



# 20-sim Tutorial

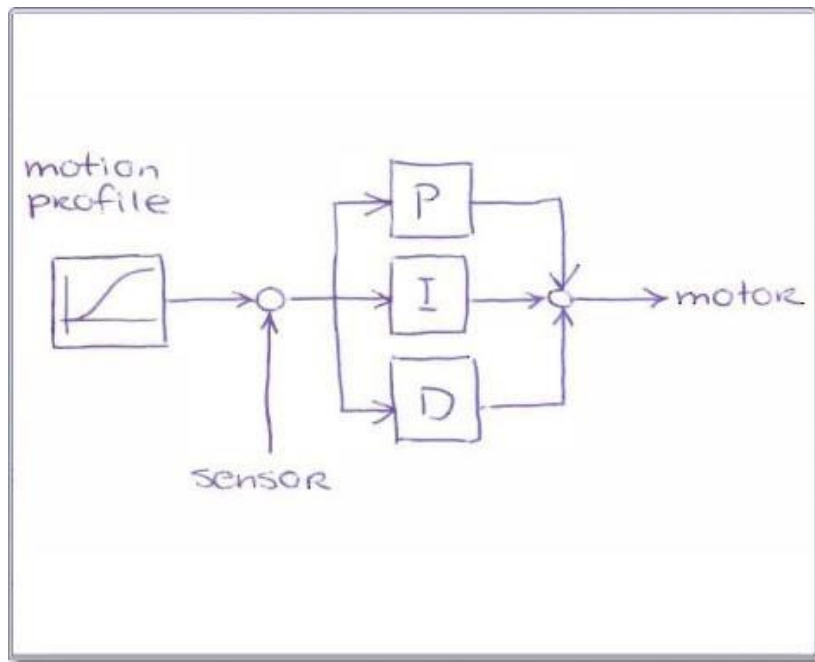
Ken Pierce, Newcastle University



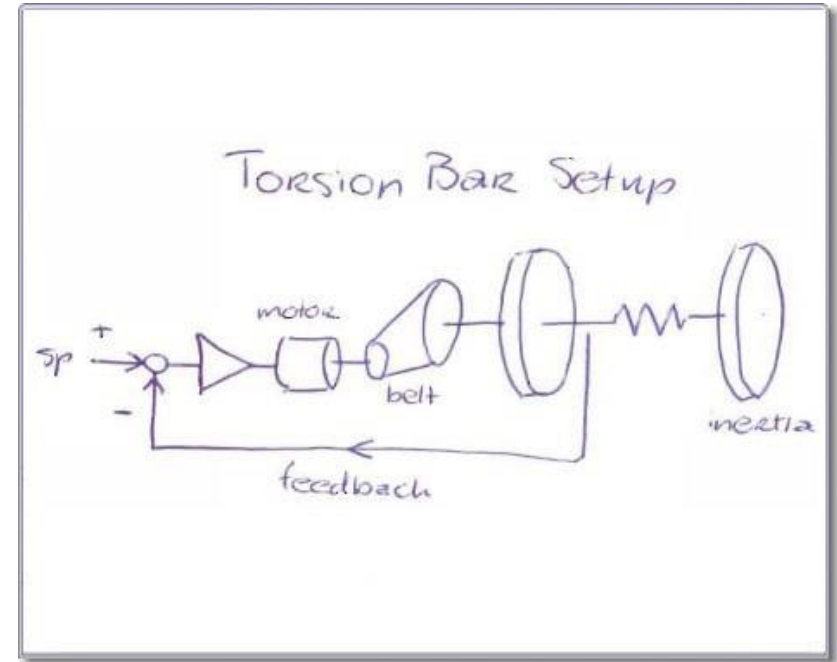
AARHUS  
UNIVERSITY



# From Sketch to ...

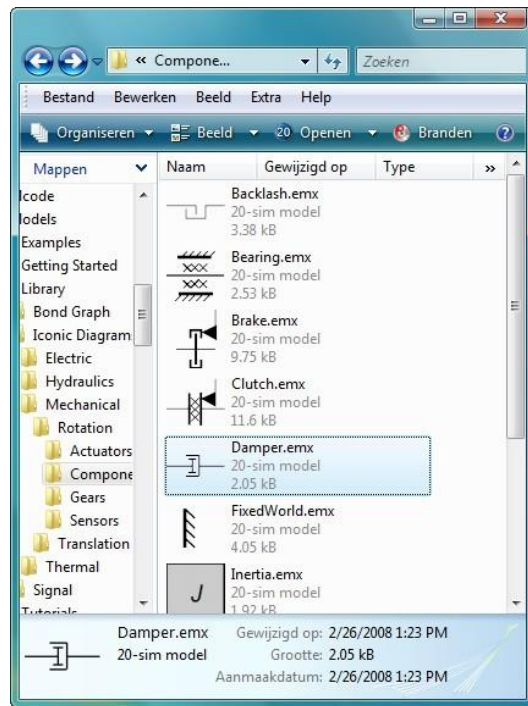


Controller

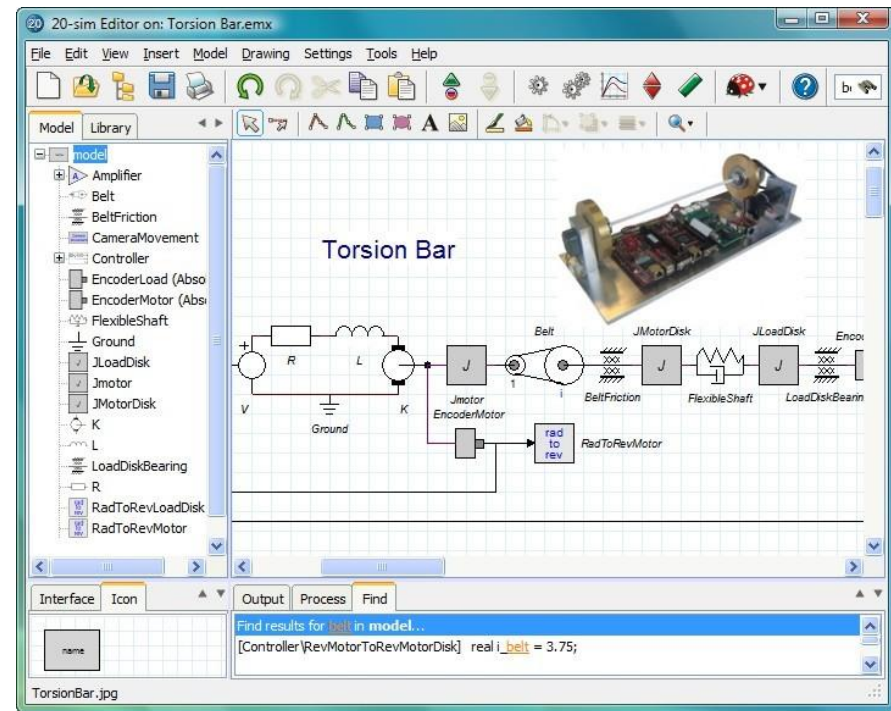


Physical System

# Modeling

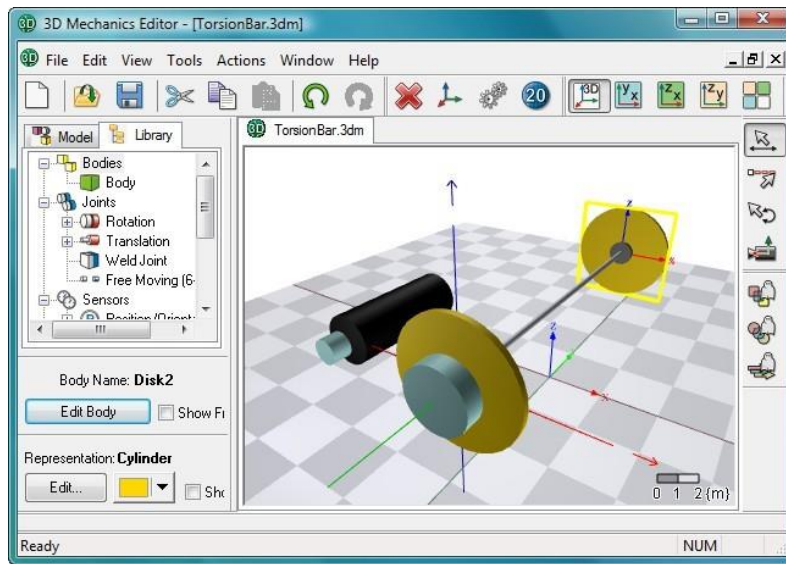


Library

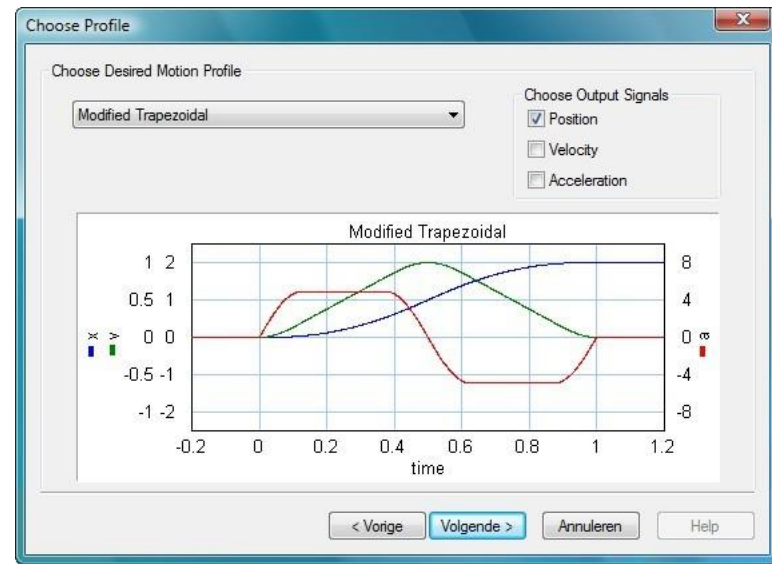


Editor

# Modeling Tools

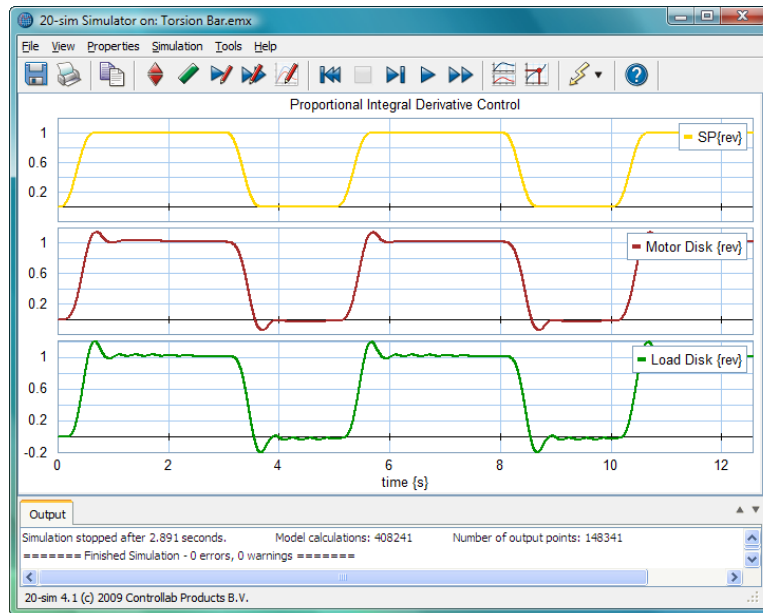


3D Mechanics Editor

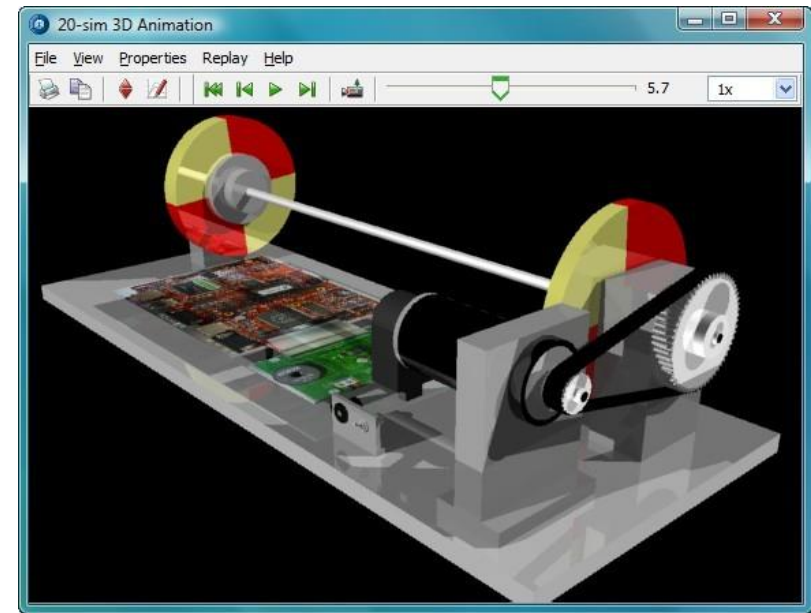


Motion Profile Wizard

# Simulation

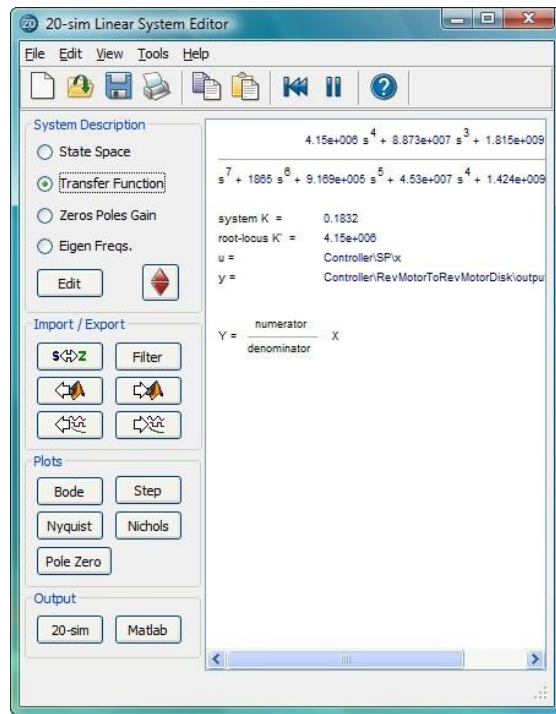


Simulator

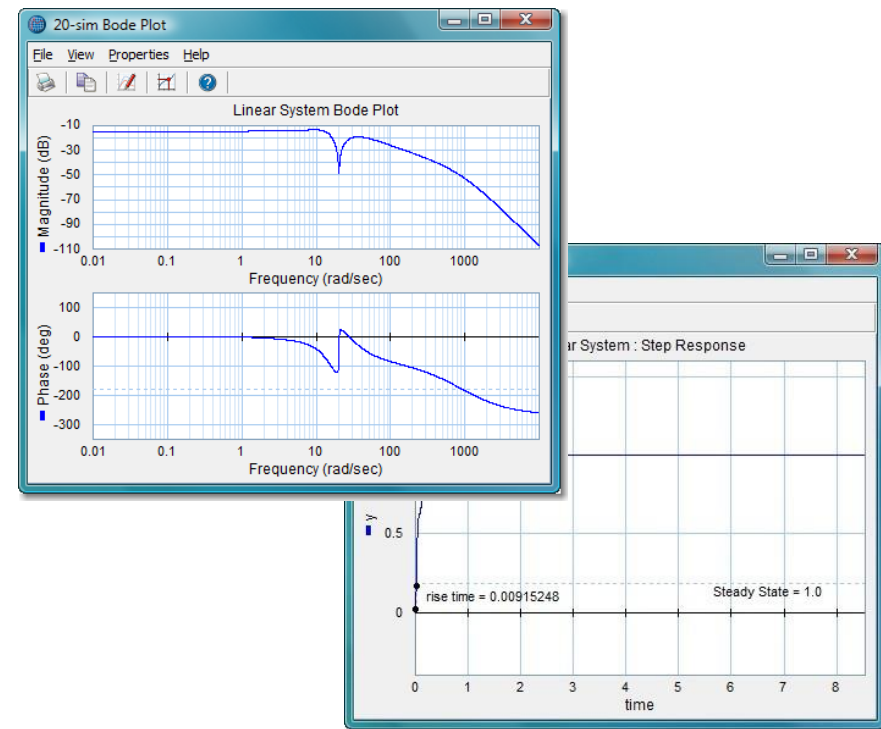


3D Animation

# Frequency Domain



Linearized model



Bode plot and step response

# C-code Generation

The screenshot shows a MATLAB script editor window titled "TextPad". The menu bar includes "File", "Edit", "Search", "View", "Tools", "Macros", and "Configure". The toolbar contains icons for file operations and editing. The left sidebar shows a project tree with "controller.c" selected under a folder icon. The main workspace displays a MATLAB script with various matrix calculations and variable assignments. Line numbers 33 through 45 are visible on the left margin.

```

xx_M[3].mat[2] = 0.0;
xx_M[3].mat[3] = xx_V[33];
xx_M[3].mat[4] = xx_V[36];
xx_M[3].mat[5] = 0.0;
xx_M[3].mat[6] = 0.0;
xx_M[3].mat[7] = 0.0;
xx_M[3].mat[8] = 1.0;

/* invkin_R10 = (invkin\Rx * inv
XXMatrixMul (&xx_M[21], &xx_M[1]
XXMatrixMul (&xx_M[4], &xx_M[21]

/* ref_z\PlusMinus8\output = ref
xx_V[291] = xx_V[290] + xx_V[284]

/* invkin_H10 = homogeneous (inv
xx_M[22].mat[0] = xx_V[272];
xx_M[22].mat[1] = xx_V[278];
xx_M[22].mat[2] = xx_V[291];
XXMatrixHomogeneous (&xx_M[5], &

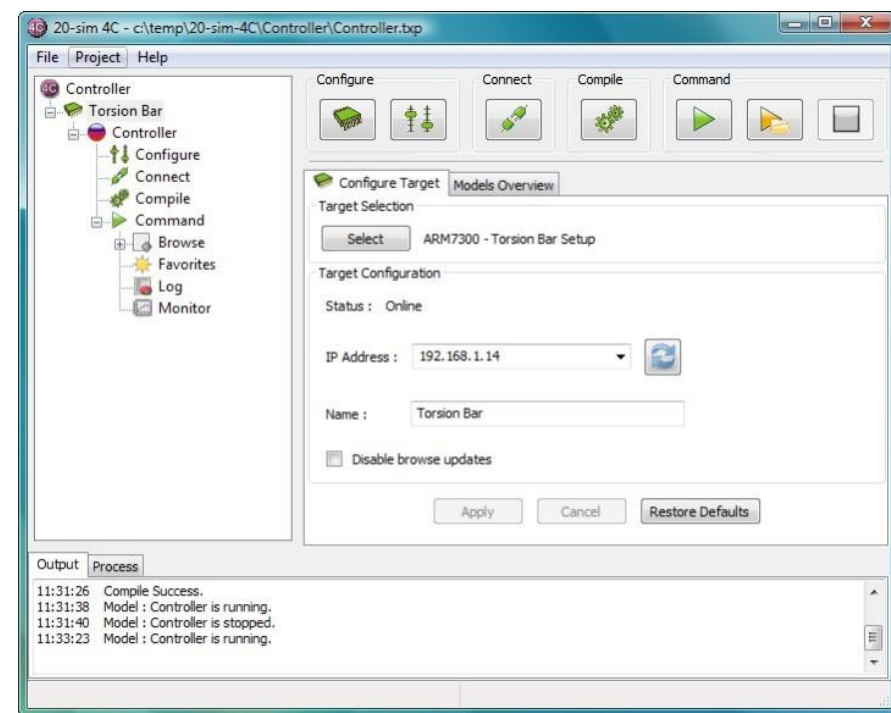
/* invkin_HA0 = invkin_H10 * inv
XXMatrixMul (&xx_M[9], &xx_M[5].

/* invkin_HB0 = invkin_H10 * inv
XXMatrixMul (&xx_M[10], &xx_M[5]

/* invkin_HC0 = invkin_H10 * inv
XXMatrixMul (&xx_M[11], &xx_M[5]

/* invkin_l1 = norm (invkin_HAA0
... M[24].mat[2] ... M[22].mat[2]
```

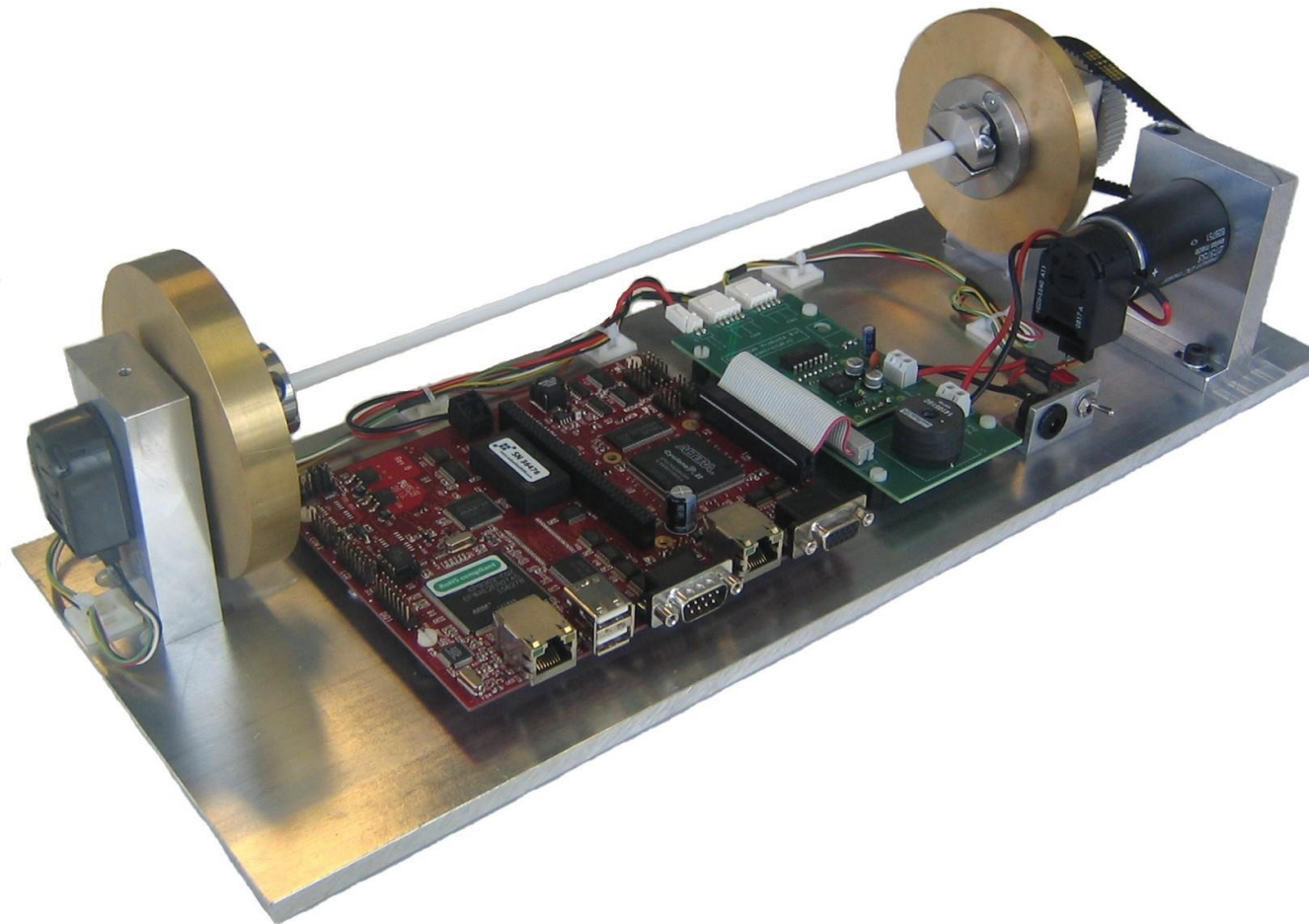
## C-code



20-sim 4C



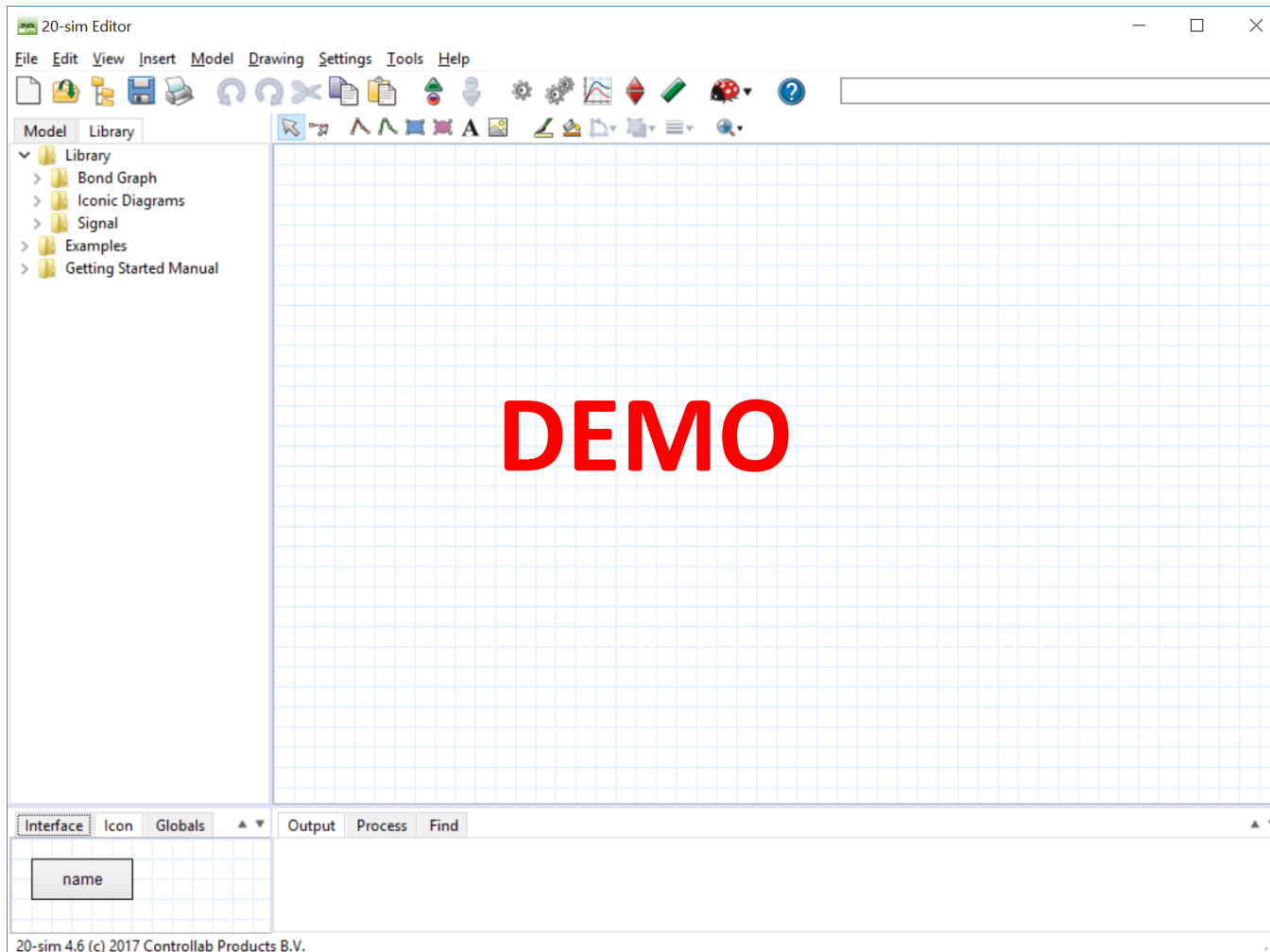
# .. to Prototype



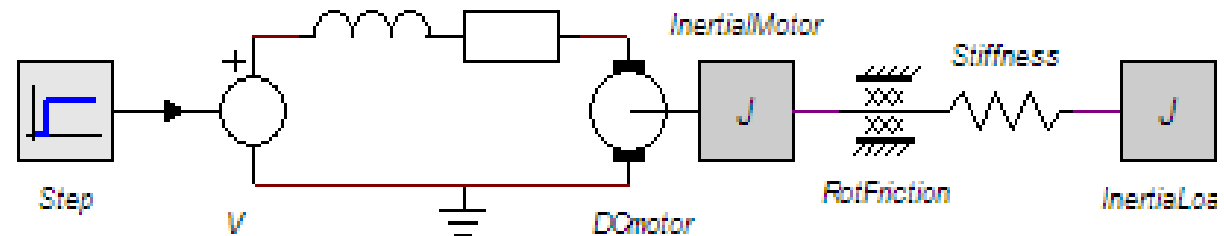
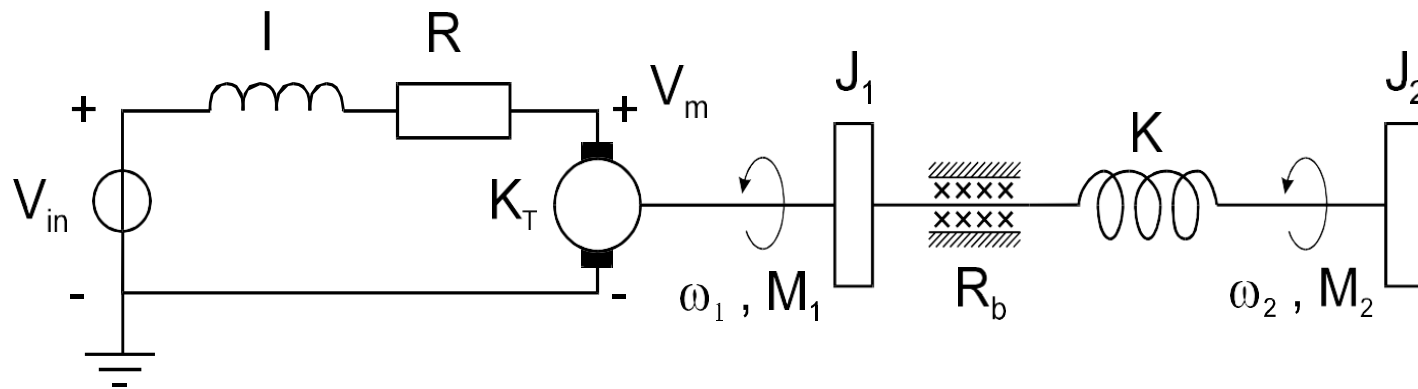




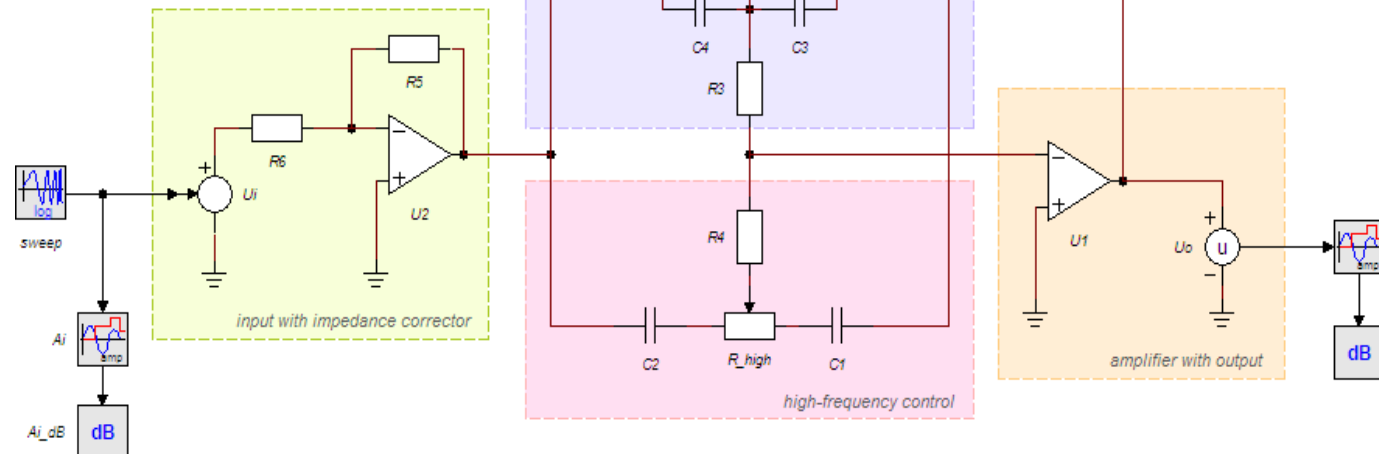
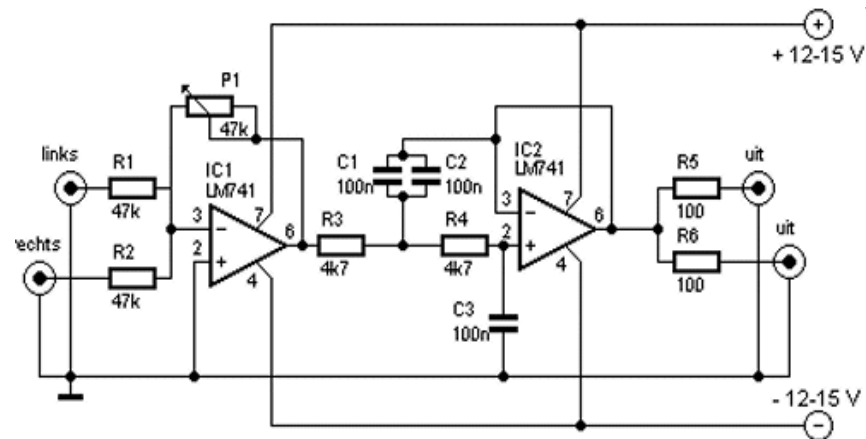
# Demonstration



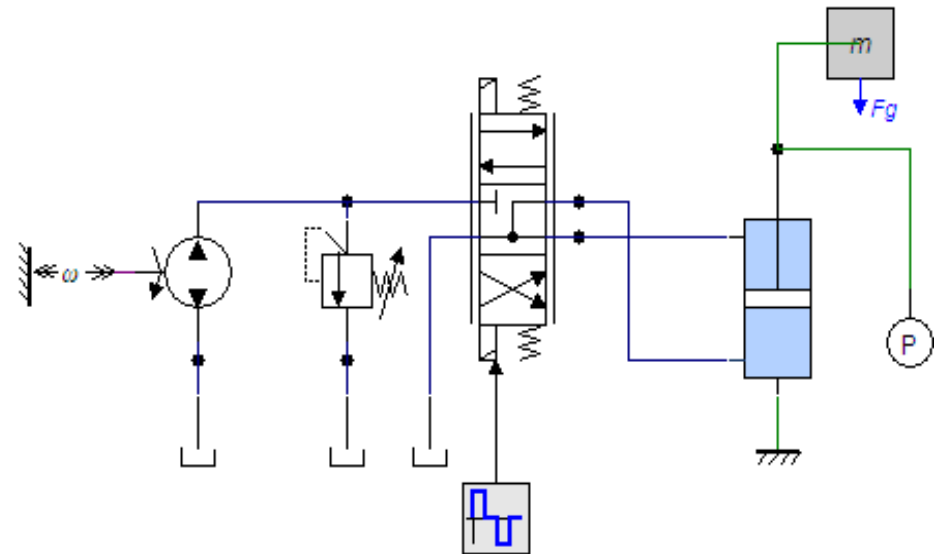
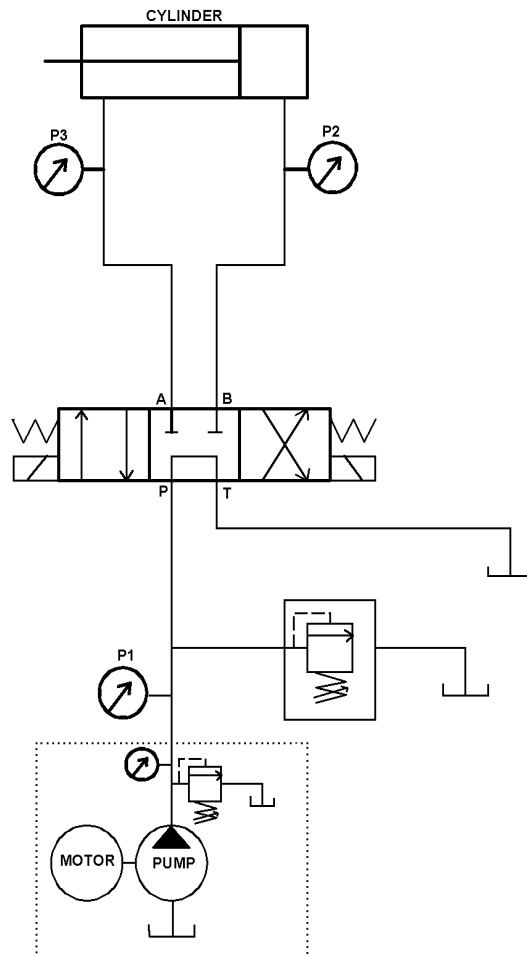
# Mechanics






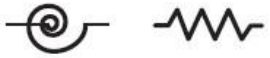
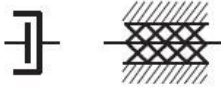




# Electronics



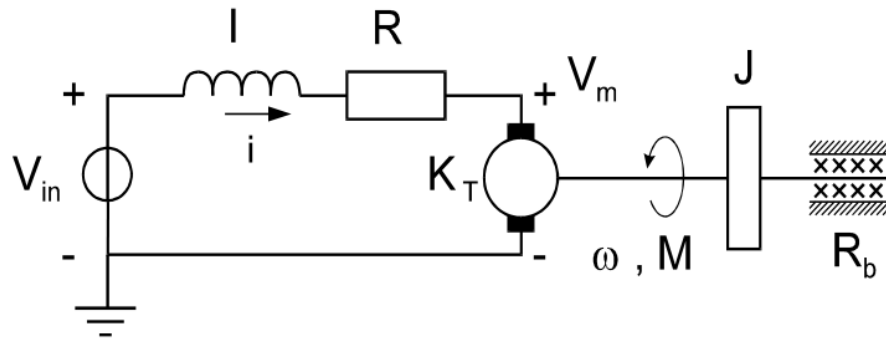
# Hydraulics



# Domains

<i>domain</i>	<i>behaviours</i>		
mechanical (translation)	spring:  $P = Fv$ $F = \frac{1}{c} \int v dt$	damper / friction:  $F - dv = 0$	mass:  $v = \frac{1}{m} \int F dt$
mechanical (rotation)	spring:  $P = T\omega$ $T = \frac{1}{c} \int \omega dt$	damper/friction:  $T - d\omega = 0$	inertia:  $\omega = \frac{1}{J} \int T dt$
electrical	capacitor:  $P = ui$ $u = \frac{1}{C} \int i dt$	resistor:  $u - Ri = 0$	inductance:  $i = \frac{1}{L} \int u dt$
hydraulic $P = p\phi$	tank: $p = \frac{1}{c} \int \phi dt$	resistance: $p - r\phi = 0$	inertia: $\phi = \frac{1}{I} \int p dt$

# Differential Equations



$$\dot{i} = \int \frac{V_{in}}{L} - \frac{R \times i}{L} - \frac{K_T \times \omega}{L} dt$$

$$\dot{\omega} = \int \frac{K_T \times i}{J} - \frac{R_b \times \omega}{J} dt$$

$$x(t) = x(0) + \int_0^t f(x, u, t) dt$$

$$y(t) = g(x, u, t)$$

General simulation model:

set of first-order equations  
and output equation





# Language Definition

Fixed during simulation	_____	<b>constants</b>	_____	Vary during simulation
		//enter your constants here		
	_____	<b>parameters</b>		
		//enter your parameters here		
		<b>variables</b>	_____	
		//enter your variables here		
Calculated at the start	_____	<b>initialequations</b>		
		// enter your initial equations here		
<b>Optimized</b>	_____	<b>equations</b>	_____	Calculated every simulation step
		// enter your equations here		
		<b>code</b>	_____	
<b>Not optimized</b>	_____	// enter your equations here		
		<b>finalequations</b>		
Calculated at the end	_____	// enter your final equations here		



# Language Example

## constants

real **g** = 9.81 {m/s<sup>2</sup>};      // gravity: free fall

## parameters

real **start\_time** = 10 {s};    // start time of the fall

string **out\_string** = 'run 1';

## variables

real **half\_g** {m/s<sup>2</sup>};

## initialequations

**half\_g** = -0.5\***g**;

## equations

if **time** < **start\_time** then

**output** = 0;

else

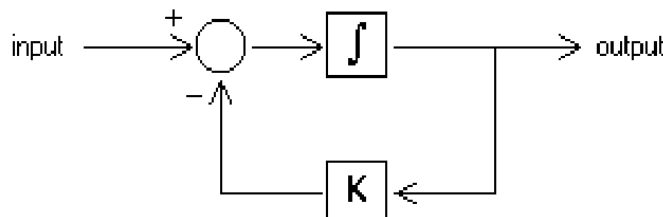
**output** = half\_g\* ( (**time**-**start\_time**)^2 );

end;

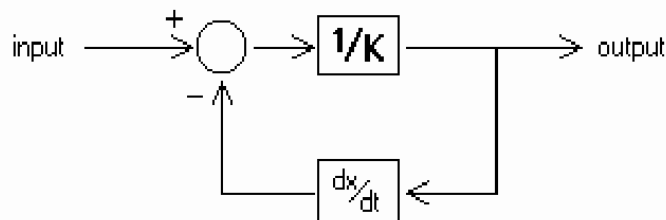
## finalequations

toMatlab ( **output**,out\_**string**);      // send output value to Matlab

# Integral Form & Differential Form



$$output = \int input - K \times output$$



$$output = \frac{input - \frac{d output}{dt}}{K}$$

- Integral form is preferred: much easier to simulate
- 20-sim will rewrite differential forms to integral form if possible
- Write your own equations in integral form if possible!



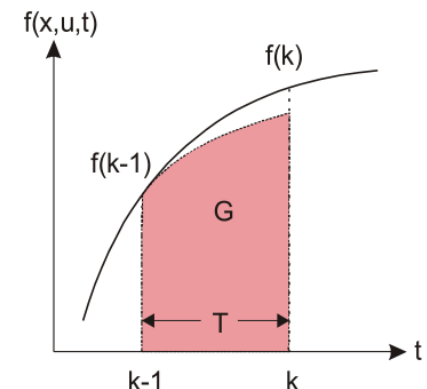
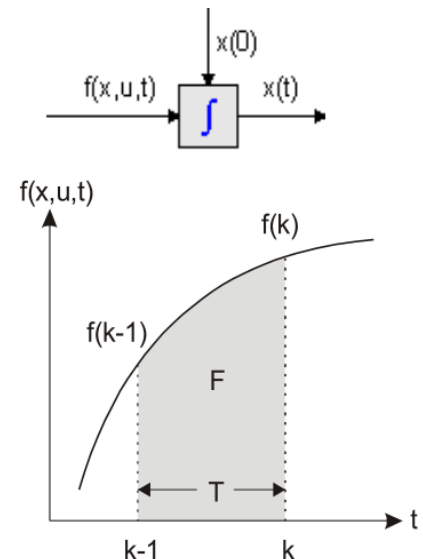
# Numerical Integration

$$x(t) = x(0) + \int_0^t f(x, u, t) dt$$

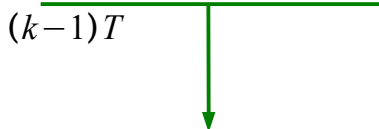
$$x(t) = x(k-1) + \int_{(k-1)T}^{kT} f(x, u, t) dt$$

Numerical Integration: approximate integral  
by analytical function G:

$$\tilde{x}(k) = \tilde{x}(k-1) + G \quad G \approx \int_{(k-1)T}^{kT} f(x, u, t) dt$$

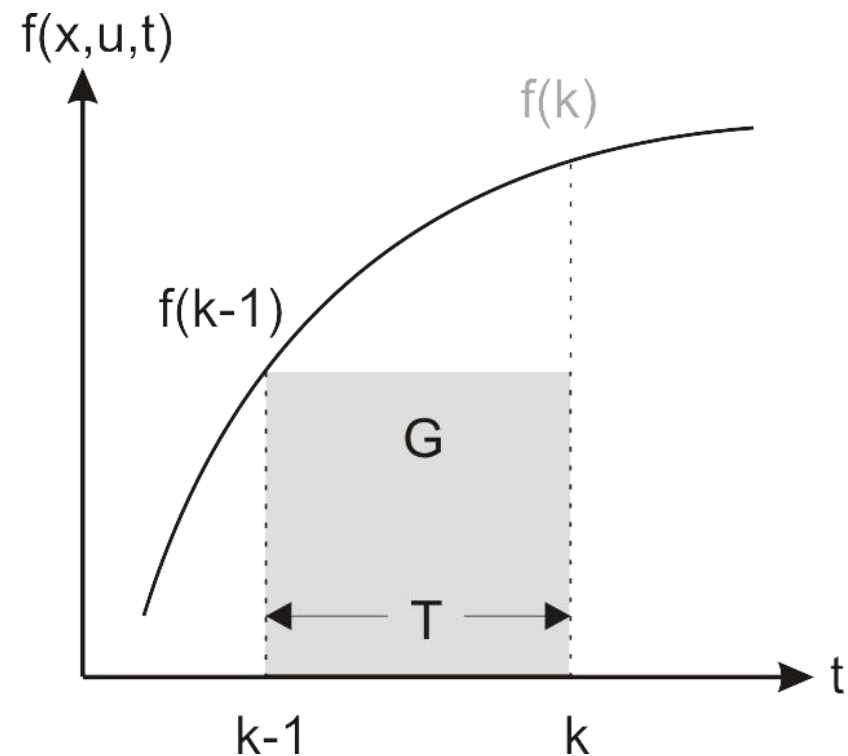


# Euler

$$x(t) = x(k-1) + \int_{(k-1)T}^{kT} f(x, u, t) dt$$


$$G = T \cdot f(k-1)$$

$$\tilde{x}(k) = \tilde{x}(k-1) + T \cdot f(k-1)$$



# Runge-Kutta 4

$$x(k) = x(k-1) + \int_{(k-1)T}^{kT} f(x,u,t) dt$$

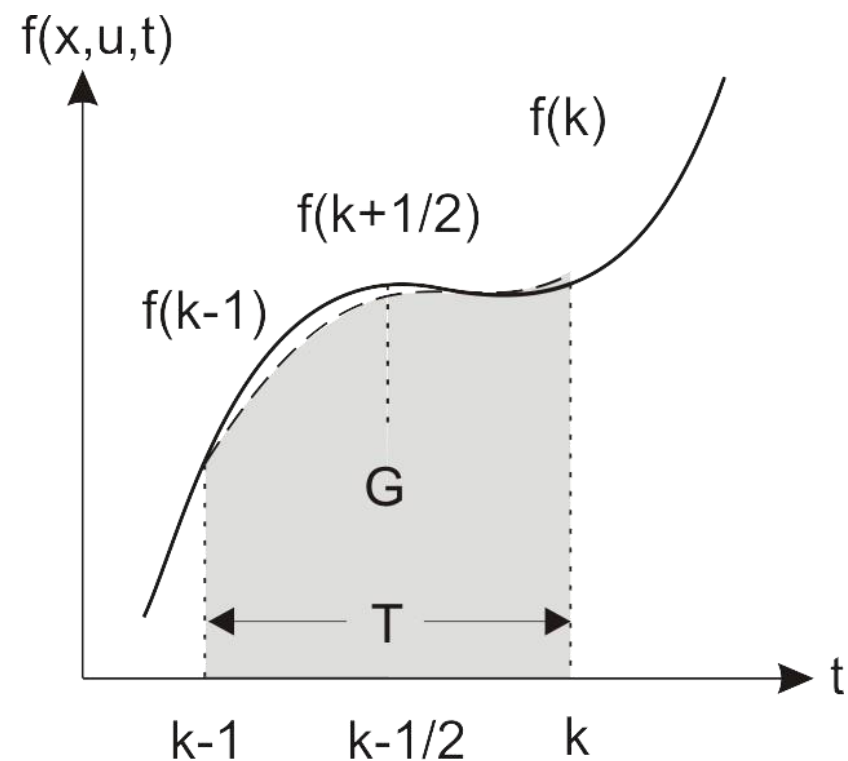
$$G = \frac{T}{6} (k_1 + 2k_2 + 2k_3 + k_4)$$

$$k_1 = f(x(k-1), u(k-1), k-1)$$

$$k_2 = f(x(k-1) + \frac{T}{2}k_1, u(k-1), k-1/2)$$

$$k_3 = f(x(k-1) + \frac{T}{2}k_2, u(k-1), k-1/2)$$

$$k_4 = f(x(k-1) + T k_3, u(k), k)$$

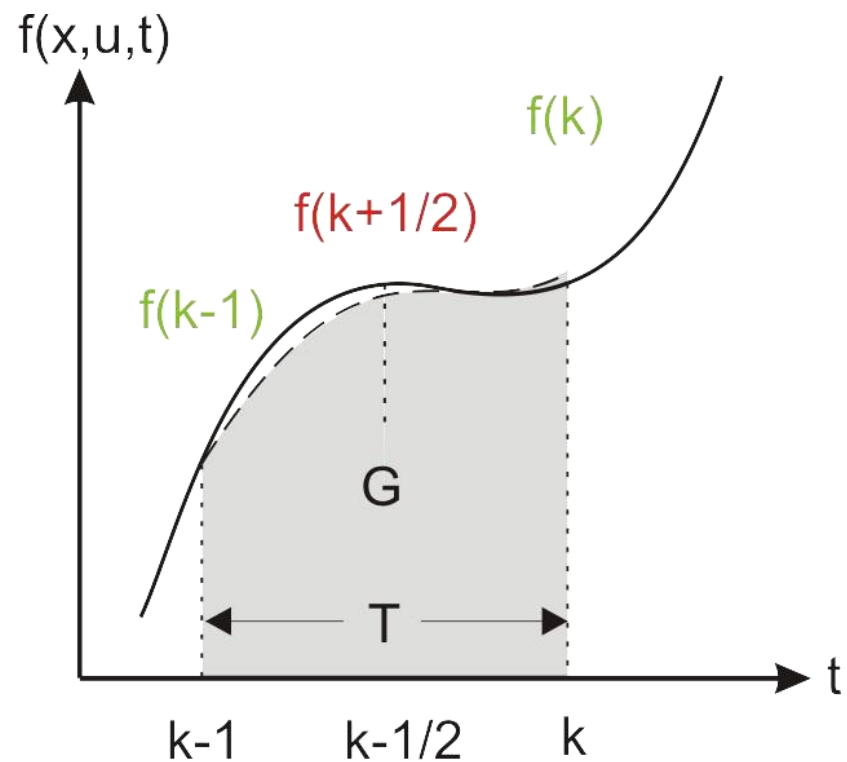




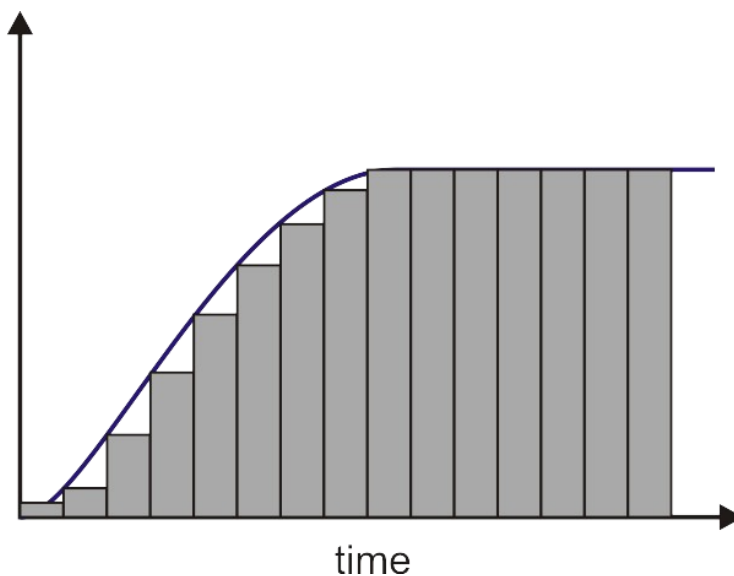
# Major and Minor Steps

- Major steps:  $f(k-1)$ ,  $f(k)$
- Minor steps:  $f(k+1/2)$
- Don't use code like:

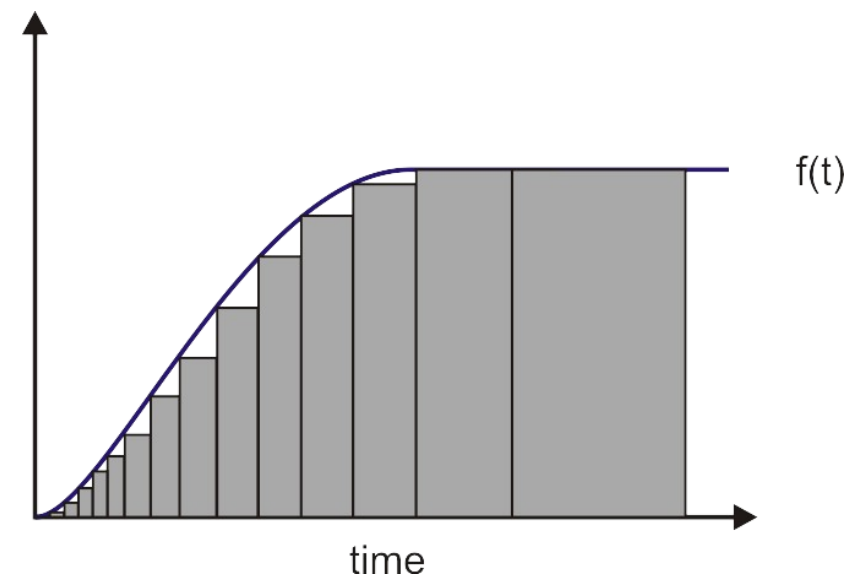
```
if (time > 0) then  
  x = x+1;  
end;
```



# Fixed and Variable Step



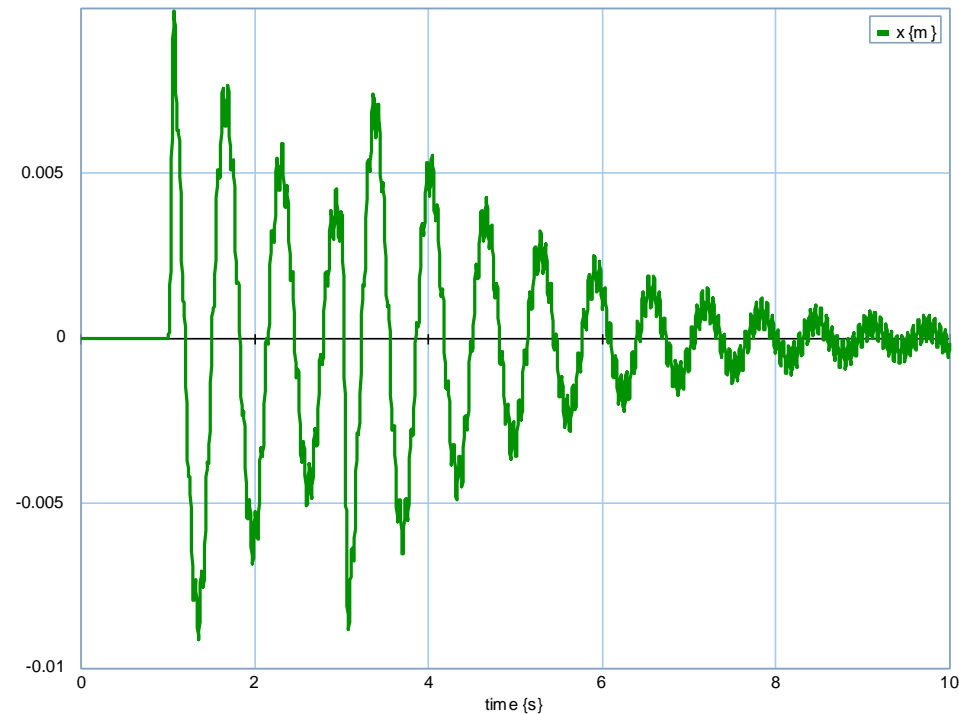
Fixed Step



Variable Step

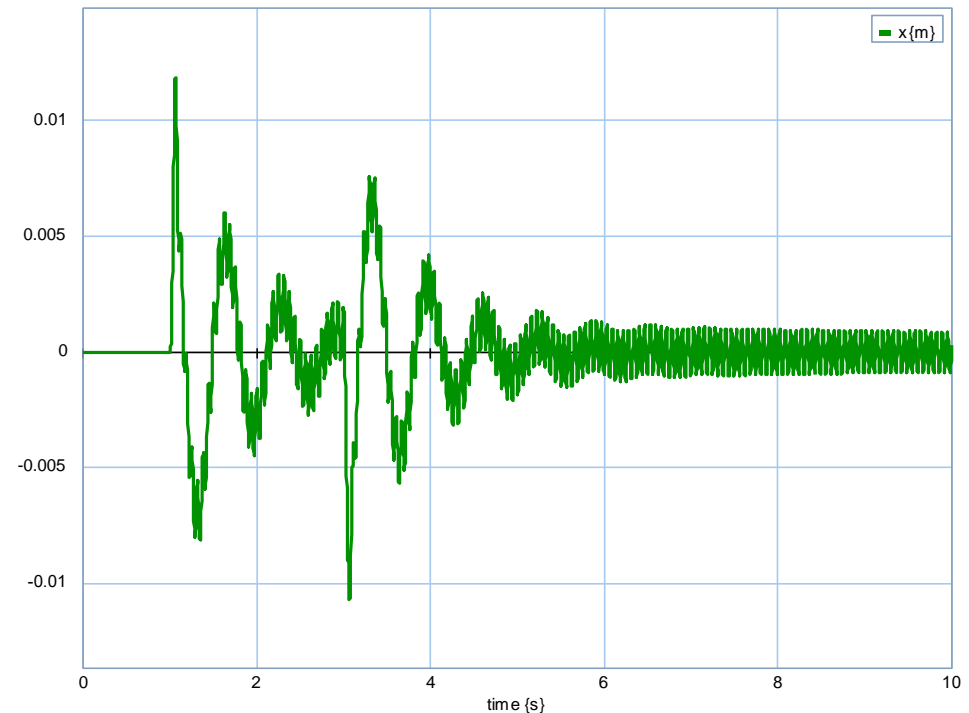
# Stiff Models

- Model has low frequency resonance  
high frequency resonances
- Hard to simulate with fixed step methods

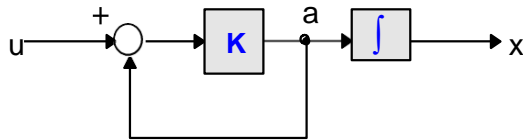


# No Damping

- Model has resonances which damp out poorly
- Hard to simulate with some methods



# Algebraic Loops



$$x(t) = x(k-1) + \int_{(k-1)T}^{kT} f(x, u, t) dt$$

$$f(x, u, t) = a$$

$$a = k \cdot u - a$$

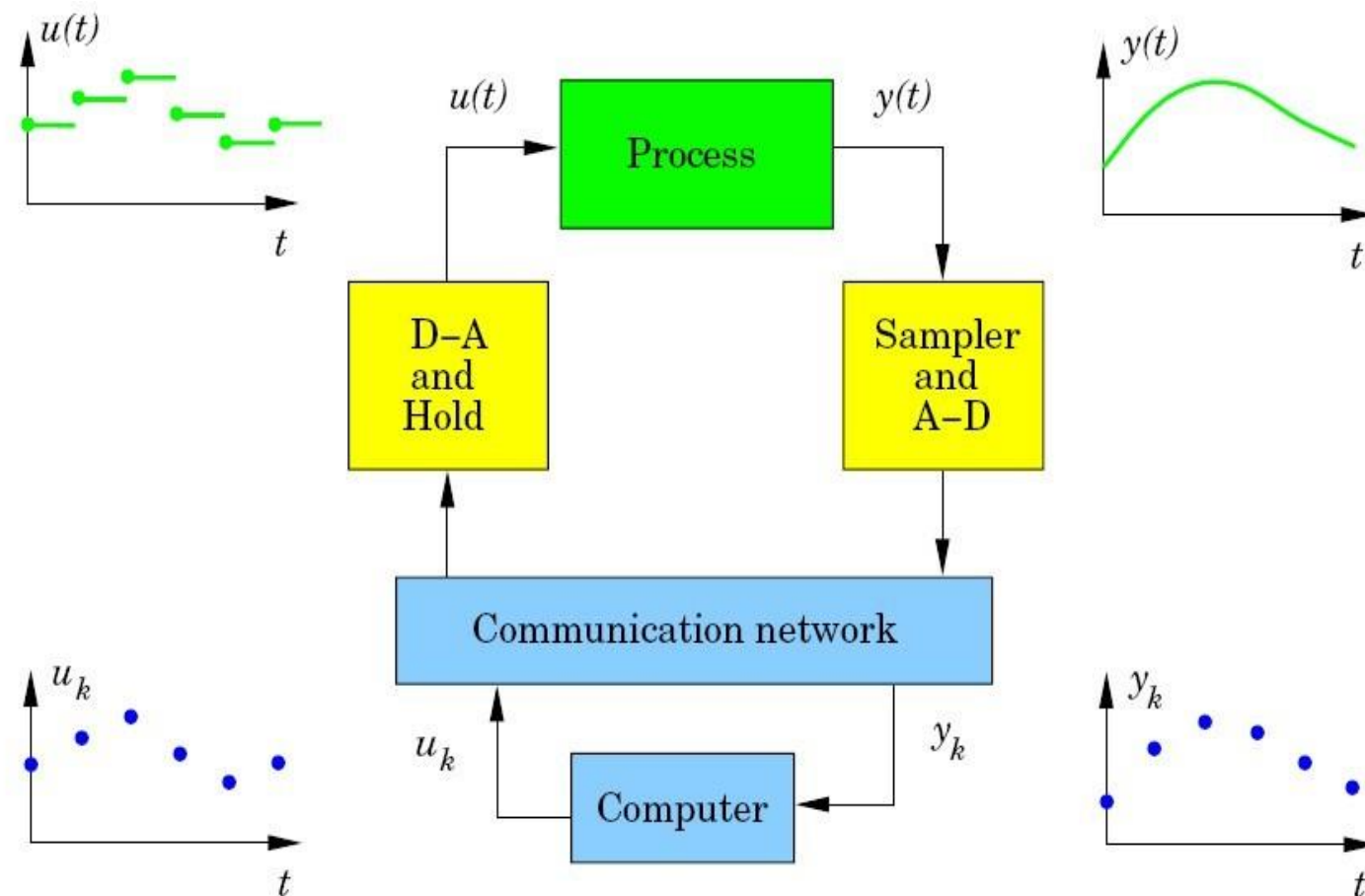


algebraic loop

Algebraic Loops:

- function that is a function of itself
- Require iteration to solve
- Reduces simulation speed

# Computer Controlled Systems



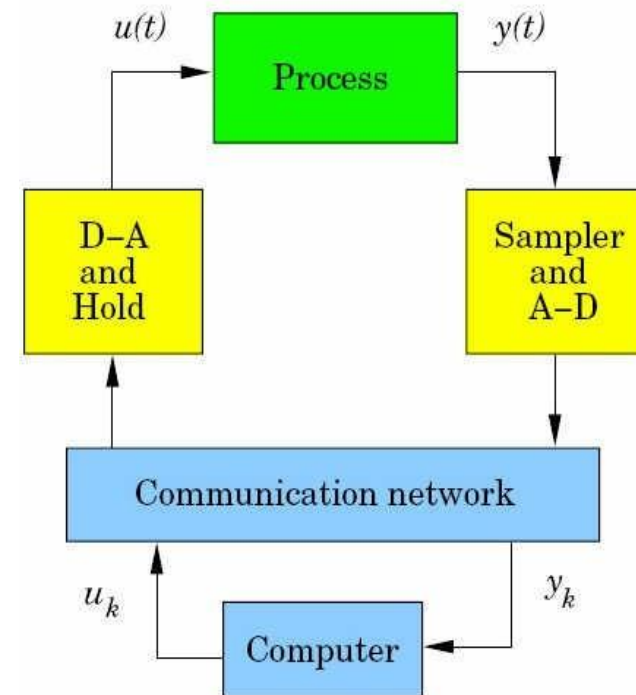




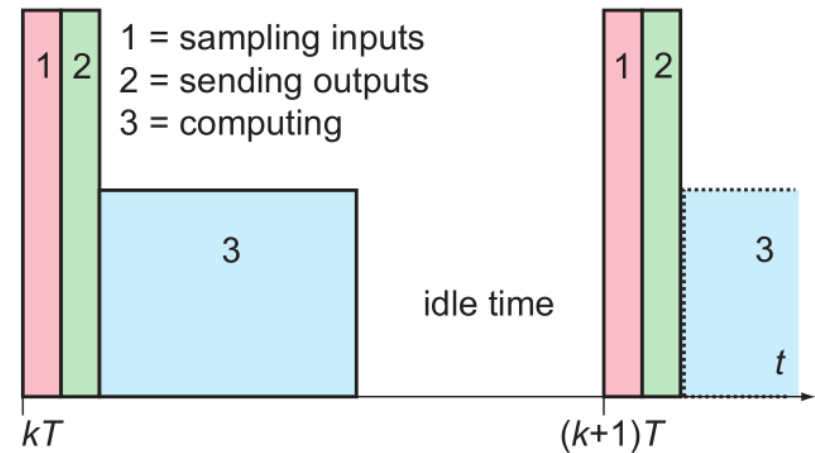
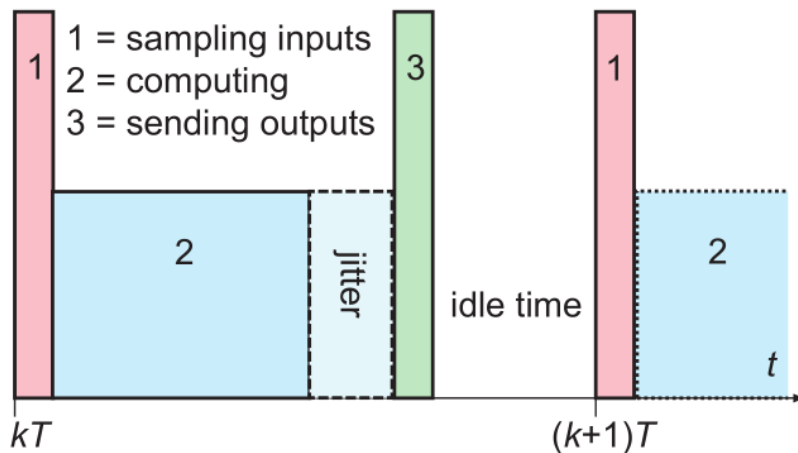
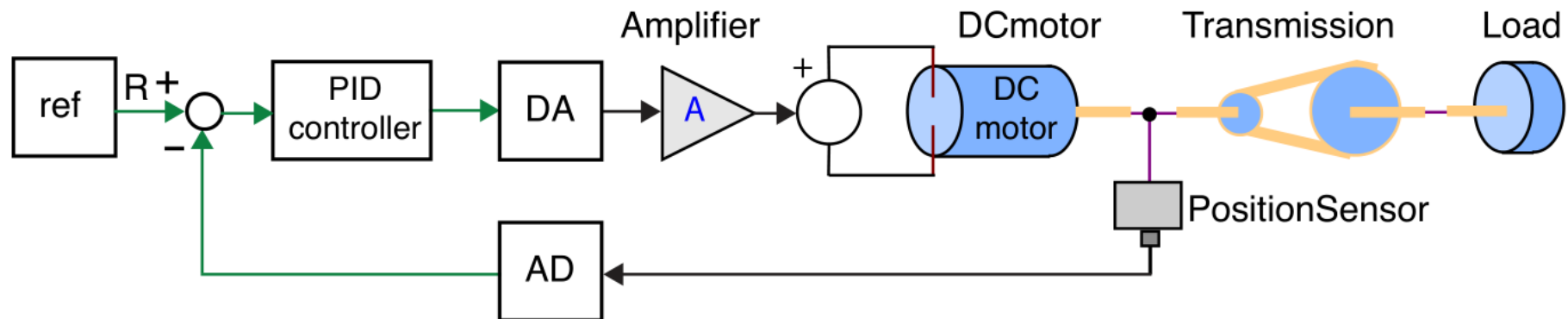
# Discrete

Using certain functions will make a block to run in discrete-time.

- sample: output of the function is discrete time, the input is continuous time
- hold: the output is continuous time, the input of the function is discrete time



# Sampling

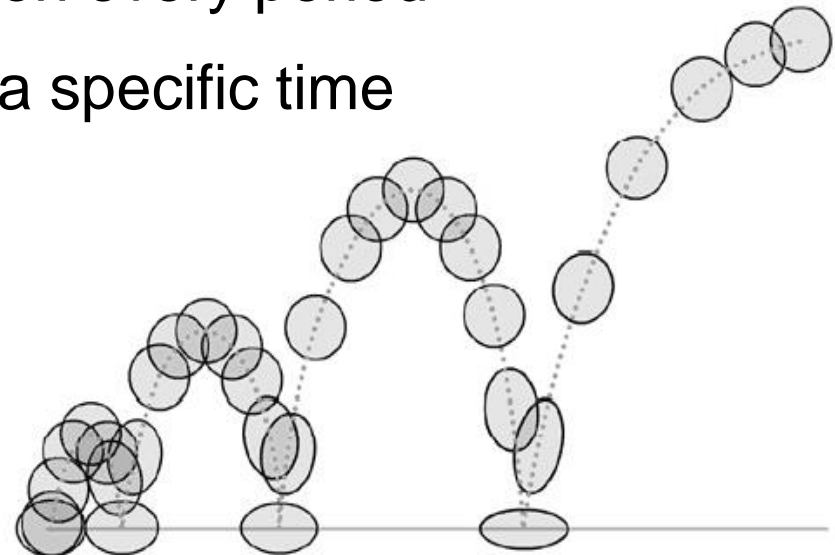




# Events

Force integration method to do a calculation at a specific point:

- event: find a zero crossing
- eventup: find a zero crossing from negative to positive
- eventdown: find a zero crossing from positive to negative
- frequencyevent: do a calculation every period
- timeevent: do a calculation at a specific time





# 20-sim Tutorial

Ken Pierce, Newcastle University



AARHUS  
UNIVERSITY

