Continuous Models (state trajectory: $\mathbb{R} \rightarrow \mathbb{R}^n$): Ordinary Differential Equation (ODE) formalism

$$\frac{d^n x}{dt^n} = f(\frac{d^{n-1} x}{dt^{n-1}}, \ldots, x, u, t)$$

f and $x(t)$ may be vectors

$$\left\{ \begin{array}{rcl} x & = & x_0 \\ \frac{dx}{dt} & = & x_1 \\ \frac{dx_1}{dt} & = & x_2 \\ & \ldots & \\ \frac{dx_{n-1}}{dt} & = & x_n = f(x_{n-1}, x_{n-2}, \ldots, x_1, x_0, u, t) \end{array} \right.$$

Euler discretisation

$$\begin{aligned}
\frac{dx}{dt} &= f(x,u,t) \\
&\Downarrow \\
\frac{x(t_i+\Delta t)-x(t_i)}{\Delta t} &\cong f(x(t_i),u(t_i),t_i) \\
&\Downarrow \\
x(t_i+\Delta t) &\cong x(t_i)+\Delta t\, f(x(t_i),u(t_i),t_i)
\end{aligned}$$

Taylor Series Expansion

$$x(t_i + \Delta t) = x(t_i) + \frac{\Delta t}{1!}\frac{dx}{dt}|_{t_i} + \frac{\Delta t^2}{2!}\frac{d^2x}{dt^2}|_{t_i} + \frac{\Delta t^3}{3!}\frac{d^3x}{dt^3}|_{t_i} + \ldots$$

Is EXACT if expansion is not truncated

ERROR $= O(\Delta t^{N+1})$ if truncated after term N

Beyond first derivative, use difference formulas

ERROR $\varepsilon_N \cong approx_{N+1} - approx_N$

Integration Methods: Euler

Single-step

$$x_0 = \alpha_0$$
$$x_{i+1} = x_i + \Delta t f(t_i, x_i),\ i \geq 0$$

Unsymmetrical: uses only derivative in begin point.

Integration Methods: Modified Euler

Single-step

$$
\begin{aligned}
&x_0 = \alpha_0\\
&k_1 = \Delta t\, f(t_i, x_i)\\
&k_2 = \Delta t\, f(t_i + \Delta t, x_i + k_1)\\
&x_{i+1} = x_i + \frac{k_1}{2} + \frac{k_2}{2},\ i \geq 0
\end{aligned}
$$

Symmetrical: uses derivative in begin and end point.

Integration Methods: Midpoint

Single-step

$$
\begin{aligned}
&x_0 = \alpha_0 \\
&k_1 = \Delta t f(t_i, x_i) \\
&k_2 = \Delta t f(t_i + \tfrac{\Delta t}{2}, x_i + \tfrac{k_1}{2}) \\
&x_{i+1} = x_i + k_2,\ i \geq 0
\end{aligned}
$$

Symmetrical: halfway point.

Integration Methods: Heun

Single-step

$$
\begin{aligned}
&x_0 = \alpha_0 \\
&k_1 = \Delta t\, f(t_i, x_i) \\
&k_2 = \Delta t\, f\left(t_i + \frac{2\Delta t}{3}, x_i + \frac{2k_1}{3}\right) \\
&x_{i+1} = x_i + \frac{k_1}{4} + \frac{3k_2}{4},\ i \geq 0
\end{aligned}
$$

Integration Methods: Runge-Kutta Methods

Single-step, q stages (function evaluations per step)

$$x_{i+1} = x_i + \Delta t \phi(t_i, x_i; \Delta t)$$

$$\phi(t_i, x_i; \Delta t) = \sum_{i=1}^{q} \omega_i k_i$$

Explicit method:

$$k_i = f(t_i + \Delta t \alpha_i, x_i + \Delta t \sum_{j=1}^{i-1} \beta_{ij} k_j),\ \alpha_1 = 0$$

Implicit method:

$$k_i = f(t_i + \Delta t \alpha_i, x_i + \Delta t \sum_{j=1}^{q} \beta_{ij} k_j)$$

- nonlinear set of equations in k_i
- explicit is a special case

Order p: *exact* solution up to polynomial of order p
$\Rightarrow$ can determine $\alpha_i, \omega_i, \beta_{ij}$

For explicit method of order p, at least $q_{min}(p)$ stages are required:

p	1	2	3	4	5	6	7	...
q_min(p)	1	2	3	4	6	7	9	...

Integration Methods: Runge-Kutta 4

Single-step

$$x_0 = \alpha_0$$
$$k_1 = \Delta t f(t_i, x_i)$$
$$k_2 = \Delta t f(t_i + \tfrac{\Delta t}{2}, x_i + \tfrac{k_1}{2})$$
$$k_3 = \Delta t f(t_i + \tfrac{\Delta t}{2}, x_i + \tfrac{k_2}{2})$$
$$k_4 = \Delta t f(t_i + \Delta t, x_i + k3)$$
$$x_{i+1} = x_i + \tfrac{k_1}{6} + \tfrac{k_2}{3} + \tfrac{k_3}{3} + \tfrac{k_4}{6}, i \geq 0$$

Integration Methods: Adams-Bashforth

Multi-step: need lower-step methods for *start-up*

2-step

$x_0 = \alpha_0$

$x_1 = \alpha_1$

$x_{i+1} = x_i + \frac{\Delta t}{2}(3f(t_i, x_i) - f(t_{i-1}, x_{i-1}),\ i \geq 1$

3-step

$x_0 = \alpha_0$

$x_1 = \alpha_1$

$x_2 = \alpha_2$

$x_{i+1} = x_i + \frac{\Delta t}{12}(23f(t_i, x_i) - 16f(t_{i-1}, x_{i+1}) + 5f(t_{i-2}, x_{i-2})),\ i \geq 2$

4-step

$x_0 = \alpha_0$

$x_1 = \alpha_1$

$x_2 = \alpha_2$

$x_3 = \alpha_3$

$$x_{i+1} =$$
$$x_i + \frac{\Delta t}{24}(55f(t_i, x_i) - 59f(t_{i-1}, x_{i+1}) + 37f(t_{i-2}, x_{i-2}) - 9f(t_{i-3}, x_{i-3})),\ i \geq 3$$

Integration Methods: Milne

Predictor-corrector

- Predictor

  $x_0 = \alpha_0$

  $x_1 = \alpha_1$

  $x_2 = \alpha_2$

  $x_3 = \alpha_3$

  $x_{i+1}^{(0)} = x_{i-3} + \frac{4\Delta t}{3}(2f(t_i, x_i) - f(t_{i-1}, x_{i-1}) + 2f(t_{i-2}, x_{i-2})), i \geq 3$

- Corrector

  $x_{i+1}^{(k+1)} = x_{i-1} + \frac{\Delta t}{3}(f(t_{i+1}, x_{i+1}^{(k)}) + 4f(t_i, x_i) + f(t_{i-1}, x_{i-1})),$

  $i \geq 2, k = 1, 2, \ldots$

Adaptive Step-size Control

- want to attain pre-set *minimum (step-wise) accuracy*
- want *mimimum computation*

Solution: use accuracy estimate to *double/halve step-size*

Obtaining accuracy estimate:

1. step doubling (*e.g.,* RK4)
2. $\varepsilon_N \cong approx_{N+1} - approx_N$

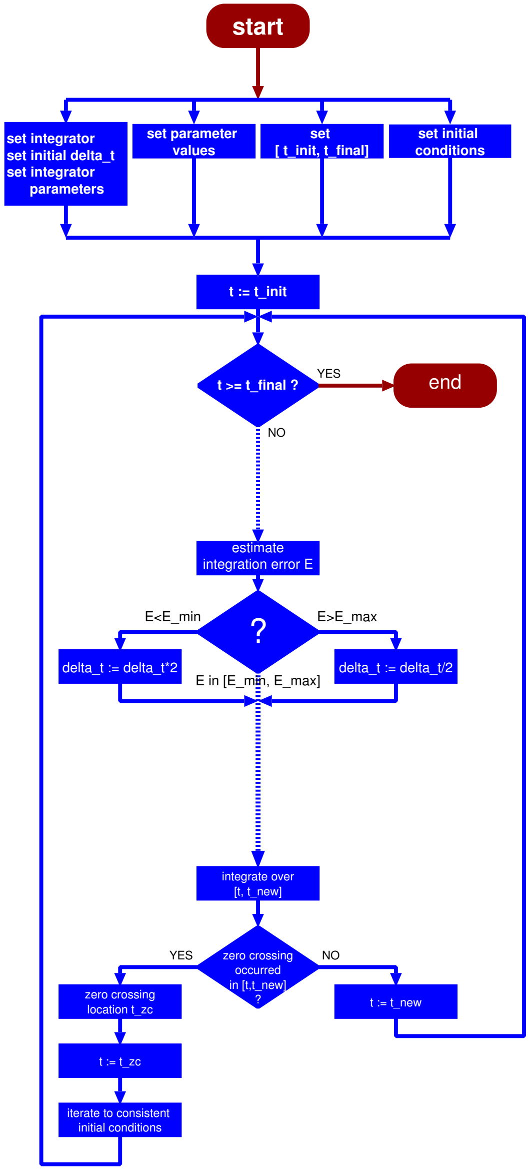
Adaptive Step-size Control

RK4 + RK5 → Runge-Kutta Fehlberg (embedded)

$$k_1 = \Delta t f(t_i, x_i)$$
$$k_2 = \Delta t f(t_i + a_2 \Delta t, x_i + b_{12} k_1)$$
$$\ldots$$
$$k_6 = \Delta t f(t_i + a_6 \Delta t, x_i + b_{61} k_1 + b_{62} k_2 + \cdots + b_{65} k_5)$$

$$x_{i+1} = x_i + c_1 k_1 + c_2 k_2 + \cdots + c_6 k_6 + O(\Delta t^6)$$
$$x^*_{i+1} = x_i + c^*_1 k_1 + c^*_2 k_2 + \cdots + c^*_6 k_6 + O(\Delta t^5)$$

$$\varepsilon_{estim} = x_{i+1} - x^*_{i+1} = \sum_{i=1}^{6} (c_i - c^*_i) k_i$$


start
set integrator
set initial delta_t
set integrator
parameters
set parameter
values
set
[t_init, t_final]
set initial
conditions
t := t_init
t >= t_final ?
YES
end
NO
estimate
integration error E
E<E_min
?
E>E_max
delta_t := delta_t*2
E in [E_min, E_max]
delta_t := delta_t/2
integrate over
[t, t_new]
YES
zero crossing
occurred
in [t,t_new]
?
NO
zero crossing
location t_zc
t := t_new
t := t_zc
iterate to consistent
initial conditions

Stiff Systems

$$
\begin{aligned}
u' &= 998u + 1998v \\
v' &= -999u - 1999v \\
u(0) &= 1, v(0) = 0
\end{aligned}
$$

$$
\begin{aligned}
u &= 2y - z \\
v &= -y + z
\end{aligned}
$$

$$
\begin{aligned}
u &= 2e^{-t} - e^{-1000t} \\
v &= -e^{-t} + e^{-1000t}
\end{aligned}
$$

Stiff Systems: solvers

$x' = -cx$

- Explicit: Forward Euler: $x_{i+1} = x_i + \Delta t x'_i$
  $x_{i+1} = (1 - c\Delta t)x_i$

- Implicit: Backward Euler: $x_{i+1} = x_i + \Delta t x'_{i+1}$
  $x_{i+1} = \frac{x_i}{1+c\Delta t}$
  Rosenbrock, Gear, ... methods

Differential Algebraic Equations (DAE)

$$\begin{array}{rcl} f(\frac{d^n x}{dt^n}, \frac{d^{n-1} x}{dt_{n-1}}, \ldots, x, u, t) & = & 0 \\ g(x,t) & = & 0 \end{array}$$

Residual Solvers

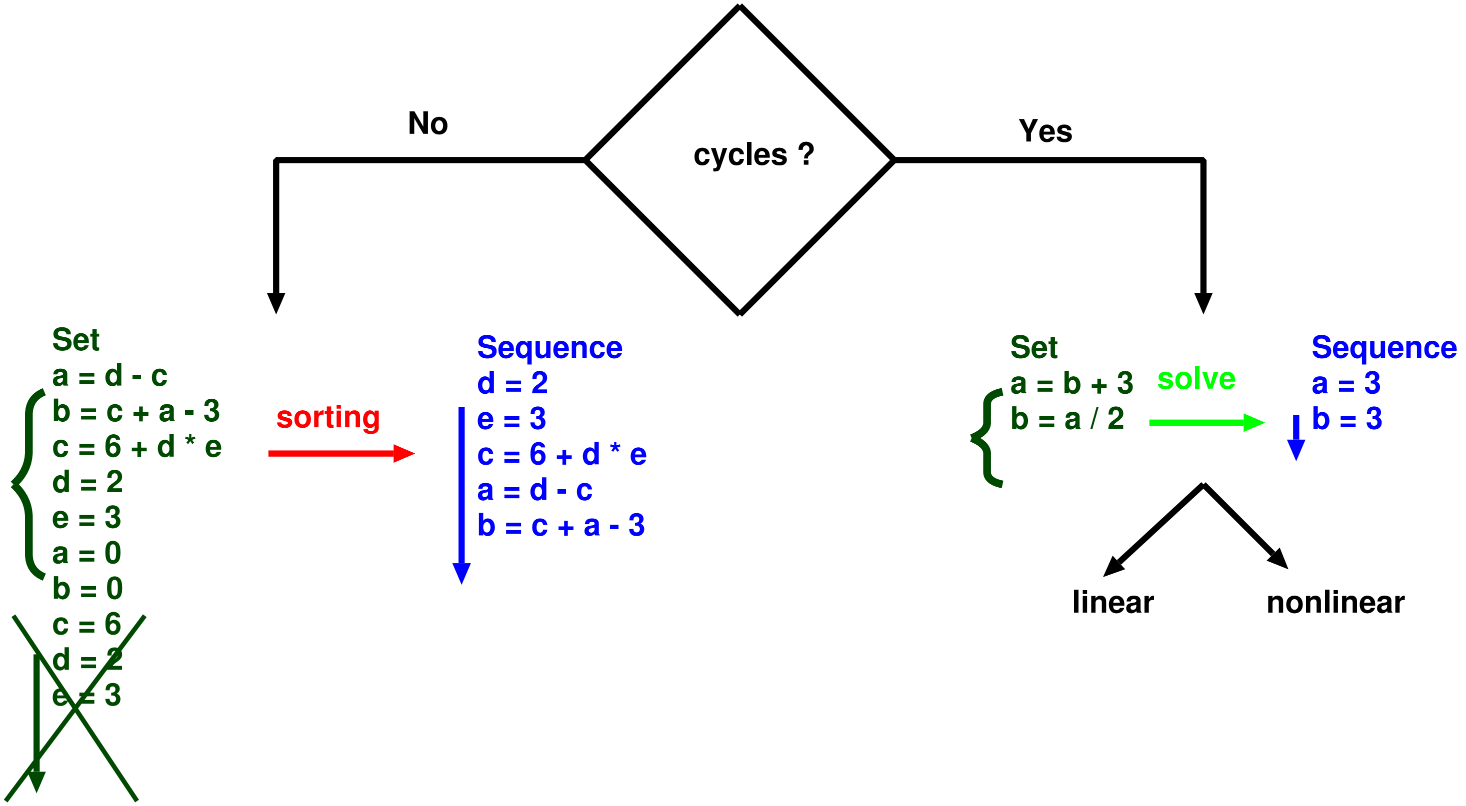
DASSL (Petzold)

`http://www.engineering.ucsb.edu/ cse/software.html`

Causal continuous-time models

Problems with algebraic model solving

DAE-set != DAE-sequence



cycles ?

No

Set
a = d - c
b = c + a - 3
c = 6 + d * e
d = 2
e = 3
a = 0
b = 0
~~c = 6~~
~~d = 2~~
~~e = 3~~

sorting

Sequence
d = 2
e = 3
c = 6 + d * e
a = d - c
b = c + a - 3

Yes

Set
a = b + 3
b = a / 2

solve

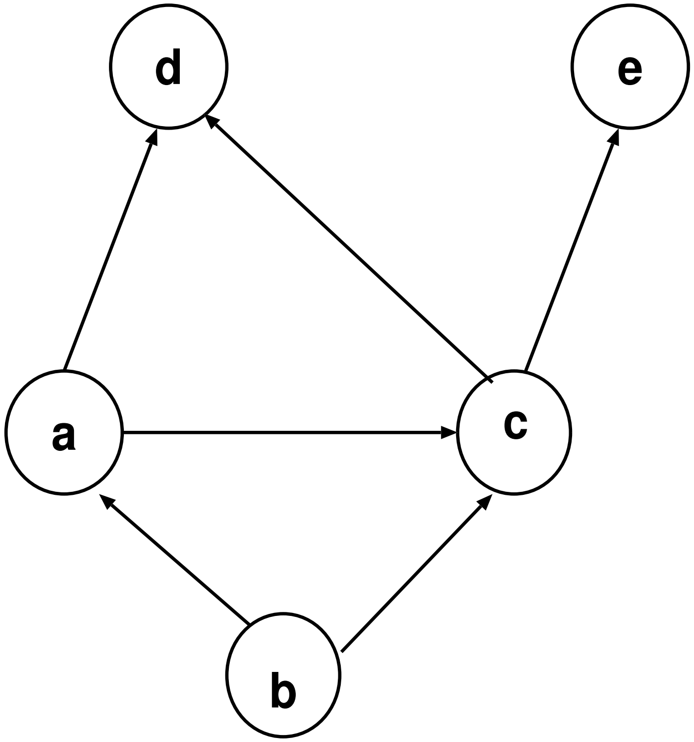
Sequence
a = 3
b = 3

linear

nonlinear

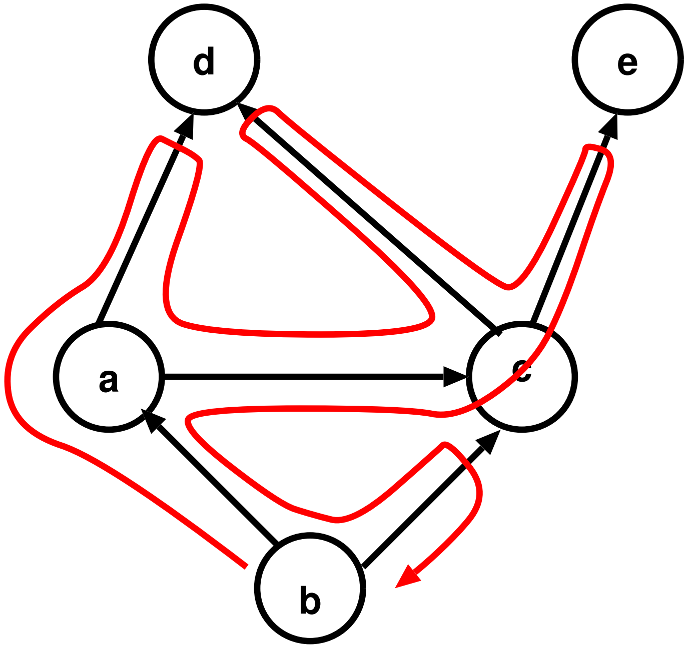
Dependency Graph

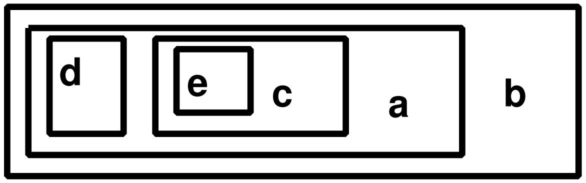
$$\left\{\begin{array}{l} a = d - c \\ b = c + a - 3 \\ c = 6 + d * e \\ d = 2 \\ e = 3 \end{array}\right.$$



Problems with model re-use: sorting post-order traversal depth-first search

```
1. find_root(s)
2. DFS(root)

DFS(node)
{
    if (not_visited(node))
    {
        mark_visited(node)
        foreach child_node of node
        {
            DFS(child_node)
        }
        print(node)
    }
}
```





Dependency Cycle (aka Algebraic Loop)

$$\begin{cases} x & = & y + 16 \\ y & = & -x - z \\ z & = & 5 \end{cases}$$

can *never* be sorted due to a dependency *cycle*
aka *strong component* (every vertex in the component is reachable from every other)

$$x \rightarrow y \rightarrow x$$

May be solved implicitly

$$
\left[
\begin{array}{c}
z = 5 \\
\left\{
\begin{array}{rcl}
x - y & = & -6 \\
x + y & = & -z
\end{array}
\right.
\end{array}
\right.
$$

Implicit set of n equations in n unknowns.

- non-linear → non-linear residual solver.
- linear → numeric or symbolic solution.

May be solved symbolically (if linear and not too large)

$$x = \frac{\begin{vmatrix} -6 & -1 \\ -z & 1 \end{vmatrix}}{\begin{vmatrix} 1 & -1 \\ 1 & 1 \end{vmatrix}} = \frac{-6-z}{2} \; ; \; y = \frac{\begin{vmatrix} 1 & -6 \\ 1 & -z \end{vmatrix}}{\begin{vmatrix} 1 & -1 \\ 1 & 1 \end{vmatrix}} = \frac{6-z}{2}$$

$$\left[\begin{array}{lcl} z & = & 5 \\ x & = & \frac{-6-z}{2} \\ y & = & \frac{6-z}{2} \end{array} \right.$$

Simple Loop Detection

1. Build dependency matrix D
2. Calculate transitive closure D^*
3. If $True$ on diagonal of D^*, a loop exists

Even with Warshall's algorithm, still $O(n^3)$ and don't know immediately which nodes involved in the loop(s).

Tarjan's $O(n+m)$ Loop Detection (1972)

1. Complete Depth First Search (DFS) on G
   (possibly multiple DFS trees), postorder numbering

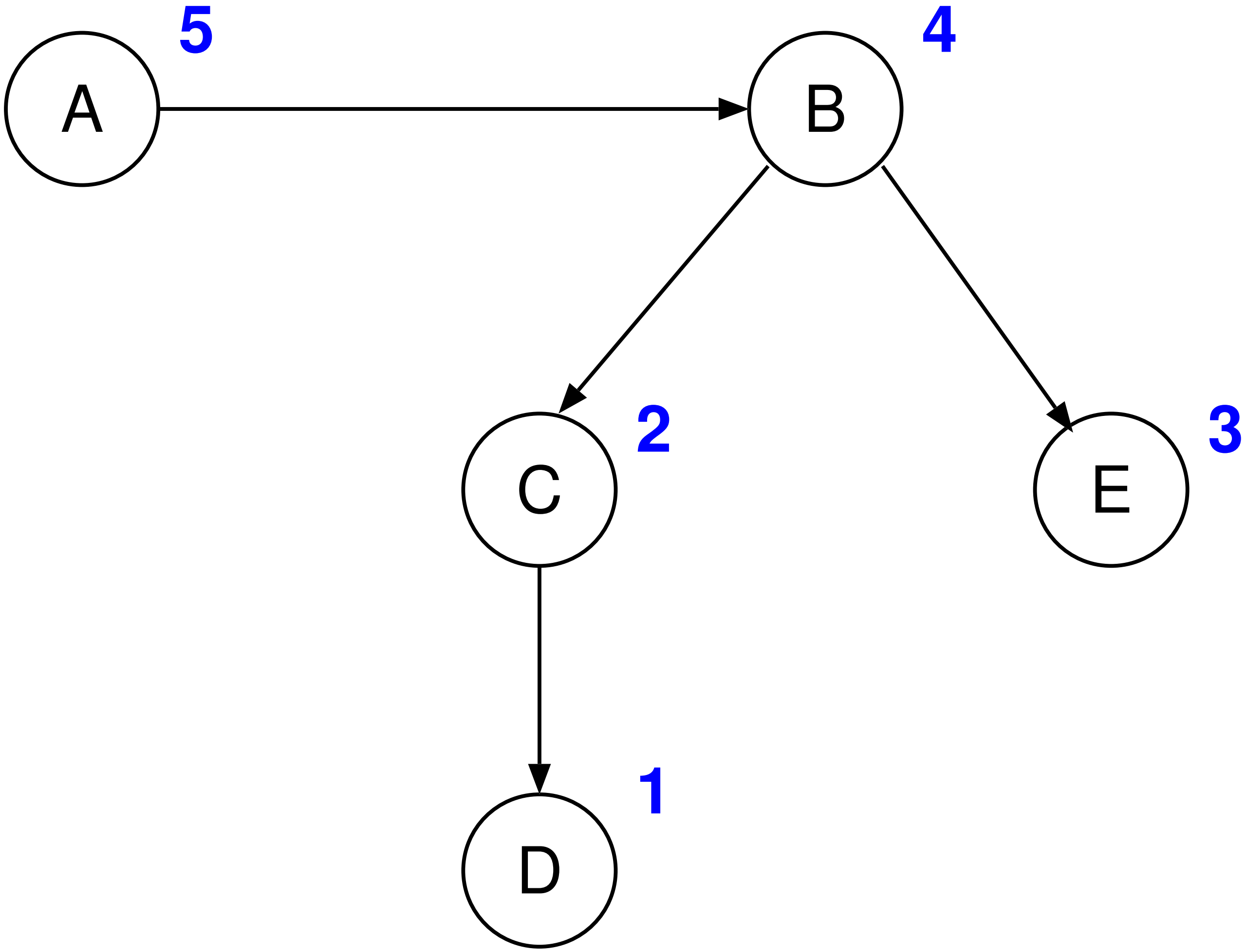
```
FOREACH v IN V
 dfsNr[v] <- 0
FOREACH v IN V
 IF dfsNr[v] == 0
  DFS(v)
```

2. Reverse edges in the annotated $G \to G_R$
3. DFS on G_R starting with highest numbered v
   set of vertices in each DFS tree $=$ strong component.
   Remove strong component and repeat.

Set of Algebraic Eqns, no Loops

$$\left\{\begin{array}{lcl} a & = & b^2+3 \\ b & = & sin(c \times e) \\ c & = & \sqrt{d-4.5} \\ d & = & \pi/2 \\ e & = & u() \end{array}\right.$$

Sorting, no Loops



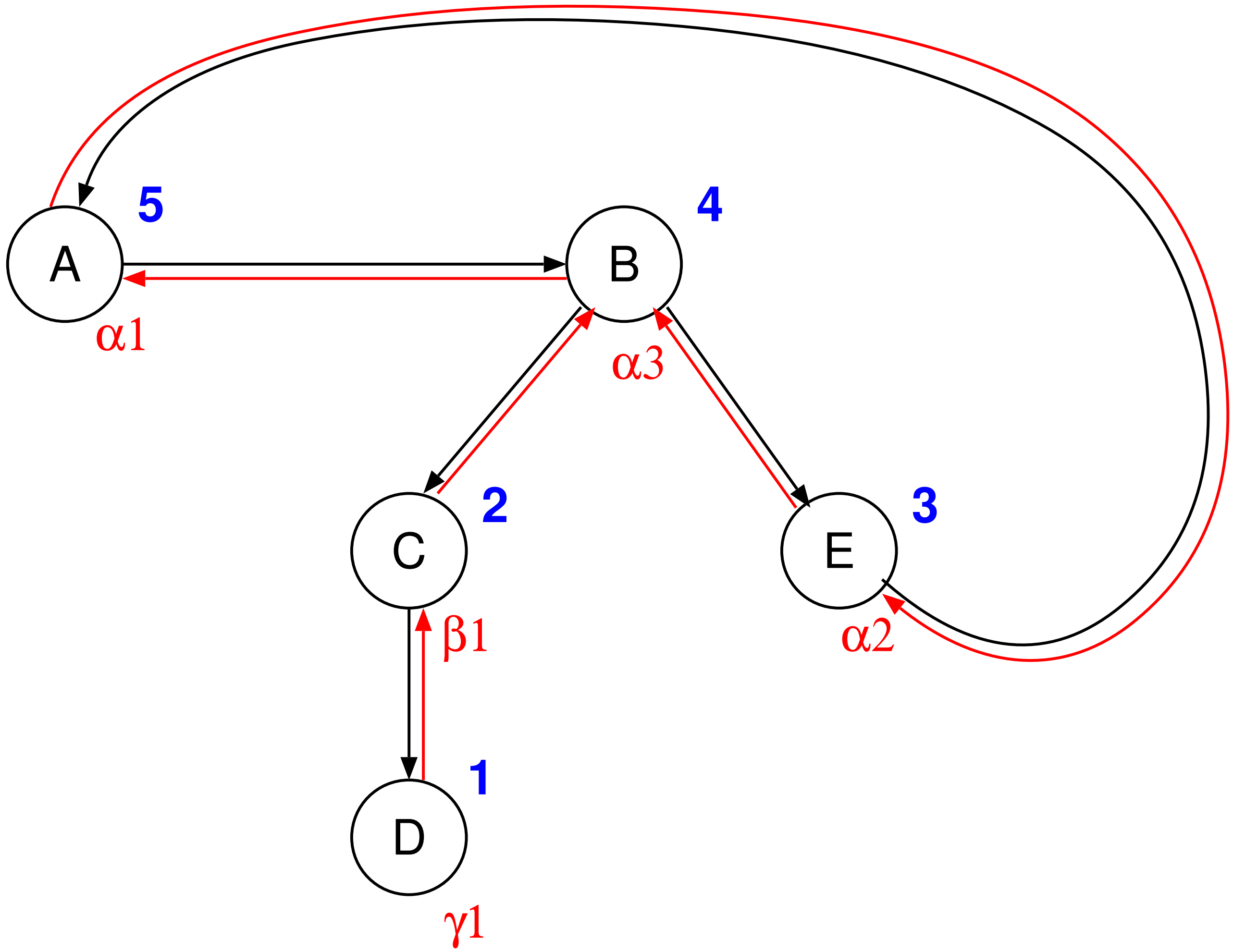
Sorting Result

$$\left[\begin{array}{lcl} d & = & \pi/2 \\ e & = & u() \\ c & = & \sqrt{d-4.5} \\ b & = & sin(c \times e) \\ a & = & b^2+3 \end{array}\right.$$

Algebraic Loop (Cycle) Detection

$$\begin{cases} a & = & b^2 + 3 \\ b & = & sin(c \times e) \\ c & = & \sqrt{d - 4.5} \\ d & = & \pi/2 \\ e & = & a^2 + u() \end{cases}$$

Algebraic Loop (Cycle) Detection



Algebraic Loop (Cycle) Detection Result

$$
\left[\begin{array}{lcl}
d & = & \pi/2 \\
c & = & \sqrt{d-4.5} \\
\left\{\begin{array}{l} b \\ a \\ e \end{array}\right. &
\begin{array}{c} = \\ = \\ = \end{array} &
\begin{array}{l} sin(c \times e) \\ b^2+3 \\ a^2+u() \end{array}
\end{array}\right. ;
\left[\begin{array}{l}
d = \pi/2 \\
c = \sqrt{d-4.5} \\
\left\{\begin{array}{cccccc}
 & b & & -sin(c \times e) & & = 0 \\
a & -b^2 & & & -3 & = 0 \\
a^2 & & -e & & +u() & = 0
\end{array}\right.
\end{array}\right.
$$

Continuous System Simulation Languages (CSSLs)

- Analog Computers
- block oriented vs. equation based
- the CSi 1968 CSSL standard
- CSSL-IV, ACSL, ADSIM/RT, . . .

CSSL Requirements

- Easy Model Description (equation based, block oriented)
- Integrator control:
  - select integrator
  - (initial) step size
  - error control
  - variable initialisation
  - parameter setting
- Documentation of model and experiments
- Structured: model vs. experiments (re-use)

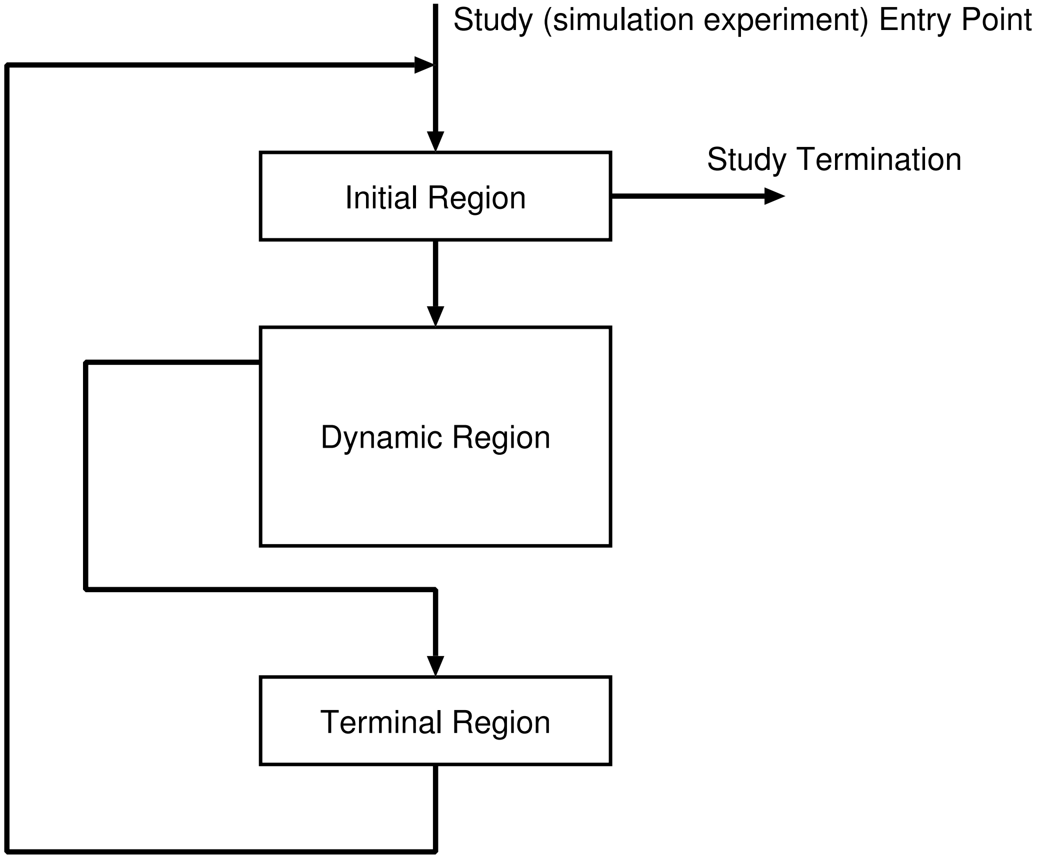
Model Description

```
DX = INTEG(F-B*X-A*DX, DX0)
 X = INTEG(DX, X0)
```

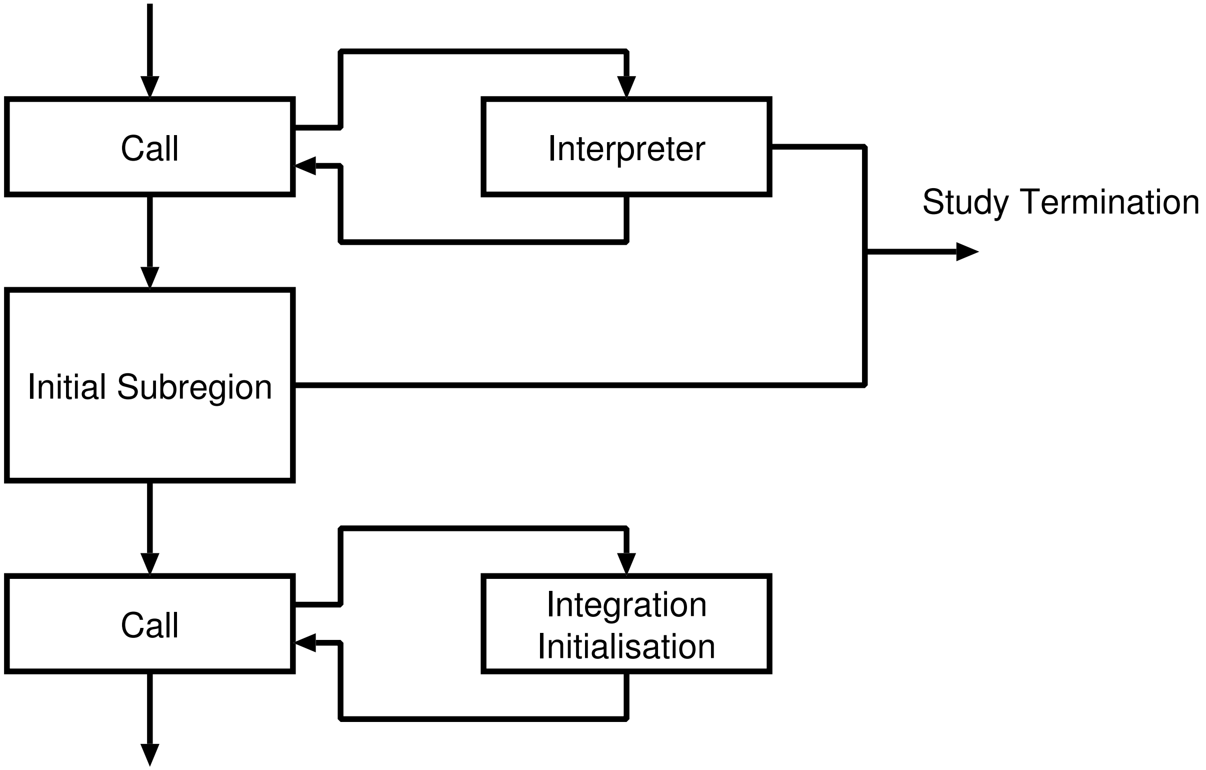
or

```
DX′ = F-B*X-A*DX
 X′ = DX

Initial Conditions at t = 0:

   X = X0
  DX = DX0
```

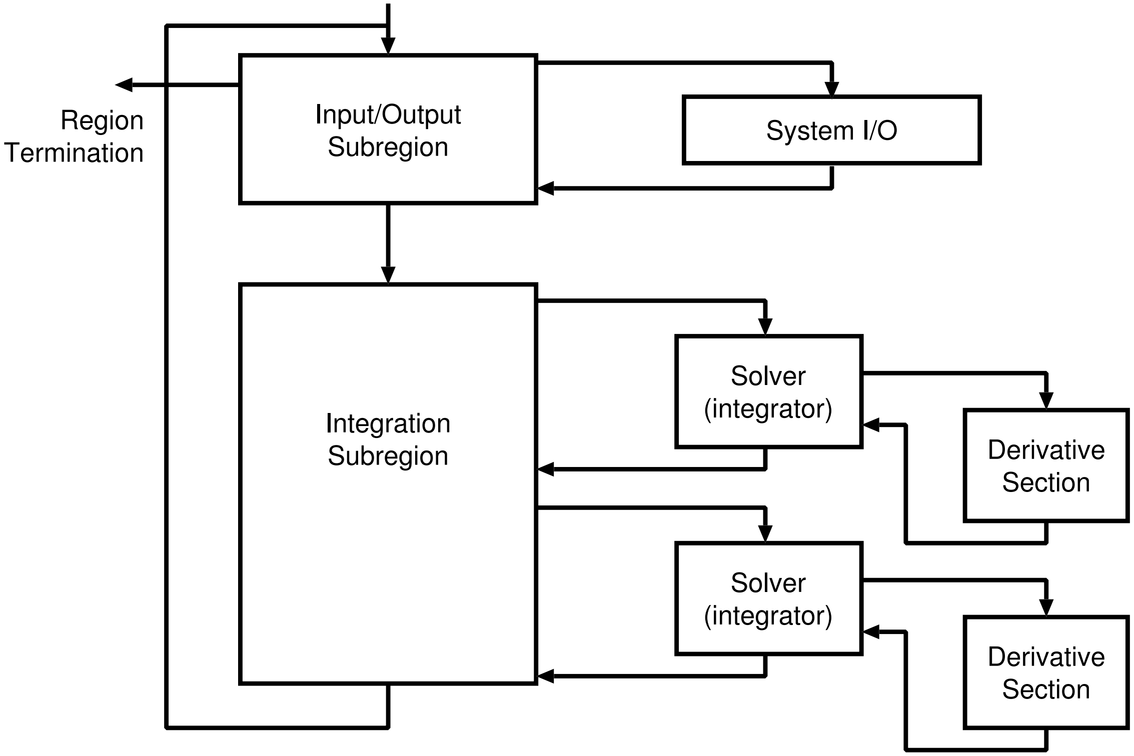
“CSSL study” structure



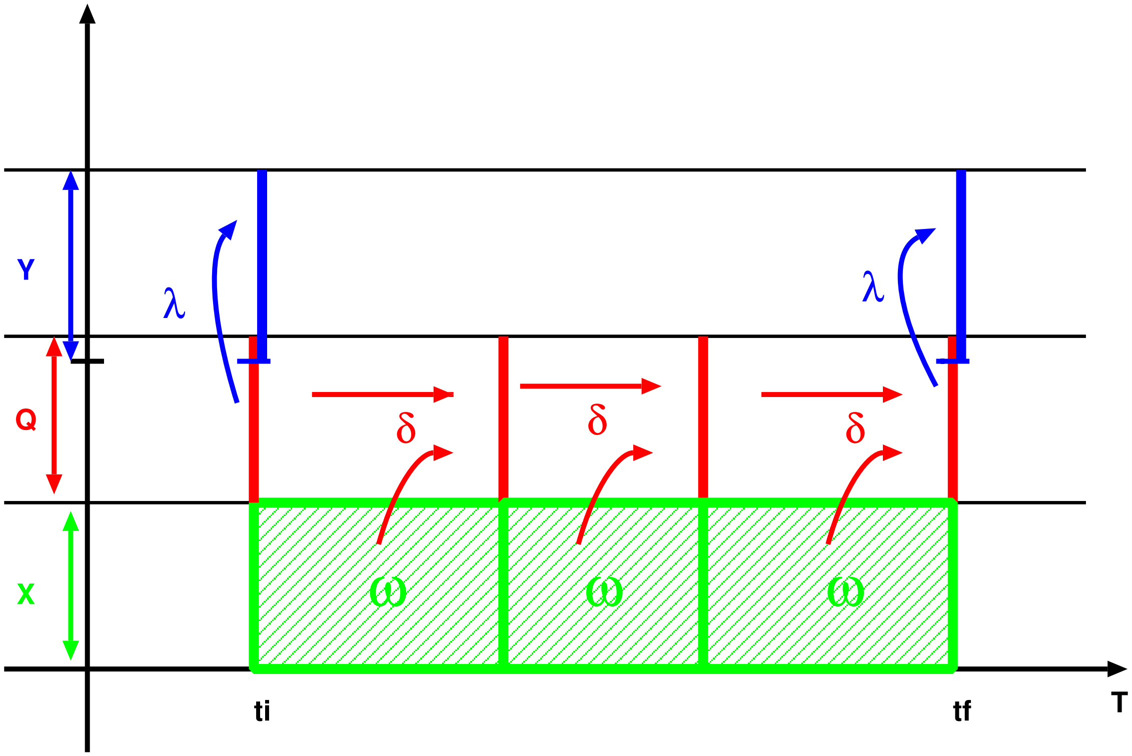
“CSSL initial region” structure



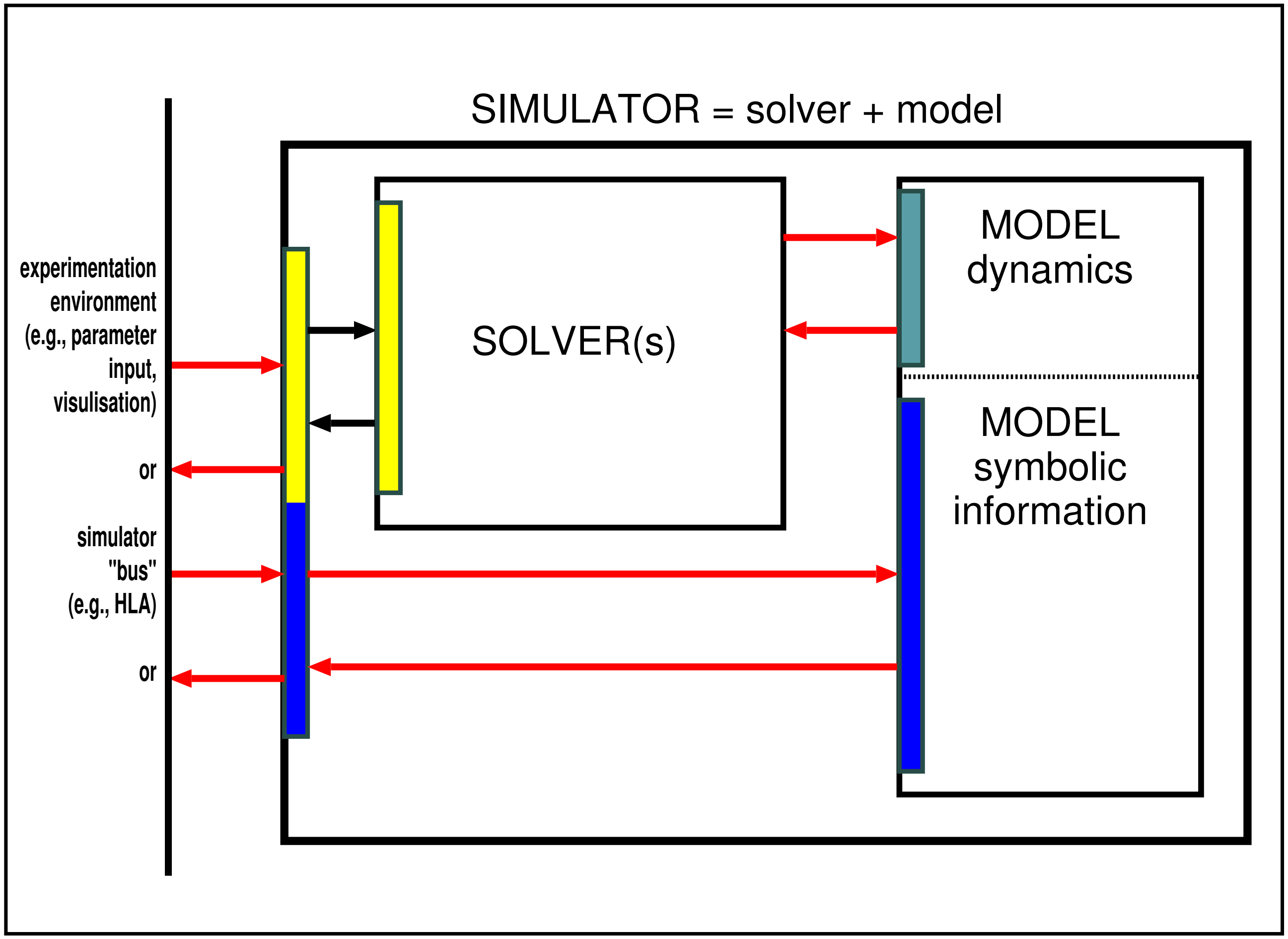
“CSSL dynamic region” structure



General, state-based simulation kernel



Model-solver Architecture



MSL-EXEC Model Representation

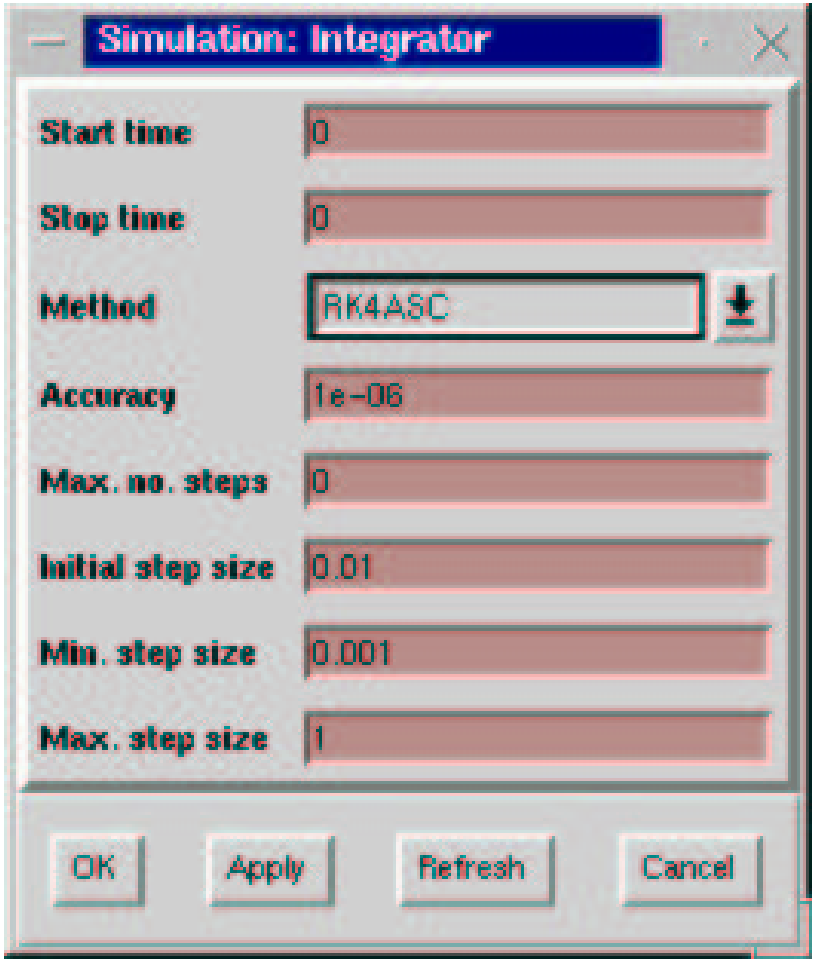
```
#include <math.h>
#include <assert.h>
#include "MSLE.h"
#include "MSLExternal.h"
#include "MSLU.h"
#include "Circle.h"

#define _t_ IndepVarValues[0]
#define _x_out_ OutputVarValues[0]
#define _y_out_ OutputVarValues[1]
#define _x_ DerStateVarValues[0]
#define _y_ DerStateVarValues[1]
#define _D_x_ Derivatives[0]
#define _D_y_ Derivatives[1]

CircleClass :: CircleClass(StringType name_arg)
{
  set_name(name_arg);
  set_description("Circle test.");
  set_class_name("CircleClass");

  set_no_indep_vars(1);
  set_indep_var(0, new MSLEIndepVarClass("t", "s"));

  set_no_output_vars(2);
  set_output_var(0, new MSLEOutputVarClass("x_out", "", 0));
  set_output_var(1, new MSLEOutputVarClass("y_out", "", 0));

  set_no_der_state_vars(2);
  set_der_state_var(0, new MSLEDerStateVarClass("x", "", 0.1));
  set_der_state_var(1, new MSLEDerStateVarClass("y", "", 0.1));
```

```
  set_no_indep_var_values(1);
  GetIndepVar(0)->LinkValue(this, MSLE_INDEP_VAR, 0);

  set_no_output_var_values(2);
  GetOutputVar(0)->LinkValue(this, MSLE_OUTPUT_VAR, 0);
  GetOutputVar(1)->LinkValue(this, MSLE_OUTPUT_VAR, 1);

  set_no_der_state_var_values(2);
  GetDerStateVar(0)->LinkValue(this, MSLE_DER_STATE_VAR, 0);
  GetDerStateVar(1)->LinkValue(this, MSLE_DER_STATE_VAR, 1);
  GetDerStateVar(0)->LinkInitialValue(this, 0);
  GetDerStateVar(1)->LinkInitialValue(this, 1);
  GetDerStateVar(0)->LinkDerivative(this, 0);
  GetDerStateVar(1)->LinkDerivative(this, 1);

  Reset();
}
```

```
void CircleClass :: ComputeOutput(void)
{
  _x_out_ = _x_;
  _y_out_ = _y_;
}

void CircleClass :: ComputeInitial(void)
{
}

void CircleClass :: ComputeState(void)
{
  _D_x_ = _y_;
  _D_y_ = -_x_;
}

void CircleClass :: ComputeTerminal(void)
{
}

#undef _t_
#undef _x_out_
#undef _y_out_
#undef _x_
#undef _y_
```

MSL-EXEC simulator demo

GUI



Simulation: Integrator

Start time	0
Stop time	0
Method	RK4ASC
Accuracy	1e-06
Max. no. steps	0
Initial step size	0.01
Min. step size	0.001
Max. step size	1