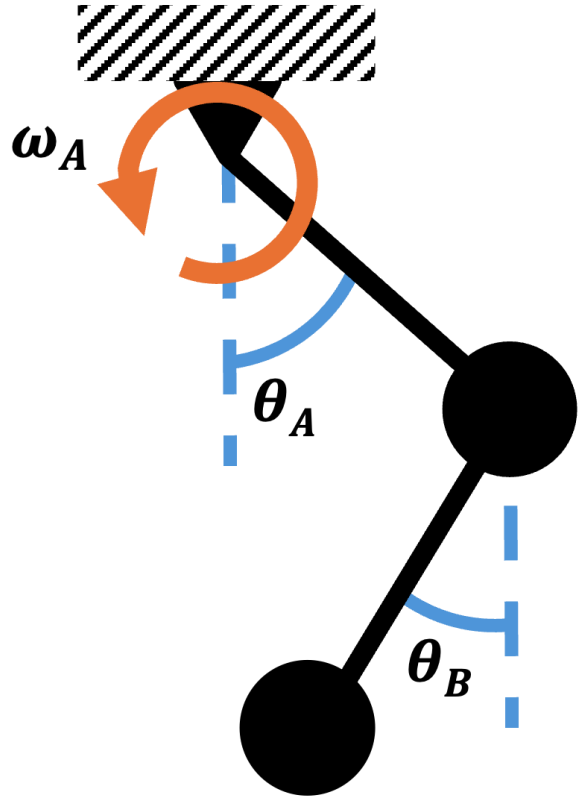
$$\ddot{y} = \alpha_A \sin\theta_A + \omega_A^2 \cos\theta_A$$

$$f_{\omega_A} : \{t\} \rightarrow \omega_A = \begin{cases} -\frac{\pi}{4} & \text{if } t \,\%\, 4 < 2 \\ \frac{\pi}{4} & \text{otherwise} \end{cases}$$

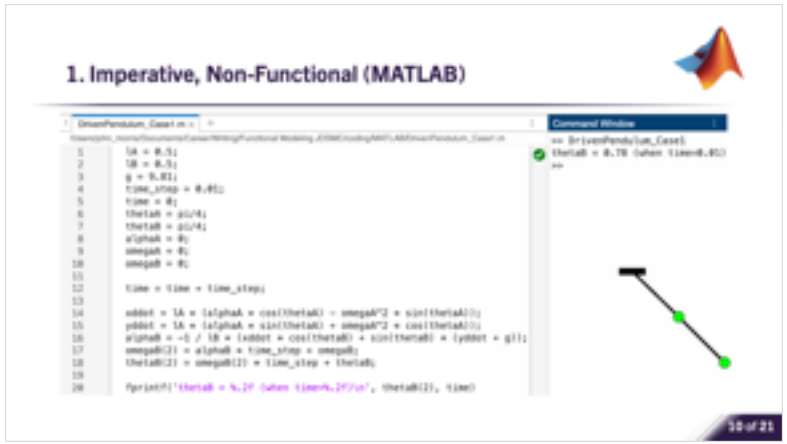
$$f_{\omega_B} : \{\alpha_B, \omega_{B_0}, \Delta t\} \rightarrow \omega_B = \alpha_B\,(\Delta t + \omega_{B_0})$$

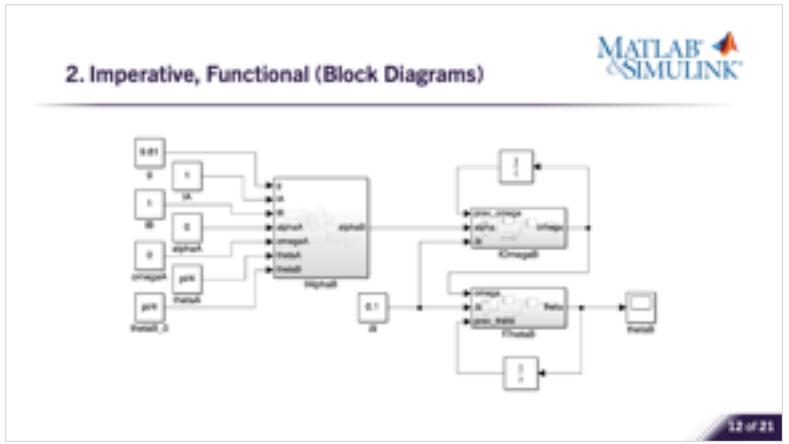
$$f_{\theta_A} : \{\omega_A, \theta_{A_0}, \Delta t\} \rightarrow \theta_A = \omega_A\,(\Delta t + \theta_{A_0})$$

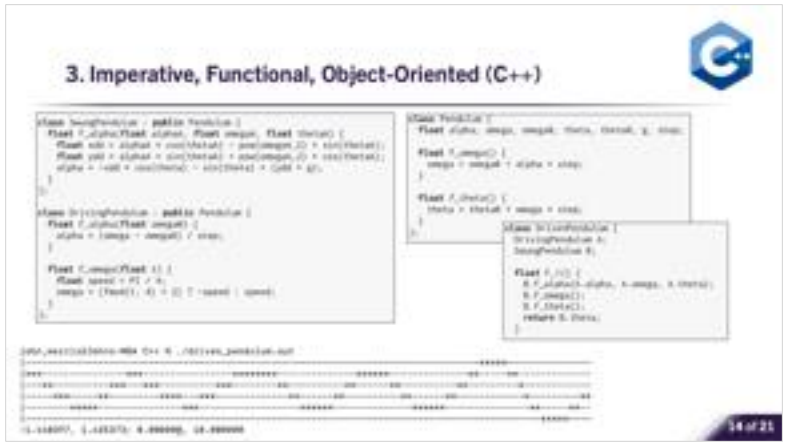
$$f_{\theta_B} : \{\omega_B, \theta_{B_0}, \Delta t\} \rightarrow \theta_B = \omega_B\,(\Delta t + \theta_{B_0})$$

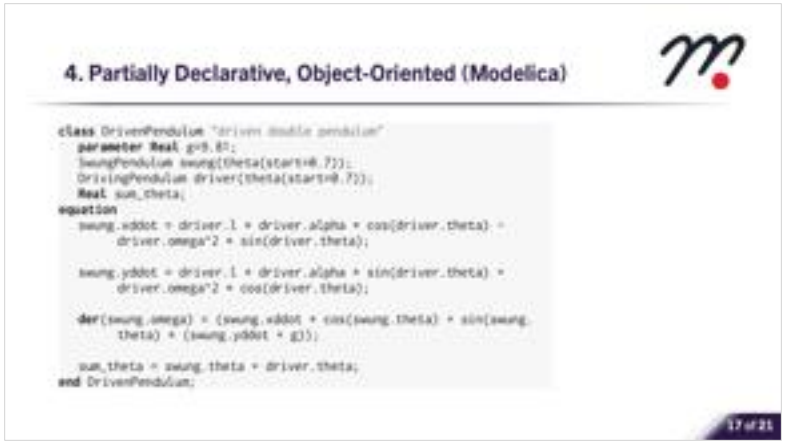


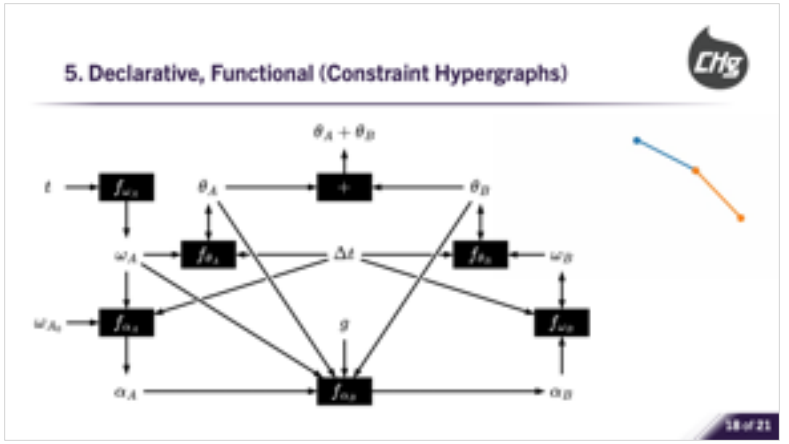
Simulation Comparison Cases



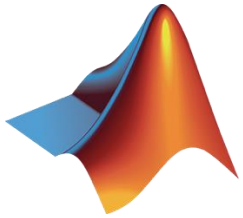








1. Imperative, Non-Functional (MATLAB)



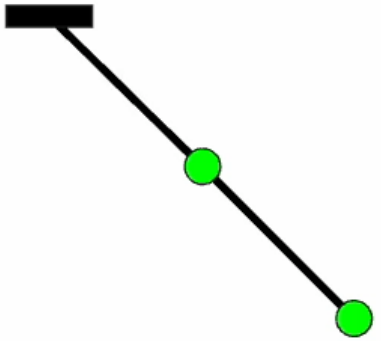
DrivenPendulum_Case1.m

/Users/john_morris/Documents/Career/Writing/Functional Modeling JDSMC/coding/MATLAB/DrivenPendulum_Case1.m

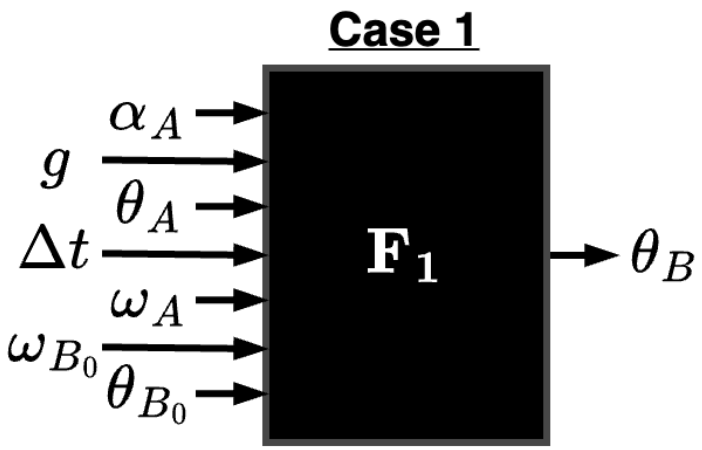
```
1   lA = 0.5;
2   lB = 0.5;
3   g = 9.81;
4   time_step = 0.01;
5   time = 0;
6   thetaA = pi/4;
7   thetaB = pi/4;
8   alphaA = 0;
9   omegaA = 0;
10  omegaB = 0;
11
12  time = time + time_step;
13
14  xddot = lA * (alphaA * cos(thetaA) - omegaA^2 * sin(thetaA));
15  yddot = lA * (alphaA * sin(thetaA) + omegaA^2 * cos(thetaA));
16  alphaB = -1 / lB * (xddot * cos(thetaB) + sin(thetaB) * (yddot + g));
17  omegaB(2) = alphaB * time_step + omegaB;
18  thetaB(2) = omegaB(2) * time_step + thetaB;
19
20  fprintf('thetaB = %.2f (when time=%.2f)\n', thetaB(2), time)
```

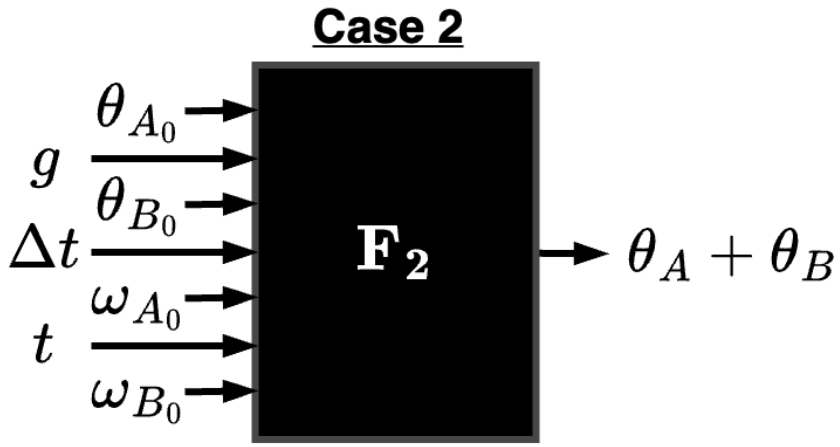
Command Window

```
>> DrivenPendulum_Case1
thetaB = 0.78 (when time=0.01)
>>
```

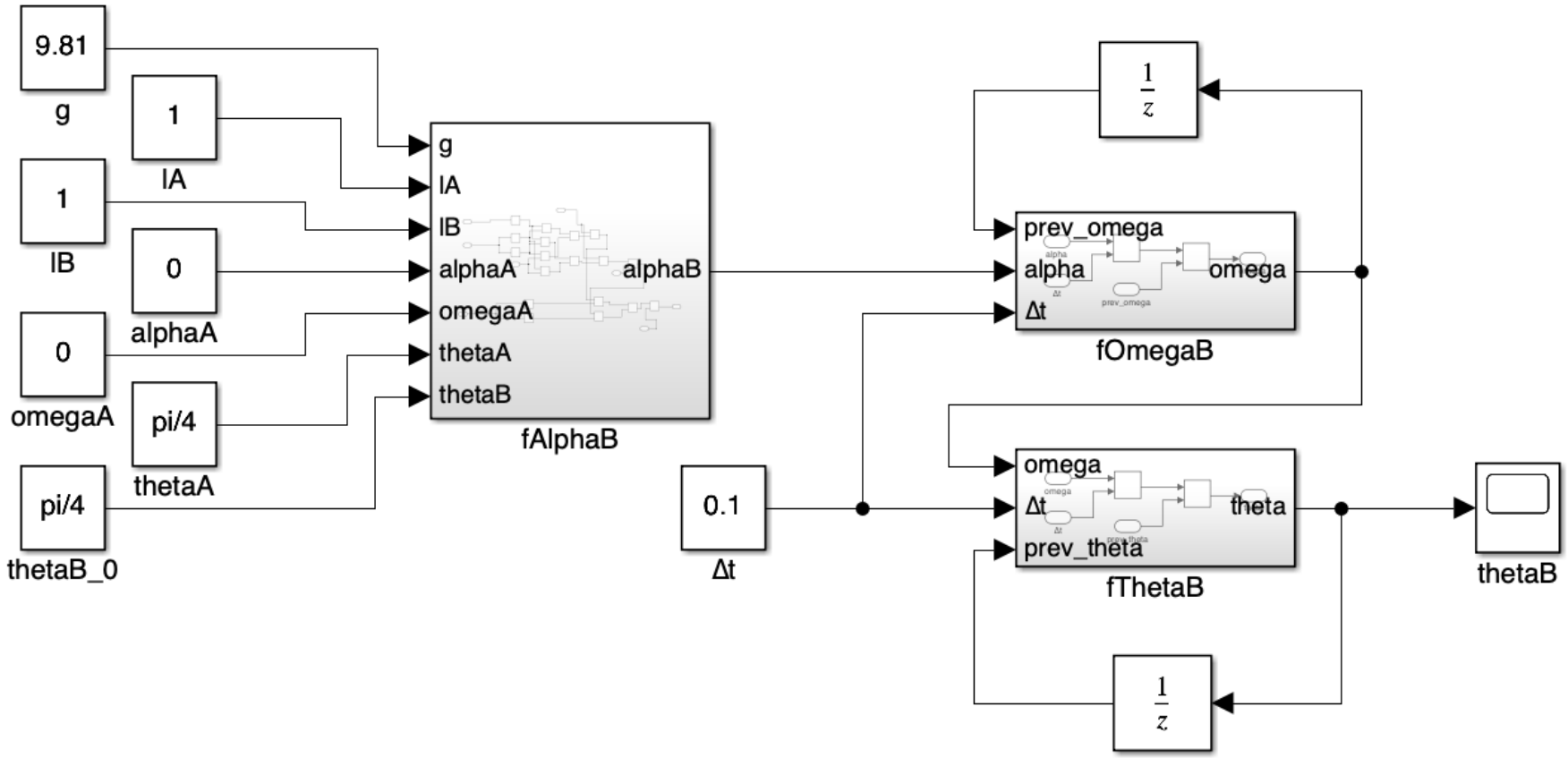


Non-functional imperative programs are fully abstracted

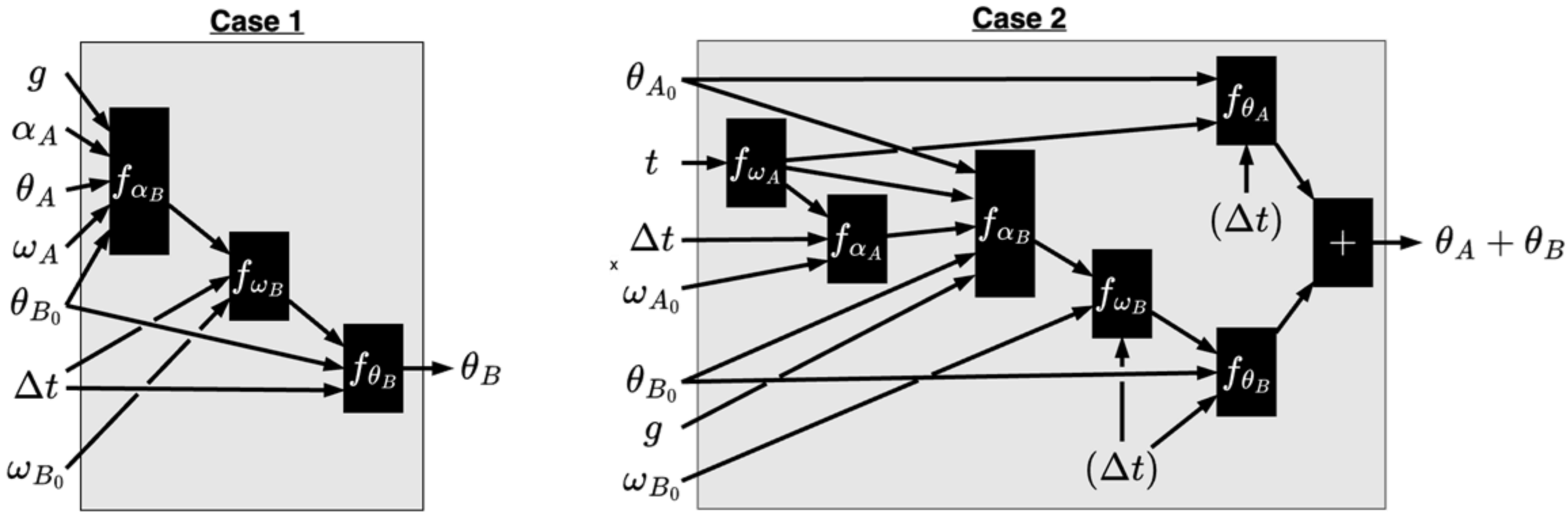




2. Imperative, Functional (Block Diagrams)



Imperative functions enforce a static wiring scheme



3. Imperative, Functional, Object-Oriented (C++)

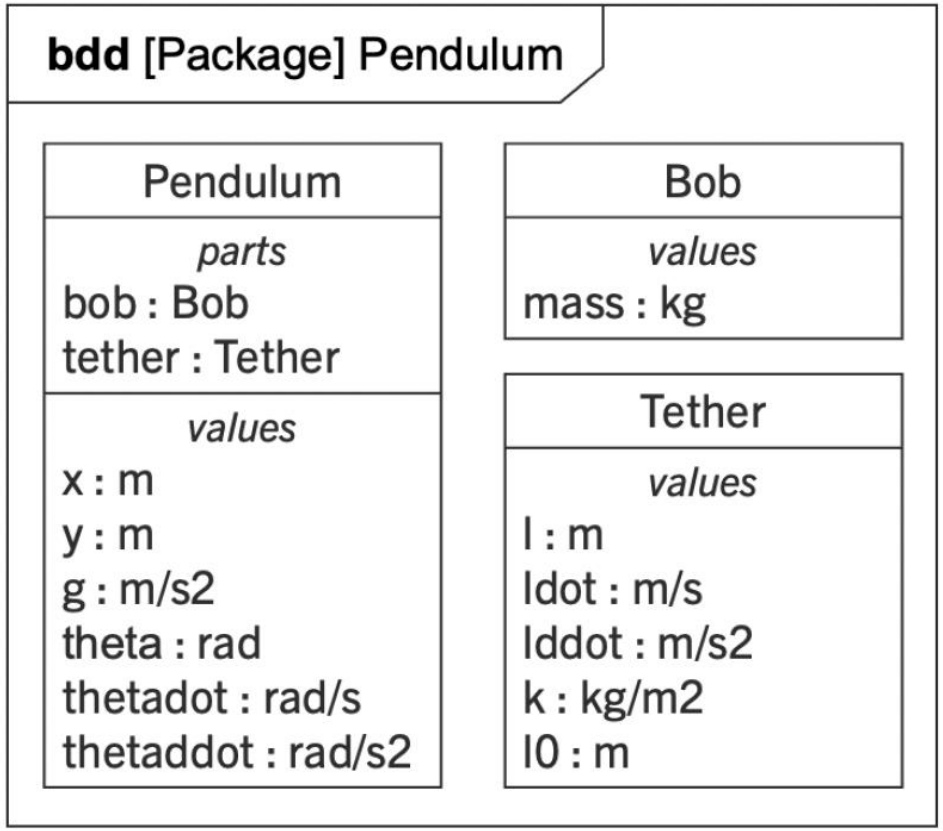
```
class SwungPendulum : public Pendulum {
  float f_alpha(float alphaA, float omegaA, float thetaA) {
    float xdd = alphaA * cos(thetaA) - pow(omegaA,2) * sin(thetaA);
    float ydd = alphaA * sin(thetaA) + pow(omegaA,2) * cos(thetaA);
    alpha = -xdd * cos(theta) - sin(theta) * (ydd + g);
  }
};

class DrivingPendulum : public Pendulum {
  float f_alpha(float omega0) {
    alpha = (omega - omega0) / step;
  }

  float f_omega(float t) {
    float speed = PI / 4;
    omega = (fmod(t, 4) < 2) ? -speed : speed;
  }
};
```

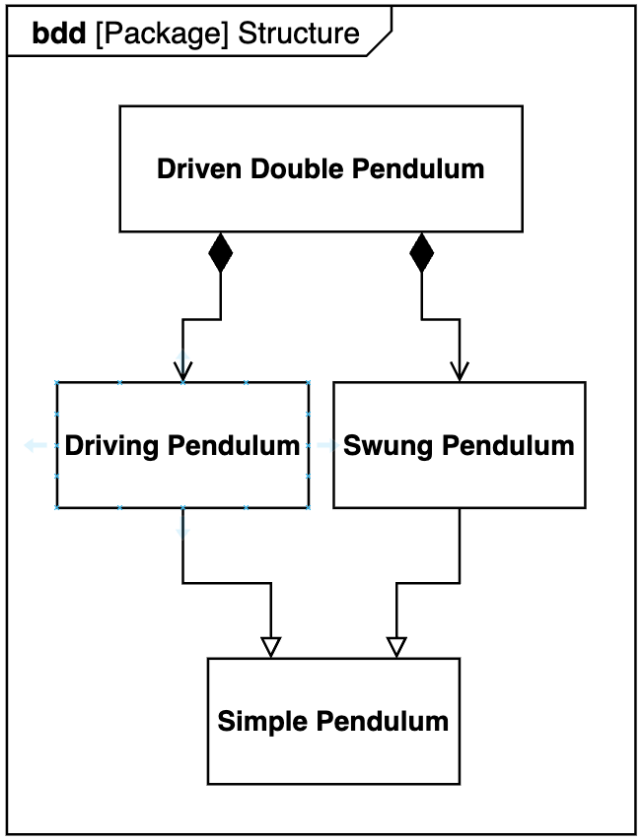
```
class Pendulum {
  float alpha, omega, omega0, theta, theta0, g, step;

  float f_omega() {
    omega = omega0 + alpha * step;
  }

  float f_theta() {
    theta = theta0 + omega * step;
  }
};
```

```
class DrivenPendulum {
  DrivingPendulum A;
  SwungPendulum B;

  float F_1() {
    B.f_alpha(A.alpha, A.omega, A.theta);
    B.f_omega();
    B.f_theta();
    return B.theta;
  }
```

```
john_morris@Johns-MBA C++ % ./driven_pendulum.out
|-----------------------------------------------------------------------------*****-------------
|***-------------***-------------*******-----------******-----------**-----**-----------
|---**----------***---***---------***-------**---------**------**---------**--------*----------
|-----***-----**--------****---***----------**------**---------**------**----------*--------**
|-------*****-------------***----------------******-----------******-------------**-----**--
|-----------------------------------------------------------------------------*****----
-1.440397, 1.435373; 0.000000, 10.000000
```

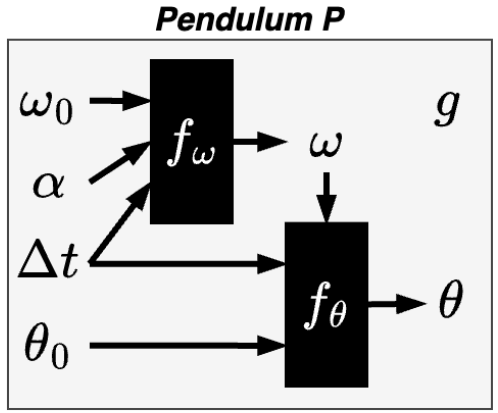
OO frameworks encapsulate behaviors into static blocks

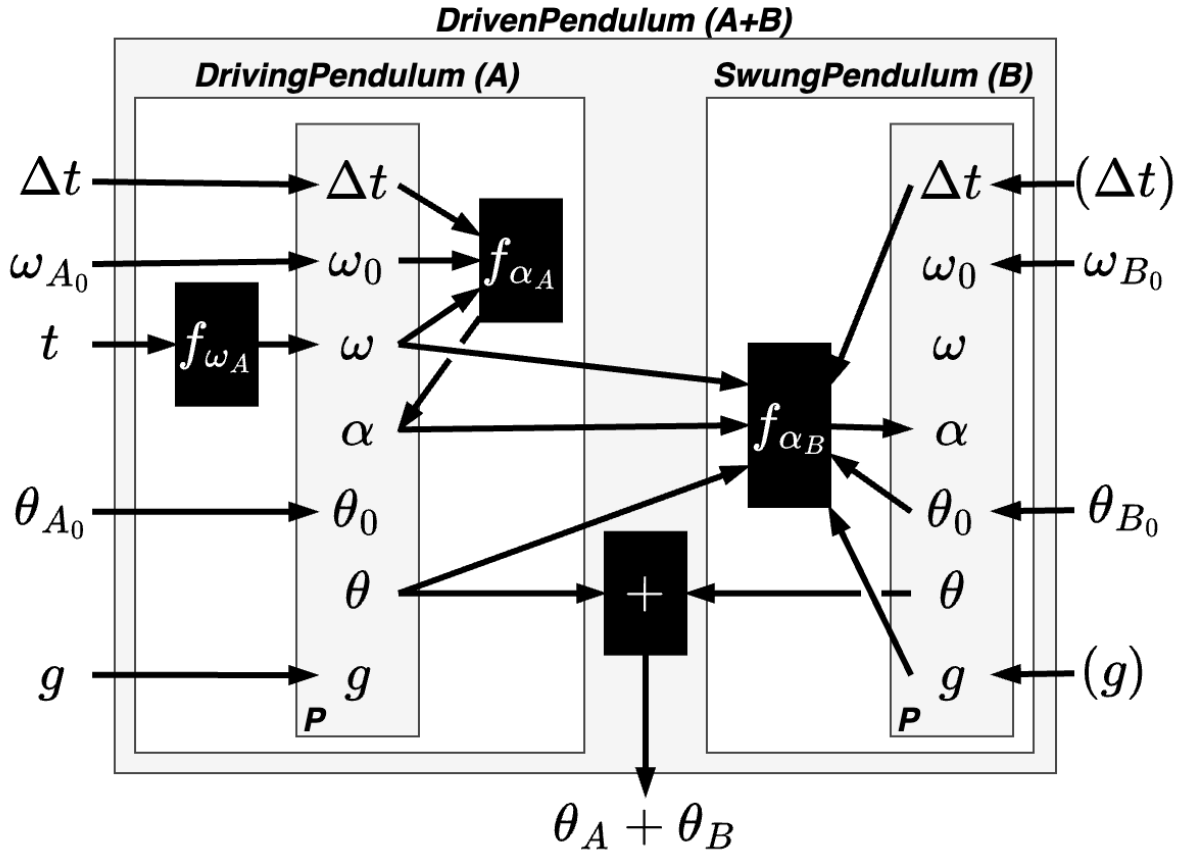




Inherited objects must maintain imperative behavioral processes

Functional

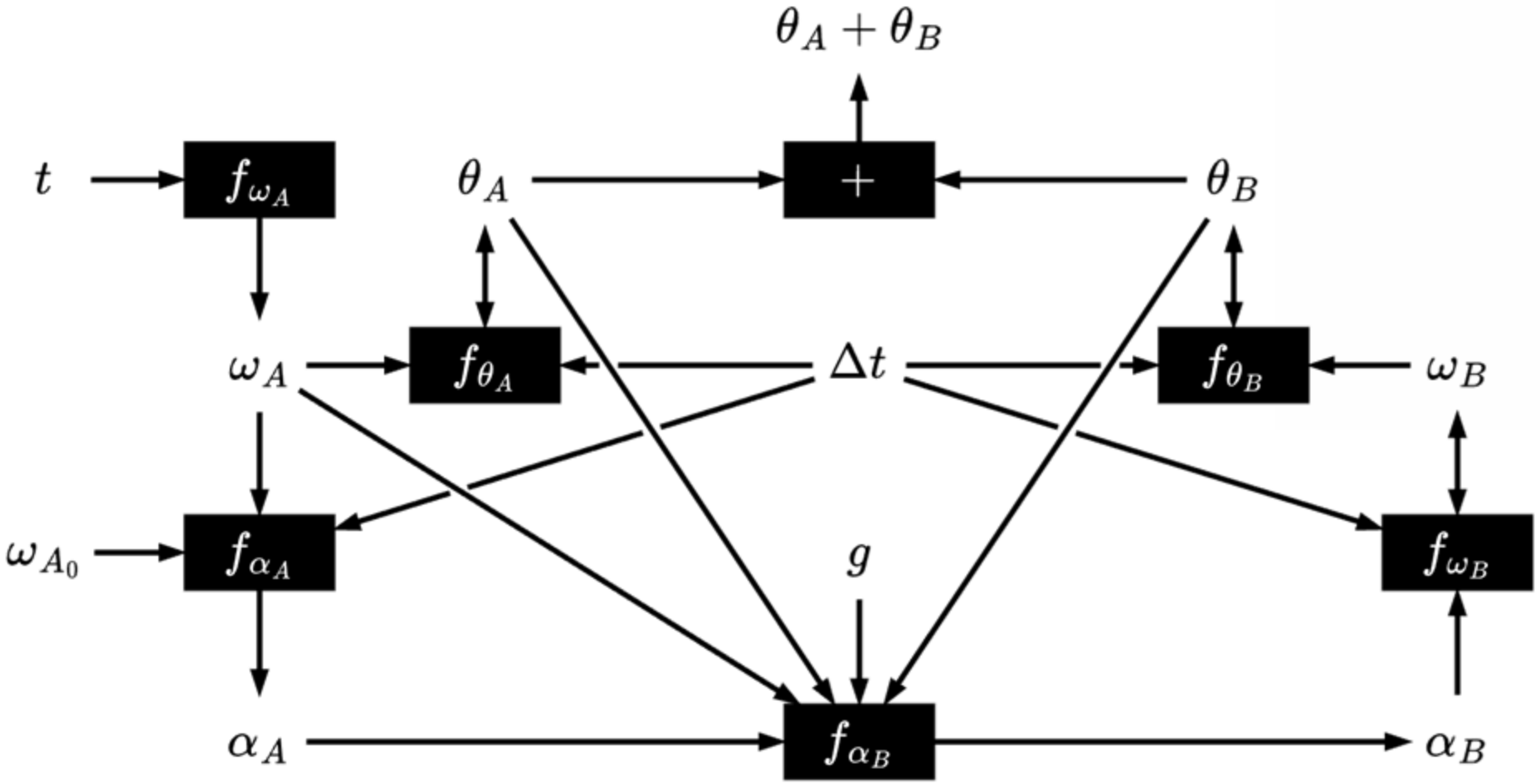


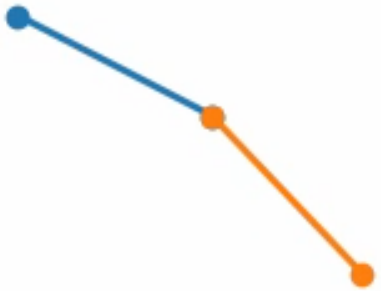


4. Partially Declarative, Object-Oriented (Modelica)

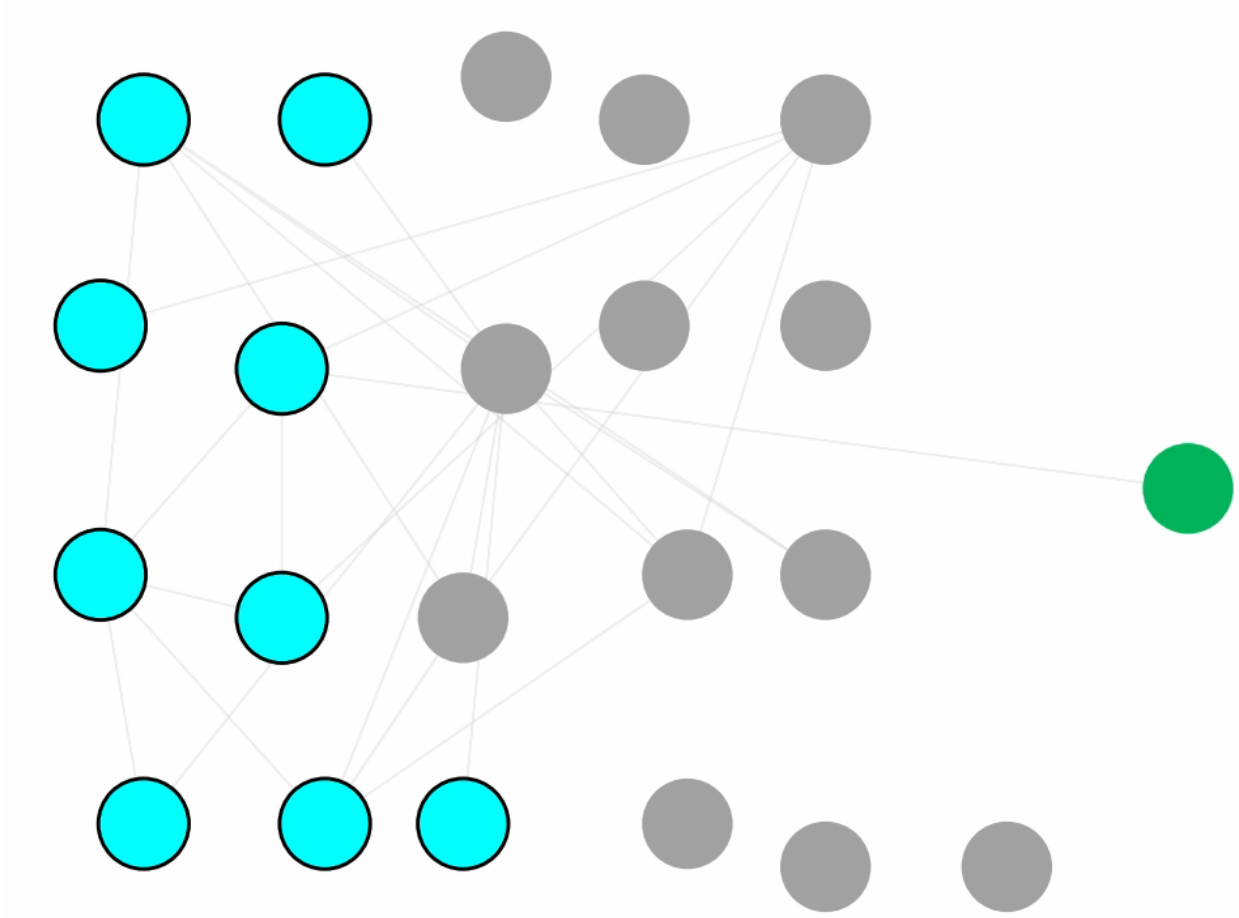
```
class DrivenPendulum "driven double pendulum"
   parameter Real g=9.81;
   SwungPendulum swung(theta(start=0.7));
   DrivingPendulum driver(theta(start=0.7));
   Real sum_theta;
equation
   swung.xddot = driver.l * driver.alpha * cos(driver.theta) -
         driver.omega^2 * sin(driver.theta);

   swung.yddot = driver.l * driver.alpha * sin(driver.theta) +
         driver.omega^2 * cos(driver.theta);

   der(swung.omega) = (swung.xddot * cos(swung.theta) + sin(swung.
         theta) * (swung.yddot + g));

   sum_theta = swung.theta + driver.theta;
end DrivenPendulum;
```

5. Declarative, Functional (Constraint Hypergraphs)





Declarative simulation requires a constructing agent



ConstraintHg

constrainthg 0.2.3

`pip install constrainthg`

Latest version

Released: Jun 25, 2025

Kernel for building and simulating constraint hypergraphs.

Navigation

- Project description
- Release history
- Download files

Verified details

These details have been verified by PyPI

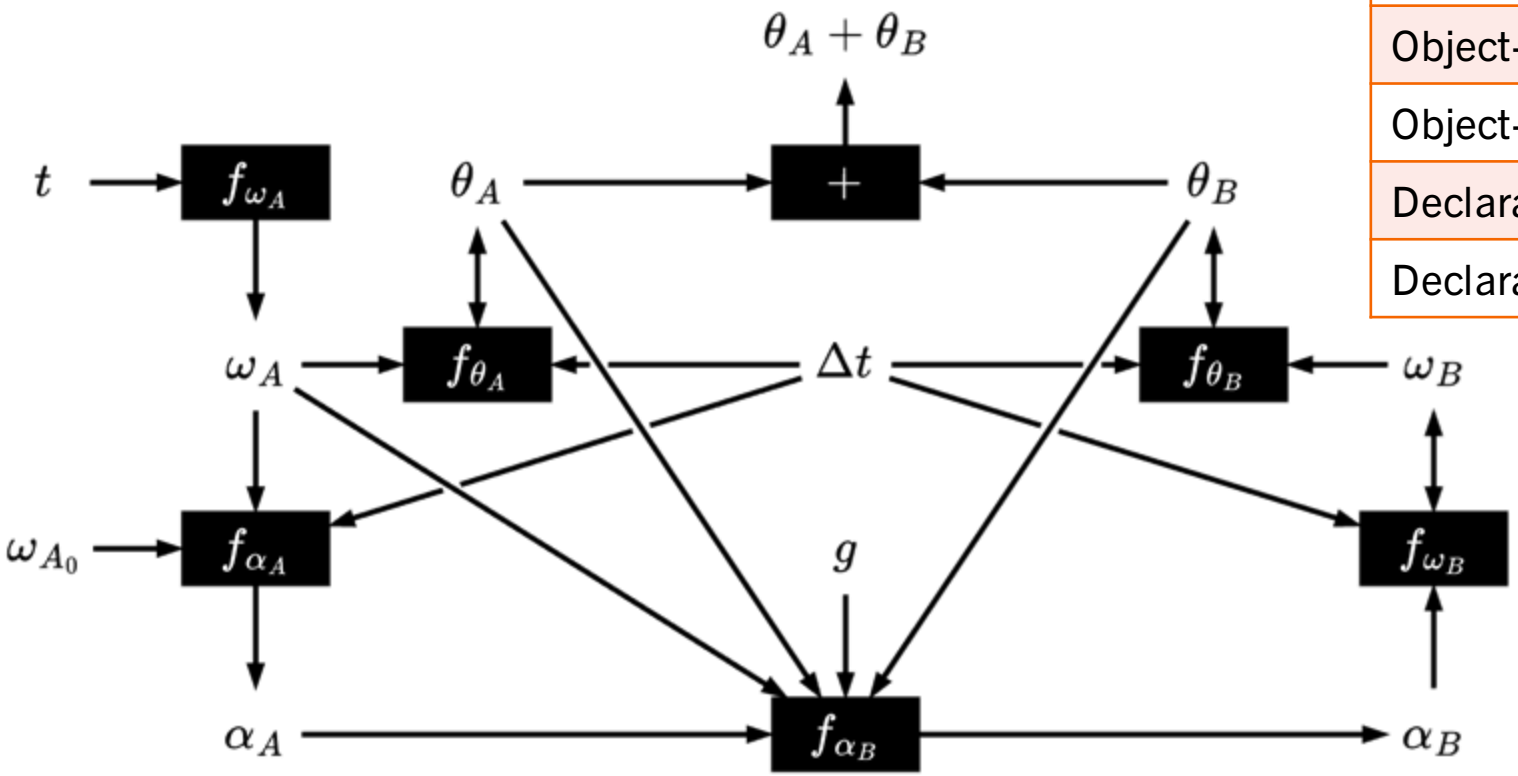
Project description

ConstraintHg

DOI 10.5281/zenodo.15741546 | docs passing | tests 26/26 | release v0.2.3 | last commit july

ConstraintHg is a systems modeling kernel written in Python that enables general definition and universal simulation of any system. The kernel breaks a system down into the informational values (nodes) and functional relationships (hyperedges), providing robust simulation through pathfinding operations. This repository is under active

Functional, declarative paradigms maximize a model's interpretability



Paradigm	Num. Simulations
Strictly imperative	1
Imperative-functional	1
Object-oriented-functional	6
Object-oriented-non-functional	4
Declarative-object-oriented	6
Declarative-functional	25

More information available at our documentation

Please reach out to jhmrrs@clemson.edu

PLMC

CEDAR
Clemson Engineering Design Applications and Research

CLEMSON