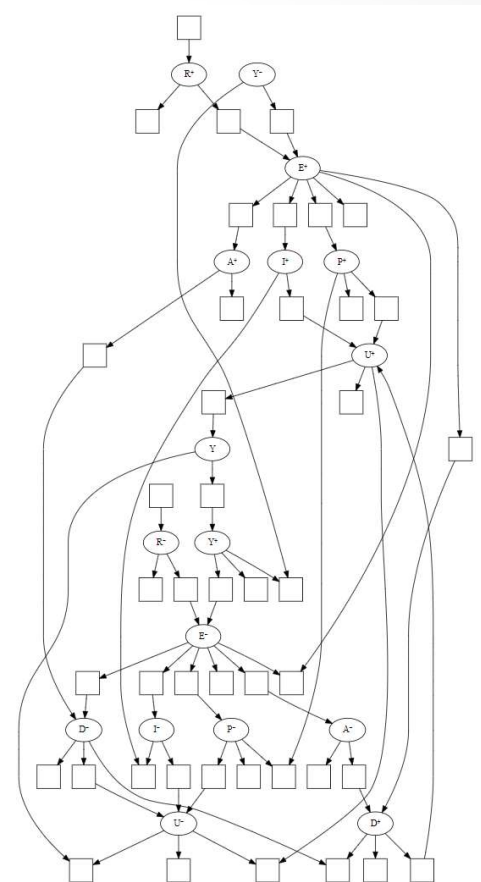


# Programming Chemical Reaction Networks in Kaemika

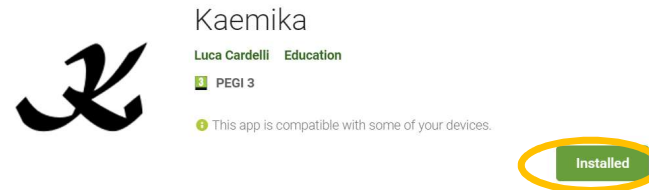


Luca Cardelli, University of Oxford  
Future of Computing  
2019-07-04 Porto

# Install Kaemika

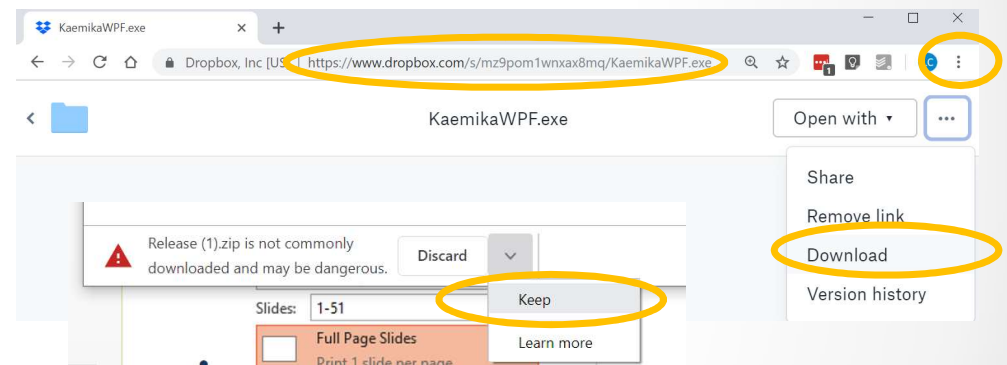
- Android version:

- Search "Kaemika" in the Play Store
- <https://play.google.com/store/apps/details?id=com.kaemika.Kaemika>



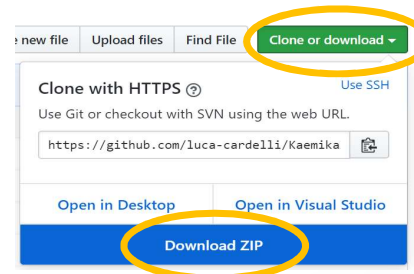
- Windows version (Dropbox): (runs without installation)

- Download & Unzip:  
<https://www.dropbox.com/s/qxity2e9hw4fw5c/Release.zip>
- Run Release\KaemikaWPF.exe



- Windows version (Github, probably need account):

- <https://github.com/luca-cardelli/KaemikaXM>
- Download & Unzip
- Run ...\KaemikaXM-master\KaemikaWPF\bin\Release\KaemikaWPF.exe



# Infinite Loop #1

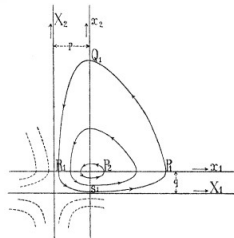
- The first ever interesting "chemical algorithm" that had nothing to do with actual chemicals

UNDAMPED OSCILLATIONS DERIVED FROM THE LAW OF MASS ACTION.

BY ALFRED J. LOTKA.  
Received June 2, 1920.

$$\frac{dX_1}{dt} = a_1 X_1 - b_1 X_1 X_2$$

$$\frac{dX_2}{dt} = a_2 X_1 X_2 - b_2 X_2$$



```
x1 -> x1 + x1      {1} // prey reproduces
x1 + x2 -> x2 + x2 {1} // predator eats prey
x2 -> #            {1} // predator dies
```

The first single, homogeneous oscillating chemical reaction was discovered accidentally by Bray [6] in 1921.

- First theoretical proof of oscillation, 1920 [Lotka]
- First experimental (accidental) chemical oscillator, 1921 [Bray]
- Ignored until the BZ reaction (accidental) discovery, 1958 [Belousov–Zhabotinsky]
- First non-accidentally-discovered chemical oscillator, 1981 [De Kepper]
- First protein/ATP-only oscillator, 2005 [Nakajima et al.]
- First DNA-only oscillator, 2017 [Srinivas et al.] (a version of Lotka's)

# Lotka Volterra

- Try removing the prey (x1) (set prey reproduction rate, reaction #1, to 0)
- Try removing the predators (x2) (set predation rate, reaction #2, to 0)
- Try making predators immortal (set predator dieout rate, reaction #3, to 0)
- Try doubling the predation rate (prey go down, but predators too!)
- Try doubling the prey reproduction rate (prey go up, but predators go up more!)

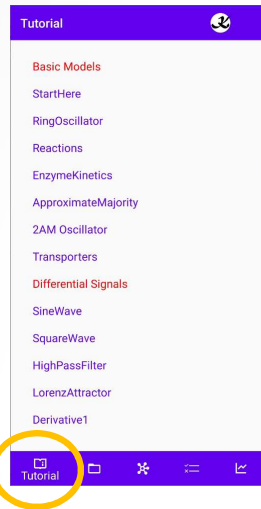
The screenshot shows the Kaemika mobile app interface. On the left is a navigation menu with categories like 'Basic Models', 'Reactions', and 'Differential Signals'. The 'LotkaVolterra' item is highlighted. The main screen displays the LotkaVolterra tutorial code and a corresponding plot of prey (x1, red) and predator (x2, green) populations over time. The plot shows oscillatory behavior. At the bottom, there are control buttons for 'Tutorial', 'Network', and 'Chart', with a play button highlighted.

The screenshot shows the Kaemika desktop application interface. The 'LotkaVolterra' tutorial is open, displaying the same code and plot as the mobile version. The 'Play' button is highlighted in yellow. The plot shows the prey (x1, red) and predator (x2, green) populations over time, exhibiting oscillatory behavior. The interface includes a menu bar, a toolbar with buttons like 'Copy', 'Paste', and 'Play', and a parameter control panel.

# Interaction (Android)



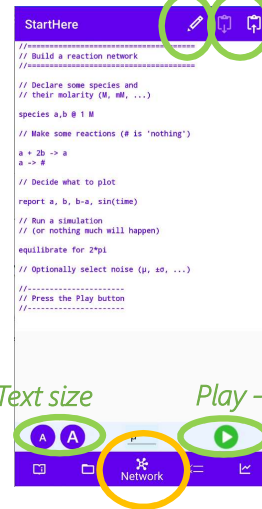
Splash!



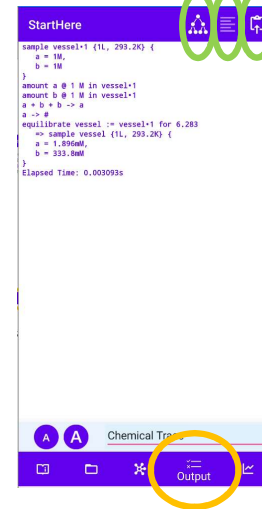
Tutorial Examples and Docs



User files



Current model



General output

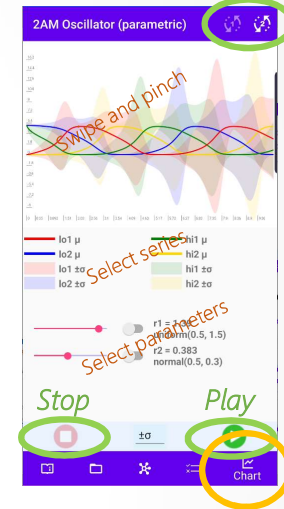
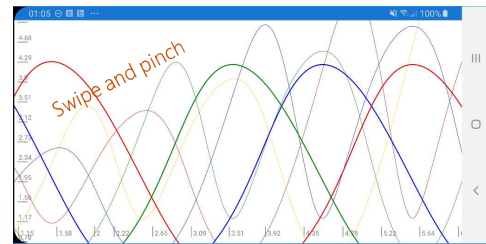


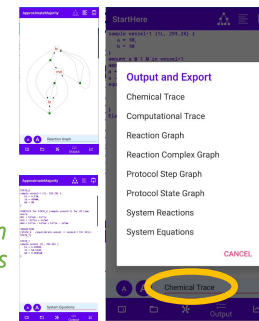
Chart output



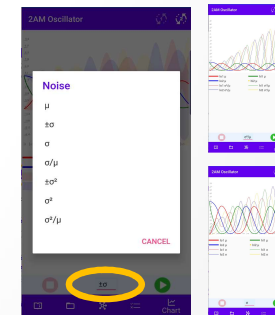
Rotate

Reaction and protocol graphs

Hybrid system kinetics



..and export



..and noise/LNA

Fano factor

stdev

Add new

Edit tutorials  
Clipboard paste/copy

Text outputs  
Graph outputs  
Export text or GraphViz

Solvers

swipe and pinch

select series  
select parameters

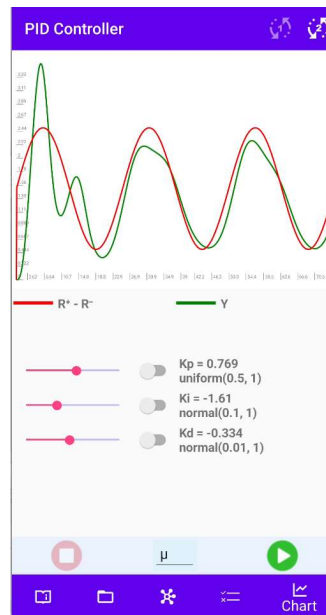
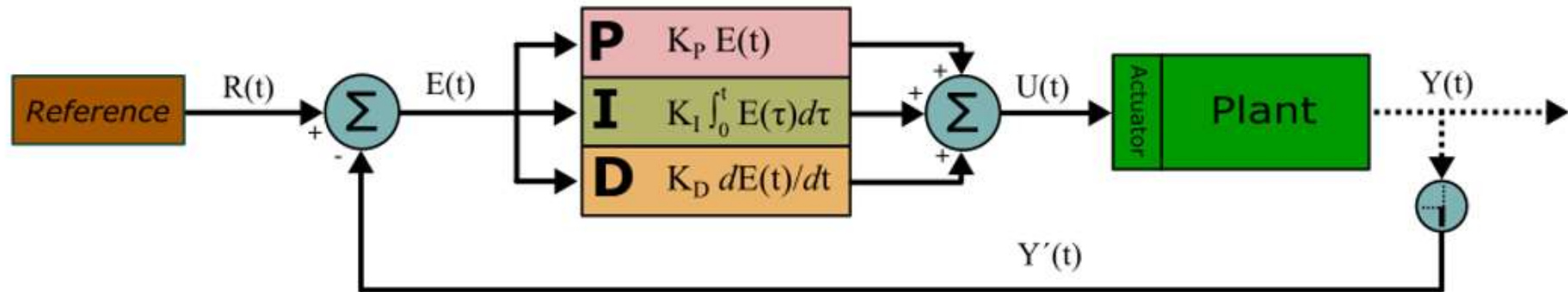
Stop Play

# Interaction (Windows)

The screenshot shows the Kaemika software interface with several key areas highlighted:

- Top Bar:** Includes a 'Tutorial' dropdown menu, 'Copy' and 'Paste' buttons, and a 'Sup' dropdown menu.
- Control Buttons:** 'Play', 'Continue', and 'Stop' buttons are located in the top right area.
- Stochastic Options:** A set of radio buttons for 'LNA',  $\pm\sigma$ ,  $\sigma/\mu$ ,  $\pm\sigma^2$ , and  $\sigma^2/\mu$ .
- Text Output:** A 'Chemical Trace' and 'Computational Trace' radio button set.
- Main Editing Area:** A large text area containing code for building a reaction network, declaring species, and running a simulation.
- Text Output Window:** A separate window showing simulation results, including species concentrations (a = 1.896mM, b = 333.8mM) and elapsed time.
- Control Panel:** Features an 'Export' dropdown, 'Copy' button, 'Parameters' button, 'Solvers' dropdown (set to RK547M), 'axes' and 'grid' checkboxes, 'alpha' dropdown (set to 32), 'Snap' button, and 'Legend' button.
- Graph Output:** A plot of Molarity (M) vs Time (s) showing curves for species 'a', 'b', 'b-a', and 'sin(time)'. A tooltip shows 'sin(time) [x = 3.723, y = -0.5493]'. A 'Zoom and scroll' label is also present.
- Legend:** A legend on the right side of the graph identifies the four data series.
- Other Outputs:** A box lists 'export to tools', 'export to GraphViz', 'analyze model', and 'copy text output'.

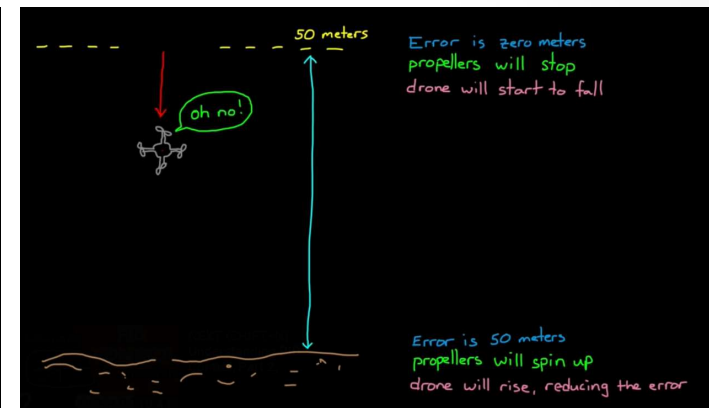
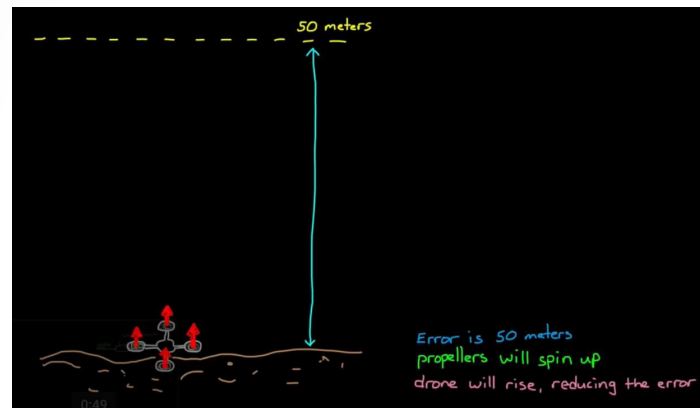
# PID Controller



# MATLAB YouTube Video

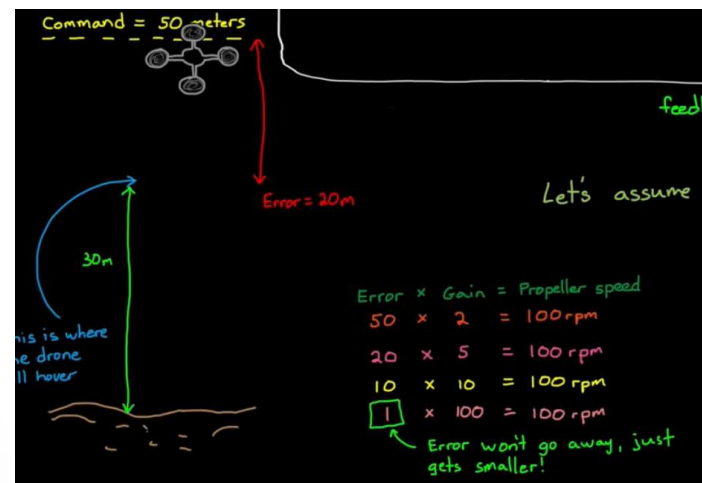
<https://www.youtube.com/watch?v=wkfEZmsQqiA>

On/Off  
Control:



Proportional  
Control:

Propeller speed = Error \* Gain  
hovering speed => Error =/= 0

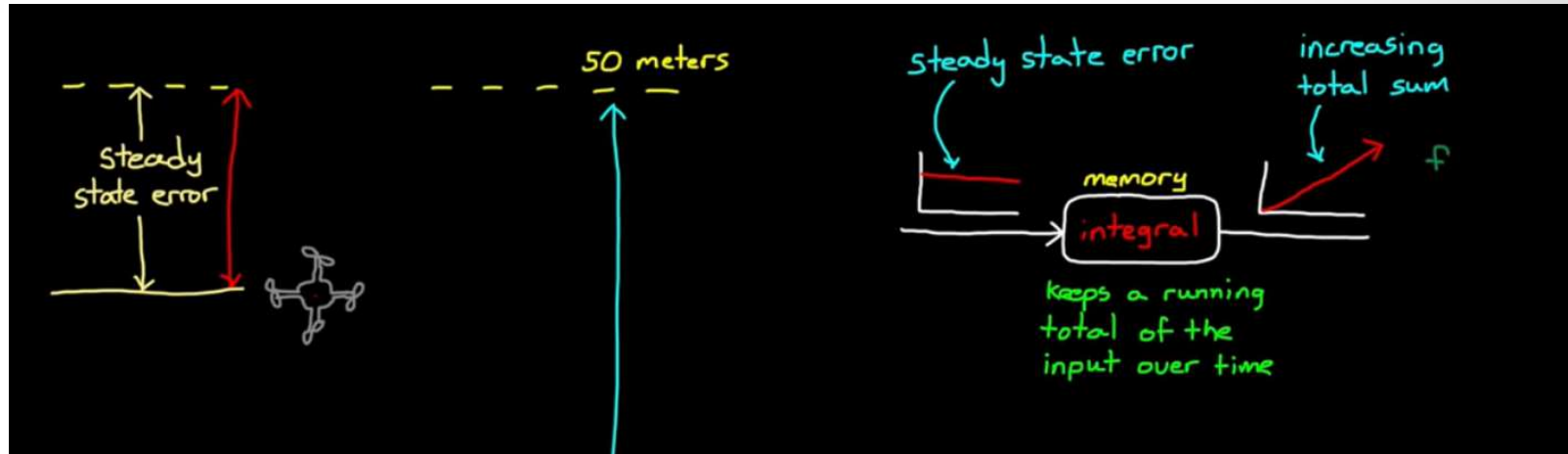




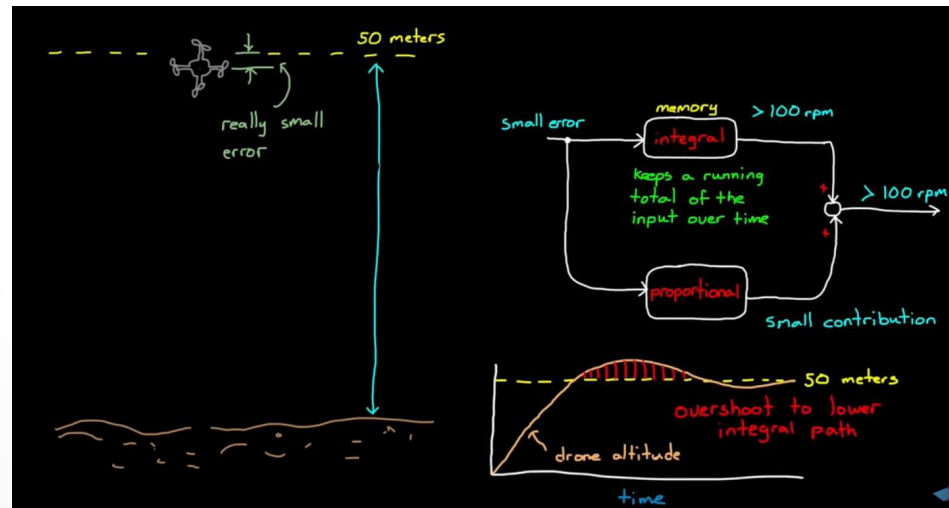
# MATLAB YouTube Video

## Proportional + Integral Control:

removes the steady state error (eventually)



but may overshoot:



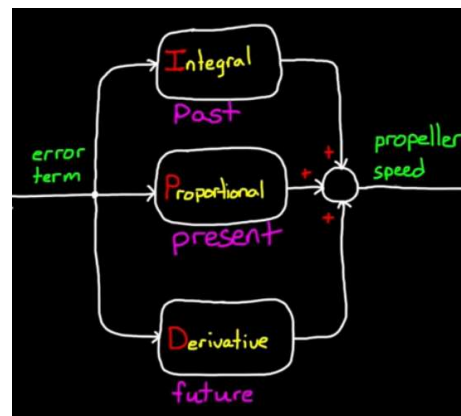
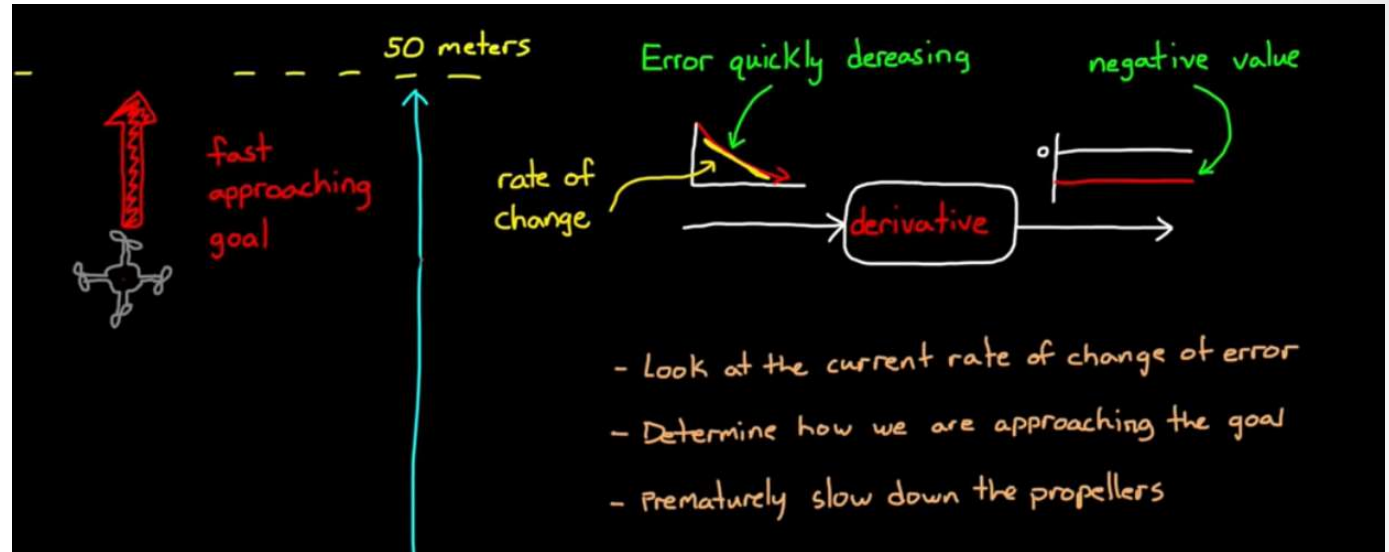
# MATLAB YouTube Video

Proportional  
+ Integral  
+ Differential  
Control:

if the error is *decreasing* fast  
a large *negative* derivative  
will be added

slowing down the ascent  
and preventing overshooting

PID: present + past + future



# Positive Arithmetic

```
// copy
species a @ 2 M
species a' @ 0 M
report a, a'

a -> a + a'
a' -> #

equilibrate for 5
```

$$\partial a = 0$$
$$\partial a' = a - a'$$

$$\partial a' = 0 \Rightarrow a' = a$$

Try increasing the **a level**  
How much longer will it  
take to reach steady state?

```
// add
species a @ 2 M
species b @ 3 M
species c @ 0 M
report a, b, c

a -> c
b -> c

equilibrate for 5
```

```
// copy and add
species a @ 2 M
species b @ 3 M
species c @ 0 M
report a, b, c

a -> a + c
b -> b + c
c -> #

equilibrate for 5
```

$$\partial a = 0$$
$$\partial b = 0$$
$$\partial c = a + b - c$$

$$\partial c = 0 \Rightarrow c = a + b$$

# Positive Arithmetic

```
//mult
species a @ 2 M
species b @ 3 M
species c @ 0 M
report a, b, c

a + b -> a + b + c
c -> #

equilibrate for 5
```

$$\begin{aligned}\partial a &= 0 \\ \partial b &= 0 \\ \partial c &= a * b - c\end{aligned}$$

$$\partial c = 0 \Rightarrow c = a * b$$

```
//div
species a @ 2 M
species b @ 3 M
species c @ 0 M
report a, b, c

a -> a + c
b + c -> b

equilibrate for 5
```

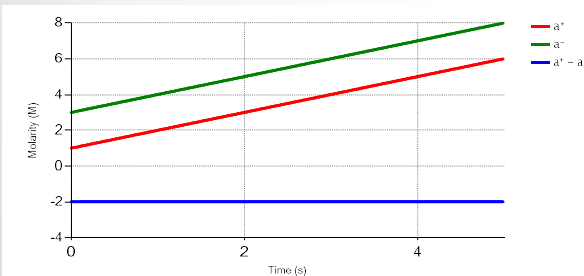
$$\begin{aligned}\partial a &= 0 \\ \partial b &= 0 \\ \partial c &= a - b * c\end{aligned}$$

$$\begin{aligned}\partial c = 0 &\Rightarrow c = a / b \\ (b = 0 &\Rightarrow c \rightarrow \infty)\end{aligned}$$

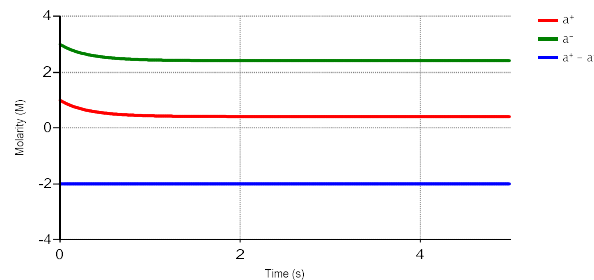
# Differential Signals

But ERRORS in the PID controller can be positive or negative, while concentrations can only be positive.

Solution: encode an integer number as the difference of two natural numbers (concentrations)



Without normalization  
(they keep growing)



With normalization  
(we are still producing  $a^-$ , so it does not go to zero, but it stabilizes)

```
// a = a+ - a-
```

```
species a+ @ 1 M  
species a- @ 3 M  
report a+, a-  
report a+ - a-
```

```
# -> a+
```

```
# -> a-
```

```
// normalization:  
// a+ + a- -> #
```

```
equilibrate for 5
```

# Addition of Differential Signals

$$\begin{aligned} a + b &= (a^+ - a^-) + (b^+ - b^-) \\ &= (a^+ + b^+) - (a^- + b^-) \\ &= c^+ - c^- \\ &= c \end{aligned}$$

```
species a+ @ 2 M
```

```
species a- @ 0 M
```

```
species b+ @ 0 M
```

```
species b- @ 3 M
```

```
species c+ @ 0 M
```

```
species c- @ 0 M
```

```
report a+ - a-, b+ - b-, c+ - c-
```

```
a+ -> c+
```

```
b+ -> c+
```

```
a- -> c-
```

```
b- -> c-
```

```
equilibrate for 5
```

# Subtraction of Differential Signals

$$\begin{aligned} a - b &= (a^+ - a^-) - (b^+ - b^-) \\ &= (a^+ + b^-) - (a^- + b^+) \\ &= c^+ - c^- \\ &= c \end{aligned}$$

```
species a+ @ 2 M
```

```
species a- @ 0 M
```

```
species b+ @ 0 M
```

```
species b- @ 3 M
```

```
species c+ @ 0 M
```

```
species c- @ 0 M
```

```
report a+ - a-, b+ - b-, c+ - c-
```

```
a+ -> c+
```

```
b- -> c+
```

```
a- -> c-
```

```
b+ -> c-
```

```
equilibrate for 5
```

# Multiplication of Differential Signals

$$\begin{aligned} a * b &= (a^+ - a^-) * (b^+ - b^-) \\ &= a^+ * b^+ - a^+ * b^- - a^- * b^+ + a^- * b^- \\ &= (a^+ * b^+ + a^- * b^-) - (a^+ * b^- + a^- * b^+) \\ &= c^+ - c^- \\ &= c \end{aligned}$$

At this point we would want to use some "subroutines", since we have already seen how to multiply and add positive quantities and we need to do a whole bunch of those.

Fortunately we will not need this multiplication for the PID controller, but we will still need to modularize reactions.

```
species a+ @ 2 M
species a- @ 0 M

species b+ @ 0 M
species b- @ 3 M

species c+ @ 0 M
species c- @ 0 M

...

equilibrate for 5
```



# Modular Chemical Programs

```
function signal(number n) {
  define
    species n+ @ pos(n) M
    species n- @ pos(-n) M
    n+ + n- -> #
  yield
    [n+, n-]
}

function copy([species a+ a-]) {
  define
    [species b+ b-] = signal(0)
    a+ -> a+ + b+; b+ -> #
    a- -> a- + b-; b- -> #
  yield
    [b+, b-]
}
```

```
function add([species a+ a-], [species b+ b-]) {
  define
    [species c+ c-] = signal(0)
    a+ -> c+; b+ -> c+
    a- -> c-; b- -> c-
  yield
    [c+, c-]
}

function sub([species a+ a-], [species b+ b-]) {
  define
    [species c+ c-] = signal(0)
    a+ -> c+; b- -> c+
    a- -> c-; b+ -> c-
  yield
    [c+, c-]
}
```

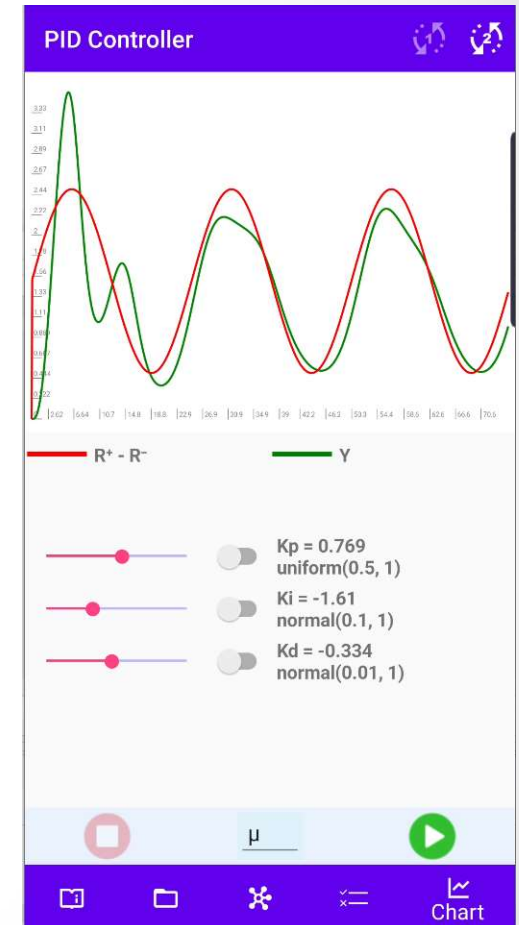
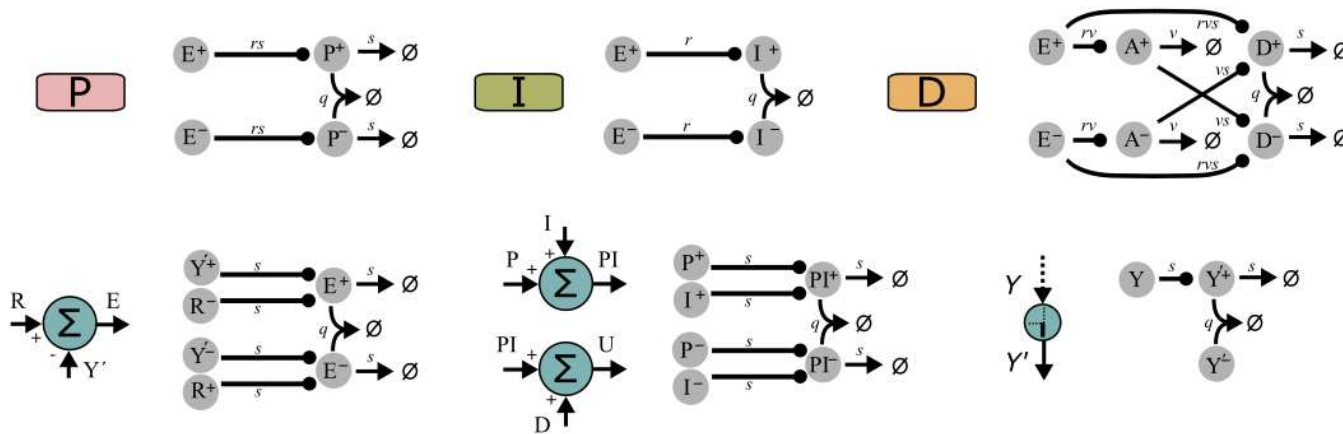
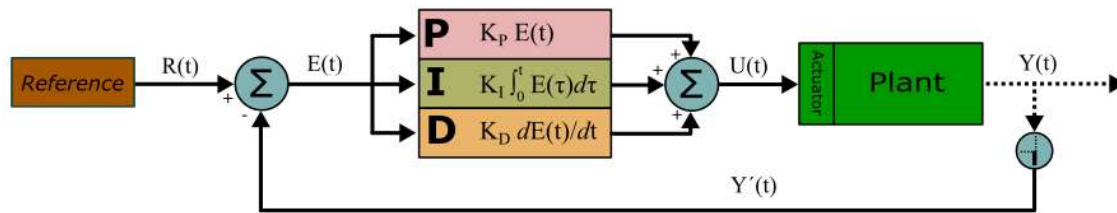
# Modular Chemical Programs

```
list a = signal(3)    // [a+, a-]  
list b = signal(-2)  // [b+, b-]  
list d = copy(a)     // [d+, d-]  
list c = add(a, b)   // [c+, c-]  
list e = sub(d, c)   // [e+, e-]
```

```
report a(0) - a(1) as "a"  
report b(0) - b(1) as "b"  
report c(0) - c(1) as "c"  
report d(0) - d(1) as "d"  
report e(0) - e(1) as "e"
```

```
equilibrate for 5
```

# PID Controller



# Proportional Block

- Amplify the error  $E = (E^+ - E^-)$  into  $P = (P^+ - P^-)$  by a tunable "gain"  $K_p$

```
//----- Proportional Block -----  
network PBlock (species E+ E- P+ P-, number Kp) {  
    E+ -> E+ + P+      {precision * Kp}  
    E- -> E- + P-      {precision * Kp}  
    P+ -> #            {precision}  
    P- -> #            {precision}  
    P- + P+ -> #      {precision}  
}
```

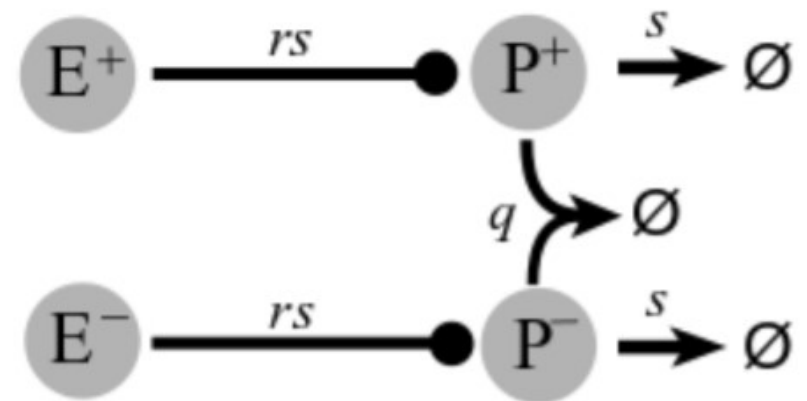
You may recognize the pattern:

$E^+$  is copied into  $P^+$  (reaction 1 & 3)

$E^-$  is copied into  $P^-$  (reaction 2 & 4)

$P^+ - P^-$  is normalized (reaction 5)

But the "copying" adds an amplification  $K_p$   
(rates of 1 & 2)



# Unit Testing the P-Block

- Change  $K_p$
- Change the "DSignal" function

# Integral Block

- Integrate the error  $\mathbf{E} = (\mathbf{E}^+ - \mathbf{E}^-)$  into  $\mathbf{I} = (\mathbf{I}^+ - \mathbf{I}^-)$  with a tunable "gain"  $K_i$

```
//----- Integral Block -----  
network IBlock (species E+ E- I+ I-, number Ki) {  
    E+ -> E+ + I+           {Ki}  
    E- -> E- + I-           {Ki}  
    I- + I+ -> #             {precision}  
}
```

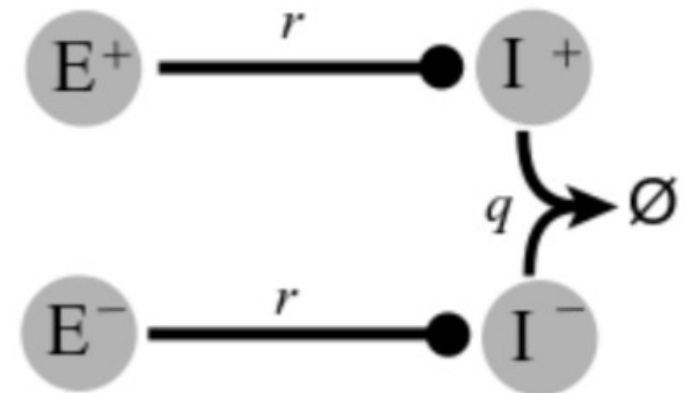
This is even easier:

$E^+$  accumulates into  $I^+$  (reaction 1)

$E^-$  accumulates into  $I^-$  (reaction 2)

$I^+ - I^-$  is normalized (reaction 3)

But the "accumulation" adds an amplification  $K_i$   
(rates of 1 & 2)



# Unit Testing the I-Block

- Change Ki
- Change the "DSignal" function

# Derivative Block

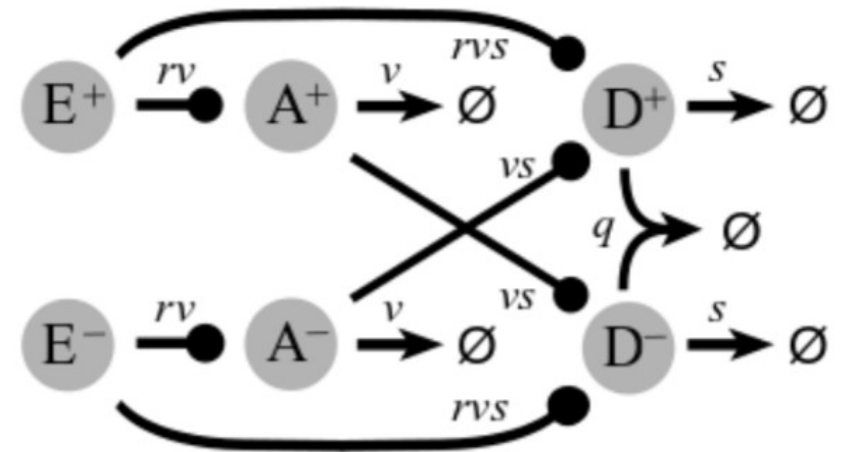
- Differentiate the error  $E = (E^+ - E^-)$  into  $D = (D^+ - D^-)$  with a tunable "gain"  $K_d$

```
//----- Derivative Block -----
network DBlock(species E+ E- D+ D-, number Kd) {
  species A+, A- @ OM // D block auxiliary species
  E+ -> E+ + A+ {precision}
  A+ -> # {precision}
  E- -> E- + A- {precision}
  A- -> # {precision}
  E+ -> E+ + D+ {precision*precision*Kd}
  A- -> A- + D+ {precision*precision*Kd}
  D+ -> # {precision}
  E- -> E- + D- {precision*precision*Kd}
  A+ -> A+ + D- {precision*precision*Kd}
  D- -> # {precision}
  D+ + D- -> # {precision}
}
```

## PID Control of Biochemical Reaction Networks

Max Whiby<sup>1</sup>, Luca Cardelli<sup>1</sup>, Marta Kwiatkowska<sup>1</sup>, Luca Laurenti<sup>1</sup>, Mirco Tribastone<sup>2</sup>, Max Tschaikowski<sup>3</sup>

- $E^+, E^-$  is copied into  $A^+, A^-$
- $E^+, E^-$  is also copied into  $D^+, D^-$
- $A^+, A^-$  is swap-copied (negated-summed) into  $D^+, D^-$
- So  $D$  is the difference of two copies of  $E$  taken at different times  $t$  and  $t+s$  (because  $E \rightarrow D$  is faster than  $E \rightarrow A \rightarrow D$ )
- Appropriate rates ensure that
 
$$D(t) = (E(t) - E(t-s))/s$$
 which converges to the derivative for  $s \rightarrow 0$  (where  $s$  is a rate)





# Derivative Block

- Informal argument

- $\partial A^+ = sE^+ - sA^+$
- $\partial A^- = sE^- - sA^-$
- $\partial D^+ = ks^2E^+ + ks^2A^- - sD^+ - sD^+D^-$
- $\partial D^- = ks^2E^- + ks^2A^+ - sD^- - sD^+D^-$
  
- $\partial(D^+ - D^-) = ks(sE^+ - sE^-) + ks(sA^- - sA^+) - s(D^+ - D^-)$
- $\partial(D^+ - D^-) = ks(sE^+ - sA^+) - ks(sE^- - sA^-) - s(D^+ - D^-)$
- $\partial(D^+ - D^-) = ks\partial(A^+ - A^-) - s(D^+ - D^-)$

- at "steady state" (when  $\partial A^+ = \partial A^- = \partial(D^+ - D^-) = 0$ )
- $A^+ = E^+$
- $A^- = E^-$
- $D^+ - D^- = k\partial(E^+ - E^-)$

- (but  $\partial(D^+ - D^-)$  may never reach steady state, so this needs a more formal argument)

```
//----- Derivative Block -----  
network DBlock(species E+ E- D+ D-, number Kd) {  
  species A+, A- @ OM // D block auxiliary species  
  E+ -> E+ + A+ {s}  
  A+ -> # {s}  
  E- -> E- + A- {s}  
  A- -> # {s}  
  E+ -> E+ + D+ {s*s*k}  
  A- -> A- + D+ {s*s*k}  
  D+ -> # {s}  
  E- -> E- + D- {s*s*k}  
  A+ -> A+ + D- {s*s*k}  
  D- -> # {s}  
  D+ + D- -> # {s}  
}
```

# Unit Testing the D-Block

- Increase the precision (say, to 100) for better fidelity
- Change the function to differentiate
  - $2 \cdot \text{time}$ , whose derivative is constant 2
  - $\text{time}^2$ , whose derivative is  $2 \cdot \text{time}$  (slope 2)
  - $\exp(\text{time})$ , whose derivative is  $\exp(\text{time})$ ! (STOP it before it crashes!)
  - the *second* derivative of  $\sin(\text{time})$  by using two D-Blocks

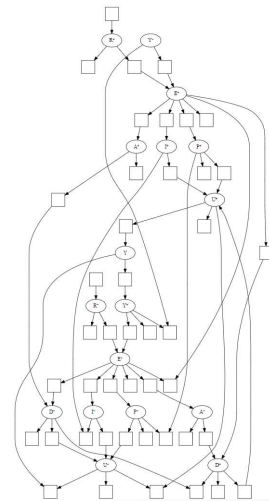
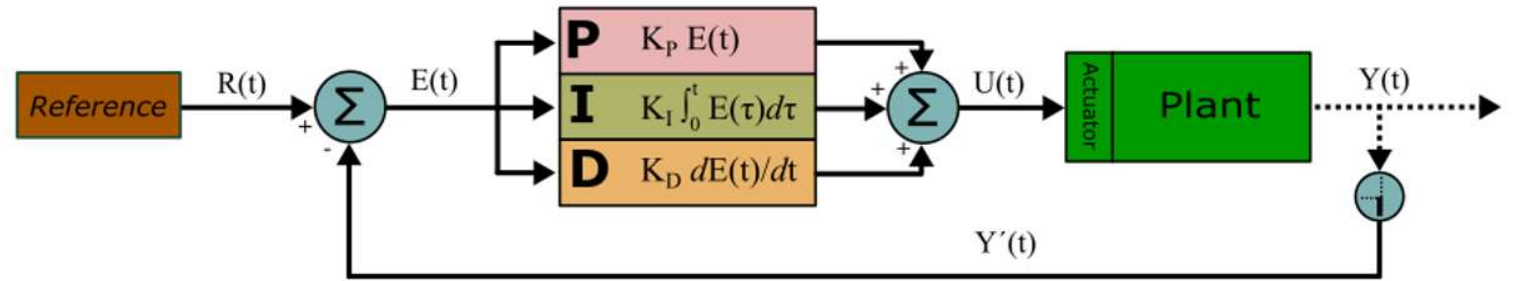
# Finally, the whole PID controller

```
//=====
// PID Controller Block
//=====
```

```
network PIDController(
  species R+ R-,
  number Kp Ki Kd,
  network Plant){
```

```
species E+,E-,P+,P-,I+,I-, D+,D-,U+,U-,Y,Y+,Y- @ OM
```

```
PBlock(E+, E-, P+, P-, Kp)
IBlock(E+, E-, I+, I-, Ki)
DBlock(E+, E-, D+, D-, Kd)
SumBloc(P+, P-, I+, I-, D+, D-, U+, U-)
Plant(U+, U-, Y)
DualRail(Y, Y+, Y-)
SubBlock(R+, R-, Y+, Y-, E+, E-)
}
```



# The Trivial Plant

```
network Plant(species U+ U- Y) {  
  U+ -> U+ + Y  
  U- + Y -> U-  
}
```

- U<sup>+</sup> increases the output Y
- U<sup>-</sup> decreases the output Y
- However it is not symmetrical:

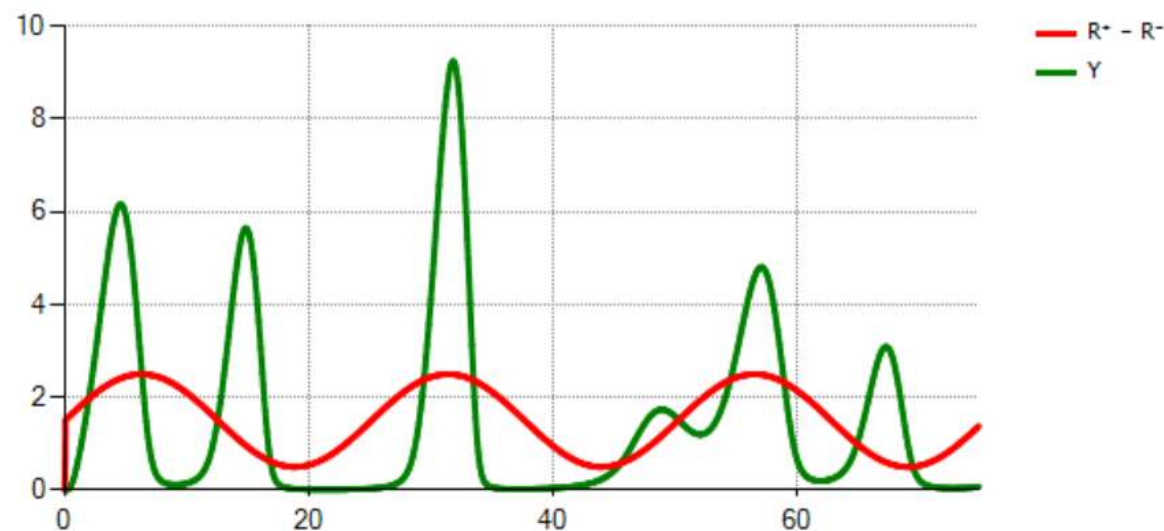
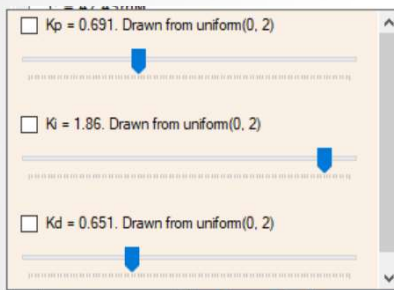
$$\partial Y = U^+ - Y * U^-$$

so, although we can control the plant, we do not have direct control of Y and the control task is still mildly non-trivial

# Testing the Controller

- Oscillating reference signal
- Just click the Play button a few times

The parameters  $K_p$ ,  $K_i$ ,  $K_d$  are drawn from uniform random distributions. They can be individually frozen by checking the checkboxes



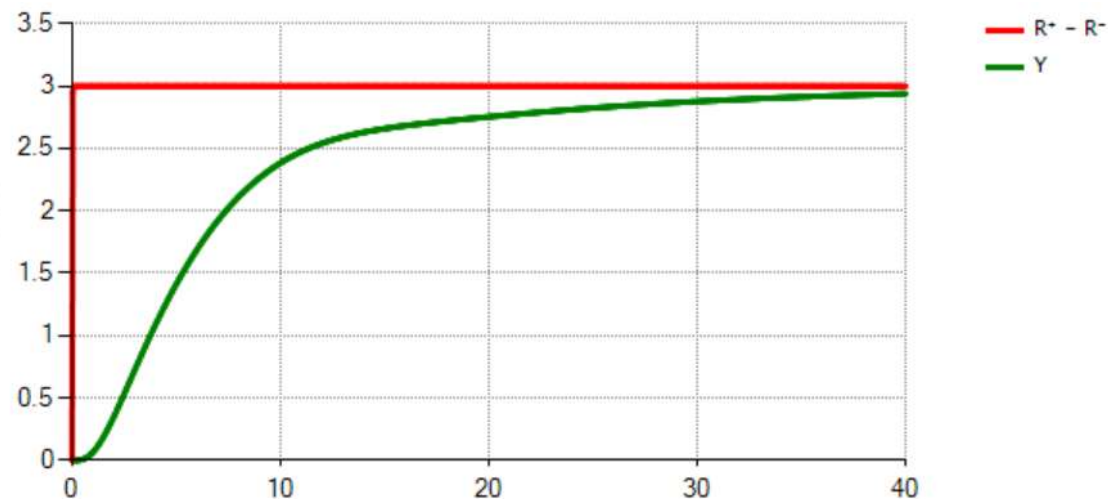
- Try to find and freeze some good parameters

# Testing the Controller

- Constant reference signal
- This is easier

Try changing the parameters by hand in this line (these are already pretty good):

```
PIDController(R+,R-, Y, 0.1, 0.02, 0.02, Plant)
```



- Likely, you will need a different set of parameters here.

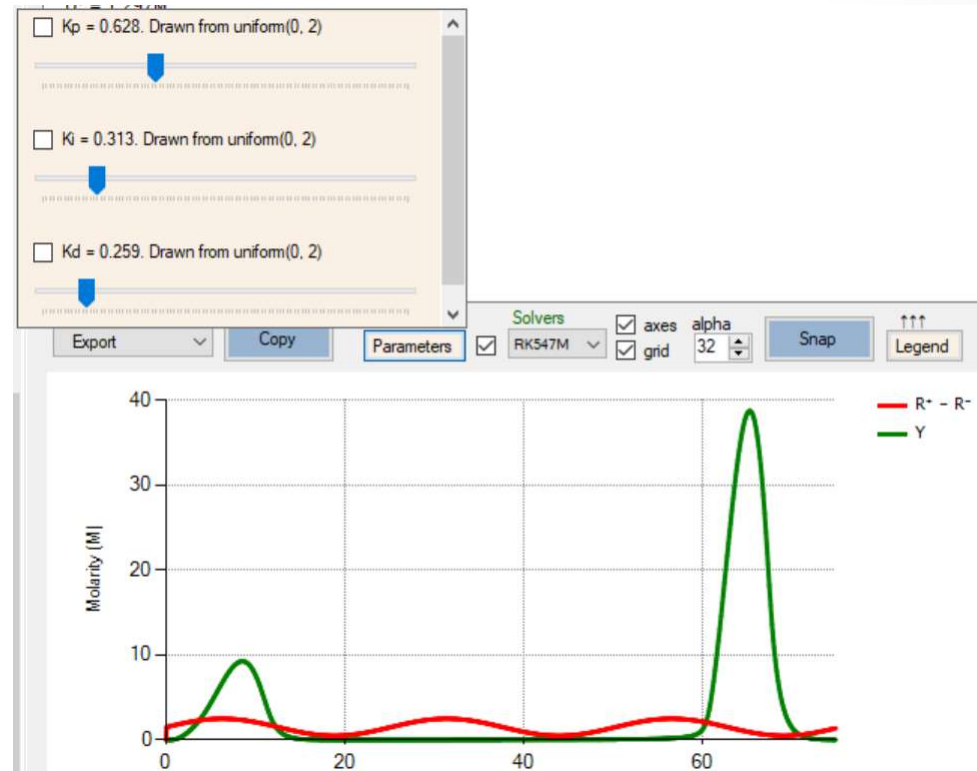
# Biochemical Plant

```
network Plant(species U+ U- Protein) {
  species mRNA @ 0 M
  species microRNA @ 0 M
  U+ -> U+ + mRNA
  U- -> U- + microRNA

  mRNA -> mRNA + Protein
  Protein-> #
  mRNA + microRNA -> #
}

// OSCILLATING REFERENCE
number Kp =? uniform(0,2)
number Ki =? uniform(0,2)
number Kd =? uniform(0,2)

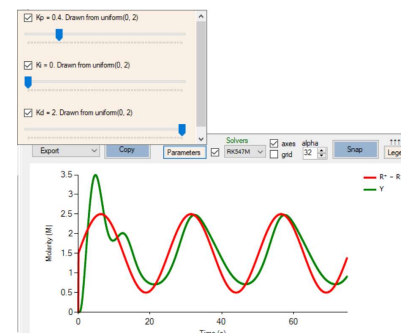
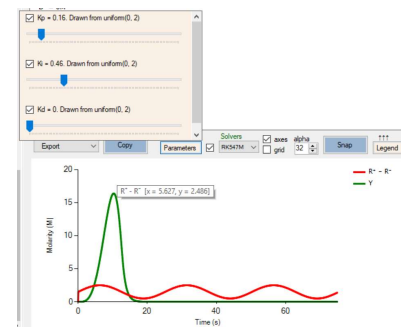
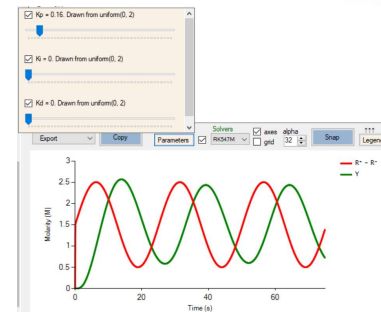
DSignal(R+,R-, fun(){1.5+sin(0.25*time)})
PIDController(R+,R-, Y, Kp, Ki, Kd, Plant)
```



- Random sampling: Not so easy!

# Biochemical Plant - my PD strategy

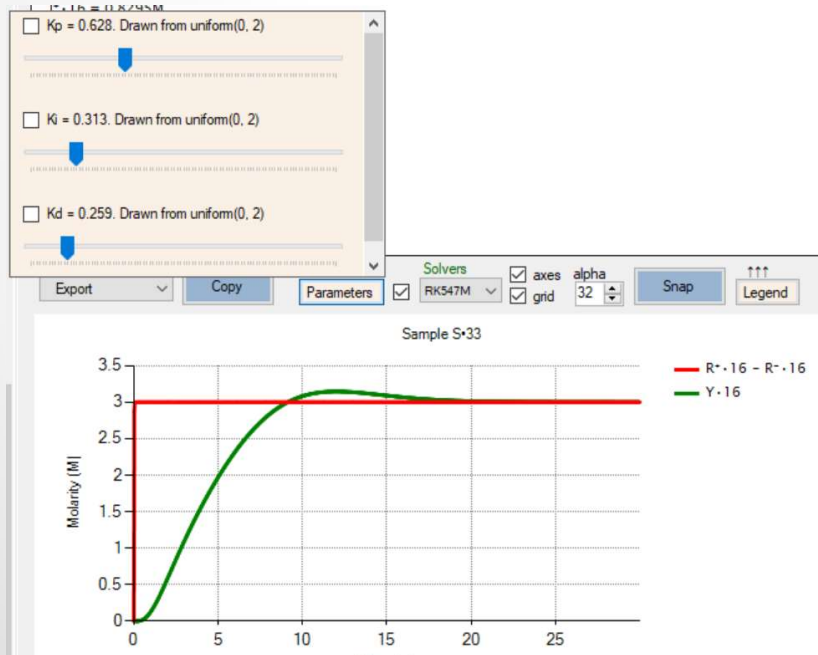
- 1. Adjust  $K_p$  until the amplitudes match (with  $K_i$ ,  $K_d$  zero)
- 2. Adjusting  $K_i$  does not seem to help at all, leave it at zero
- 3. Adjusting  $K_d$  gives better results to put the oscillation in phase





# Automatic Parameter Search

- Kaemika has a multi-dimensional gradient descent search (the "argmin" primitive, implementing the BFGS algorithm)
- Gradients (partial derivatives) must be provided for  $K_p$ ,  $K_i$ ,  $K_d$ ; see the "PIDController Optimization" tutorial about how to do that



Was asked to zero-out error already at time 20.  
So it overshoots a bit to get there in time.  
It is using each of P, I, and D.

Found (local) minimum in only 5 tries.

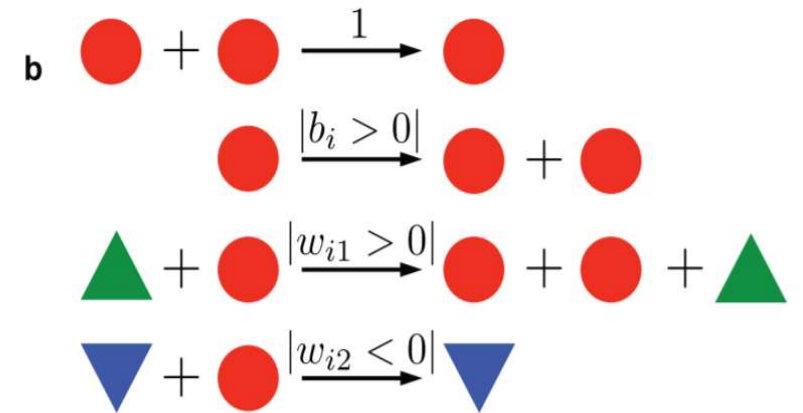
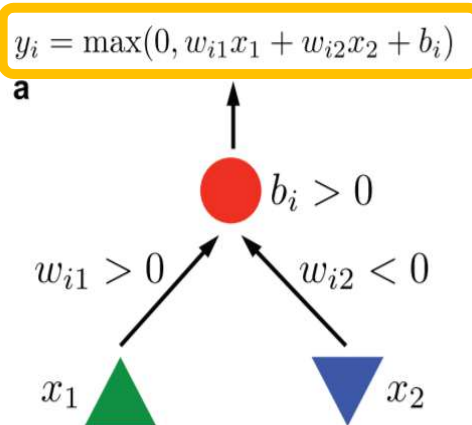
# Chemical Perceptron

species a @ 0.01 M  
 number bias = 0.3  
 number pw = 1.1  
 number nw = 0.9  
 species exc @ 1.5 M  
 species inh @ 0.4 M

a + a -> a  
 a -> a + a  
 exc + a -> a + a + exc  
 inh + a -> inh

{1}  
 {bias}  
 {pw}  
 {nw}

equilibrate for 40



**Nanoscale artificial intelligence: creating artificial neural networks using autocatalytic reactions**

Filippo Simini<sup>1</sup>

# Conclusions

- Programming chemistry is fun. But it is no fun without modularization! Chemical reactions provide a nice almost-high-level language if properly modularized.
- There will always be "cheaper" ways of implementing those programs by direct "low-level chemical hacking" (c.f. trade-off between high-level and assembly languages).
- But a compiler could optimize higher-level programs for specific architectures (e.g. DNA strand displacement).
- There are already higher-level languages. Synthetic biology "programs" (gene assemblies) can be compiled from libraries of standard parts into molecules (plasmids). Chemical reactions there figure prominently as an "intermediate language" between gene specification and analysis.
- Control of biochemical "plants" is a major issue in synthetic biology.

