Applied Programming

Numerical Differentiation
and
Digital Differentiators

More details in: U. Ascher and C. Grief, "A First Course in Numerical Methods", chapters 14, 15

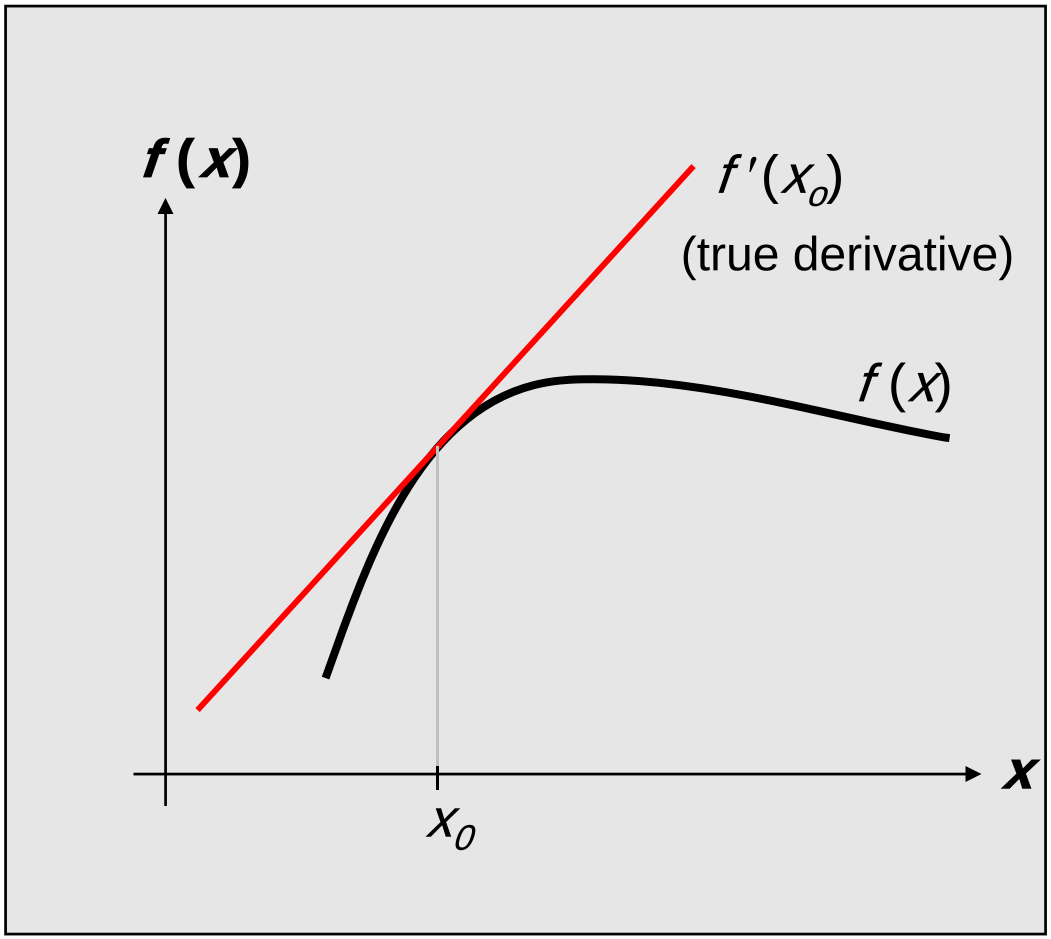
Why Numerical Differentiation ?

Analytical differentiation is:

- Difficult:
  - requires symbolic computations.
- Not feasible:
  - closed form of function not available.
- Not practical:
  - in many situations a numerical version of the derivative is preferred.
- Necessary:
  - ➢ Many sensors read position but we may want velocity (encoders)

Derivative: Geometric Interpretation

In the plane (2D), the derivative at a point x_o is the ***slope of the tangent*** to a curve *f(x)* ***at the point*** x_o



Derivative: Definition

Calculus:

- The derivative of a function *f(x)* at a point x_i defined as

$$f(x_i) = \lim_{h \to 0} \frac{f(x_i + h) - f(x_i)}{h}$$

- In many practical situations we do not know the function *f(x) explicitly*

- Our general objective is to ***estimate the derivative*** of the function *f(x)* ***at a desired point*** x_i

Numerical Differentiation

Numerical differentiation is required for:

I. Estimation of the derivative of a function available only at discrete set of points, e.g., $y_i = f(x_i), i=1..n$

   Example: Compute velocity using position encoder measurements.

II. Discretization of differential equations to simulate physical systems

   Example: Flight simulators

Numerical Differentiation in Estimation (I)

Objective

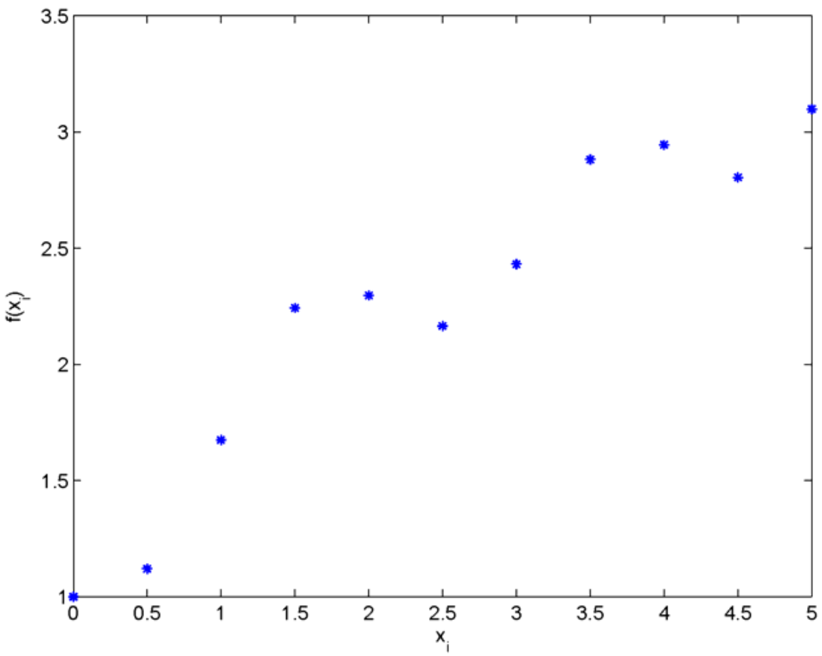
Find the derivative of a function *given only samples of the function* at discrete points, *e.g.*, $y_i = f(x_i), i=1..n$

- Sometimes we want to find the derivative at all points: $f'(x)$
- Sometimes we want the derivative at a given point: $f'(x^*)$

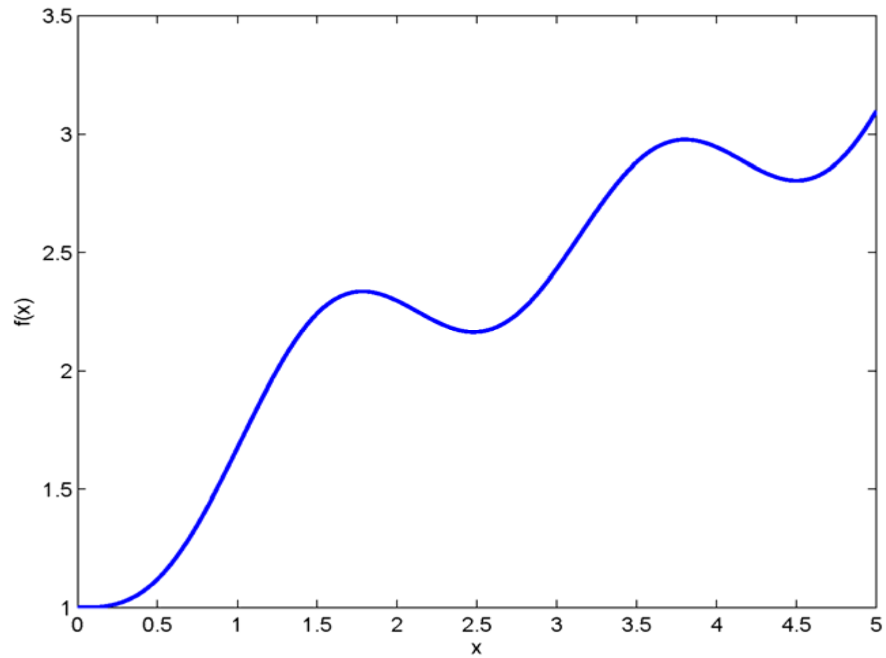
Warning: In most applications the samples of *f (x)* are noisy

Numerical Differentiation I

Problem: Find the derivative of a function ***given a finite number of points*** $\{(x_i, y_i)\}, y_i = f(x_i)$



Given data points



We are not given “the function”

Numerical Differentiation I

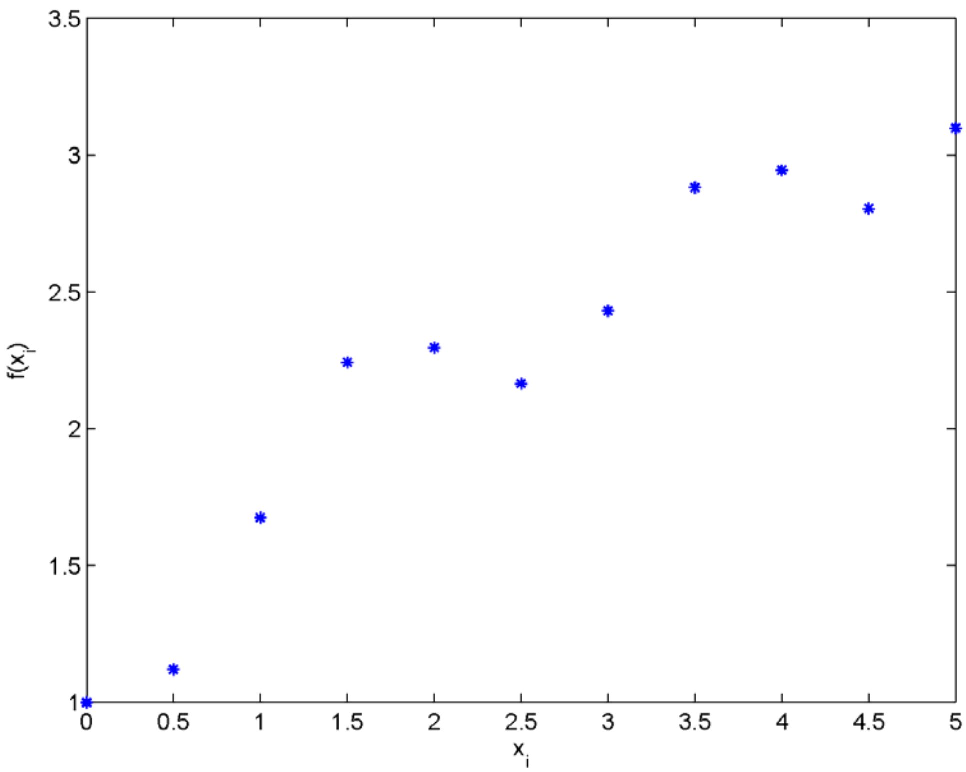
Given a finite set of points $\{(x_i, y_i)\}$, $y_i = f(x_i)$ ***estimate the derivative*** of the function

- One way to solve this problem is in two steps:
  1. *Interpolate* (or fit) a function to the data (*e.g.*, find a "model" for the data)

  2. *Differentiate* (analytically) the closed form of the interpolated (fitted) function.

Example: Interpolation/Differentiation

Example: Find the derivative of $f(x)$ given the points shown



Proposed Solution:

1. Perform a *cubic spline interpolation*

2. Take derivative of interpolated splines

Example: Interpolation/Differentiation

Results: (Plots on next slide)

- Spline Coefficients (d,c,b,a)

-1.5488	-0.1895	2.0000	0
1.5559	-2.5127	0.6489	0.7590
2.0461	-0.1788	-0.6968	0.6498
-1.2713	2.8903	0.6590	0.5125
-2.2372	0.9834	2.5958	1.4056
0.8653	-2.3724	1.9013	2.6698
2.2376	-1.0744	0.1779	3.1355
-0.6938	2.2819	0.7816	3.2355
-2.4108	1.2412	2.5432	4.1101
0.1565	-2.3750	1.9763	5.3907

$$d_i(x-x_i)^3 + c_i(x-x_i)^2 + b_i(x-x_i) + a_i$$

- Derivatives

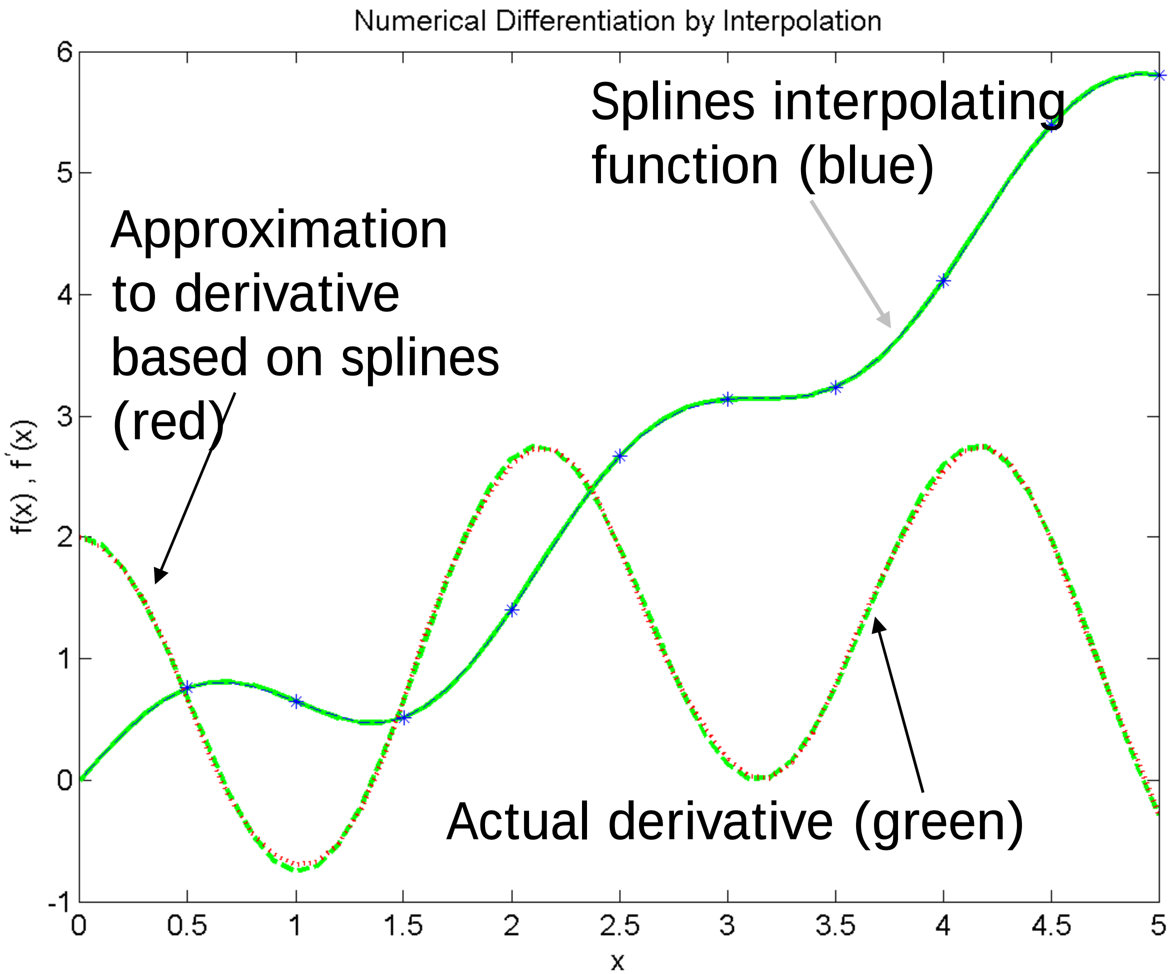
-4.6464	-0.3789	2.0000
4.6678	-5.0253	0.6489
6.1382	-0.3576	-0.6968
-3.8138	5.7807	0.6590
-6.7116	1.9668	2.5958
2.5959	-4.7448	1.9013
6.7127	-2.1489	0.1779
-2.0814	4.5638	0.7816
-7.2324	2.4824	2.5432

$$3d_i(x-x_i)^2 + 2c_i(x-x_i) + b_i$$

Advantage: We can find the *derivative at any point* in the range (not only at data points)

[note: derivative is a quadratic spline]

Example: Interpolation/Differentiation



Numerical Differentiation I

Given a finite set of points $\{(x_i, y_i)\}$, $y_i=f(x_i)$ estimate the derivative of the function

- If we only need to compute the derivative *at the data points* other alternatives could be more efficient.

- To derive algorithms for numerical differentiation we can start from the definition of derivative.

Numerical Differentiation at Points

Using the definition from calculus

$$f'(x_i) = \lim_{h \to 0} \frac{f(x_i + h) - f(x_i)}{h}$$

- We can *approximate derivative* at a point x_i by taking the ***difference between to adjacent*** values h units apart and dividing by h
  - ***h*** is a ***very small*** non-zero value

Numerical Differentiation

The proposed approximation has clear practical limitations

$$f'(x_i) = \frac{f(x_i + h) - f(x_i)}{h}$$

$h \to 0$ (for best accuracy)

1. Machine precision

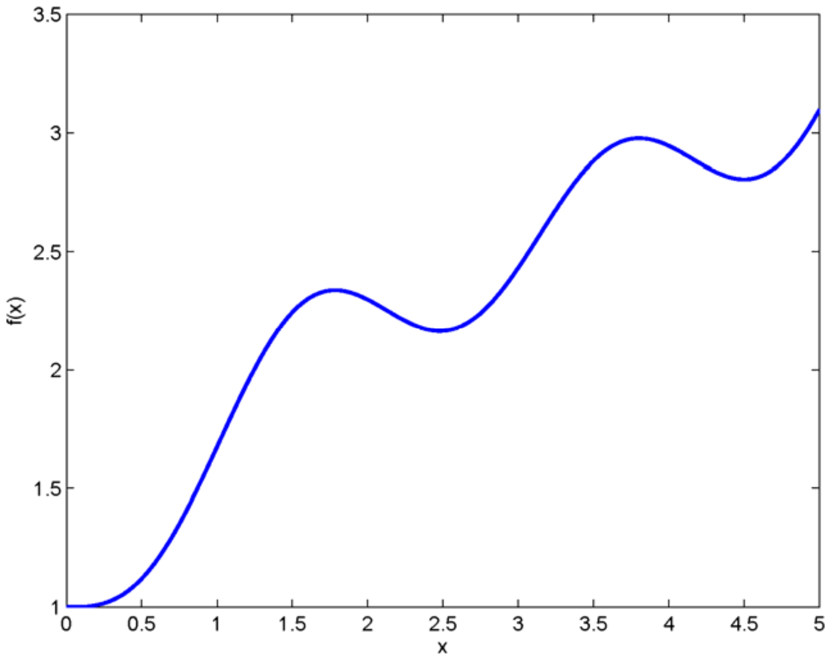
   $\Rightarrow$ *h* may not be made arbitrarily small

2. May not have control over *h*

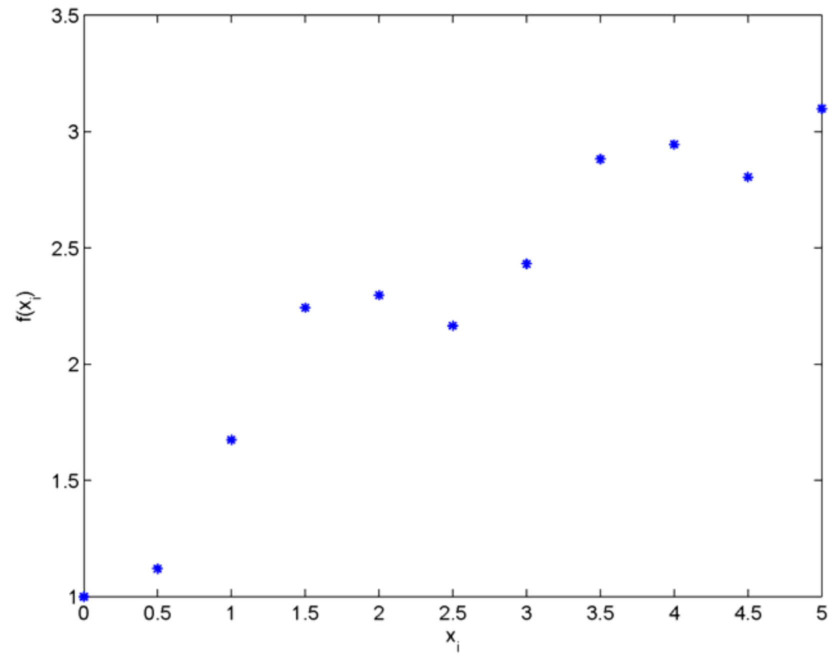
   $\Rightarrow$ *h* may be *fixed* (by the problem) and *too large* for accurate approximation

Problem to be Solved

How do we estimate the derivative at x_i from just a set of points $\{(x_i, y_i)\}$, $y_i = f(x_i)$ *[for a given h]* ?

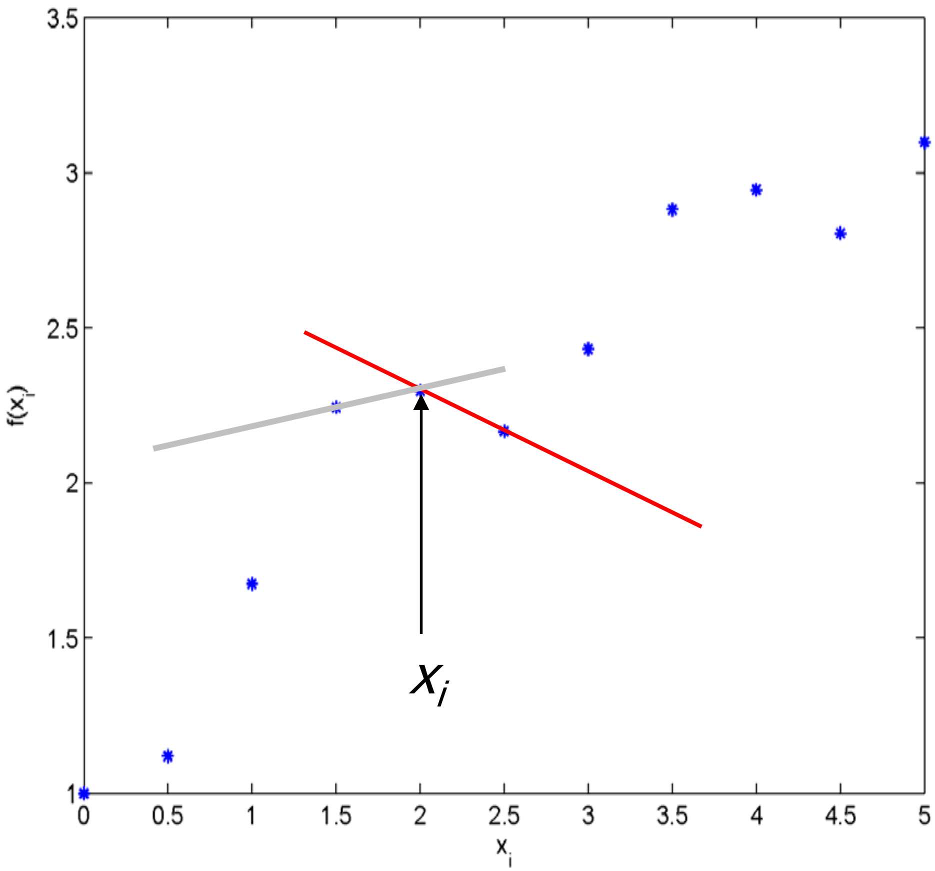


We are not given this



... but only some points

Numerical Differentiation at Pts



Calculus: Use two points (the current point and one neighbor) to estimate the derivative at the current (i^{th}) point.

We could choose:

a. x_i and x_{i+1}, (right derivative, or forward)

b. x_{i-1} and x_i (left derivative, or backward)

These choices lead to the two point ***forward*** and ***backward*** *difference approximations*, respectively.

Calculus

$$f(x_i) = \frac{f(x_i + h) - f(x_i)}{h}$$

Forward Difference

Let:

x_i – current point

x_{i+1} - next point

h = delta = $x_{i+1} - x_i$

so:

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}$$

Backward Difference

Let:

x_i – current point

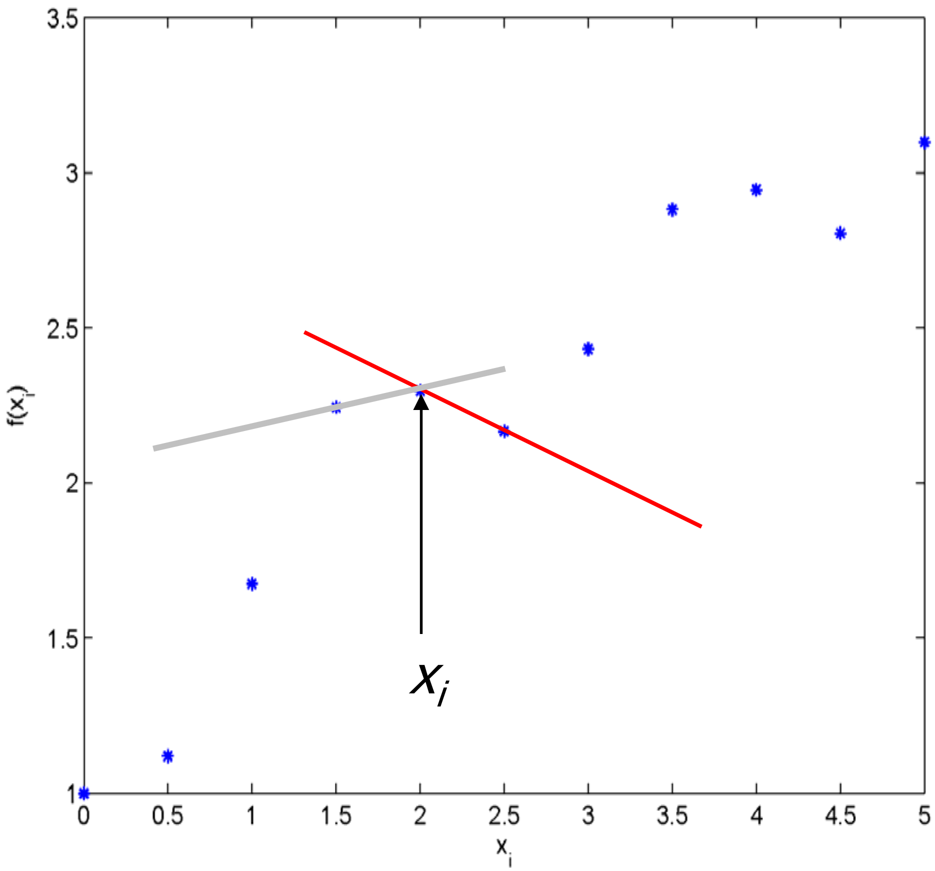
x_{i-1} - previous point

h = delta = $x_i - x_{i-1}$

so:

$$f'(x_i) = \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$

Numerical Differentiation at Pts



In mathematical notation we have:

- Forward Difference

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}$$

- Backward Difference

$$f'(x_i) = \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$

Are any of these “better” ?

Which one is Better

- Depends on *what we mean by better!*
  - depends on the application
  - In principle they coincide when h=0

- Two common criteria:
  i. The *size of the* **(truncation)** ***error*** as a function of the increment *h*

  ii. The ***causality*** of the formula (required in most DSP applications)

Truncation Error Analysis

- To estimate the error (as h changes) we perform a *Taylor series expansion in n terms* ***with remainder*** around the point x_i:

$$f(x_i + \Delta x) = f(x_i) + \Delta x f^{(1)}(x_i) + \frac{(\Delta x)^2}{2!} f^{(2)}(x_i) + \cdots + \frac{(\Delta x)^n}{n!} f^{(n)}(x_i) \boxed{+ \frac{(\Delta x)^{[n+1]}}{(n+1)!} f^{(n+1)}(\xi),}$$

Remainder

$$x_i < \xi < x_i + \Delta x$$

(The number of terms *n* in the expansion depends on the differentiation formula analyzed)

Error: Two Point Formulas

Expand the function f around the point x_i in a Taylor series ***up to second order***

$$f(x_i \pm h) = f(x_i) \pm h f^{(1)}(x_i) + \frac{h^2}{2!} f^{(2)}(\zeta)$$

Then it follows that

FW: $$f^{(1)}(x_i) = \frac{f(x_i + h) - f(x_i)}{h} - \frac{f^{(2)}(\zeta)}{2}(h)$$

BW: $$f^{(1)}(x_i) = \frac{f(x_i) - f(x_i - h)}{h} + \frac{f^{(2)}(\zeta)}{2}(h)$$

* * * Both have $\mathcal{O}(h)$ error * * *

Functions as "Signals"

Mathematical notation:

$$\{(x_k, y_k)\}_{k=0}^{n}, \quad y_k = f(x_k)$$

Signals notation: $h_k = x_k - x_{k-1}$

$$\{(x_k, y_k)\}_{k=0}^{n}, \quad f(kh_k) = y_k$$

(h_k is the "sampling period")

Special case: Uniform sampling $h = x_k - x_{k-1}$

$$\begin{aligned} f(kh_k) = y_k &\Rightarrow f(kh) = y_k \\ &\Rightarrow f[k] = y[k] \end{aligned}$$

It is sufficient to specify k, the index of the k^{th} sample

Causality

- Assume that the function to be differentiated is a ***time signal*** (*i.e.*, the independent variable, is time)

- Assume that the signal is ***sampled uniformly*** at increments of *h* units apart

$$f(x) \xrightarrow{h} f(k\,h), \quad k \text{ indep. var}$$

- Using the notation introduced, we can write

$$f(x_k) = f(k\,h) = f[\,k\,]$$

The notation *f[i]* (with square brackets) is common in DSP (where the "sampling period" *h* is implicit)

- Consider the problem of estimating the *derivative* of a signal *at the* ***current time k***

Forward Difference Approximation

Using a signal interpretation we have:

$$f'(x_k) = \frac{f(x_{k+1}) - f(x_k)}{x_{k+1} - x_k} = \frac{f[k+1] - f[k]}{h}$$

To compute derivative at "time" kh ,we need

- Value of signal at **future "time", *(k+1)k***
- Value of signal at **current "time", *kh***

Computation of derivative at current time ***requires value of the signal in the future !***

(since we cannot use future values, the forward difference is a **non-causal** algorithm for differentiation)

Backward Difference Approximation

Using a signal interpretation we have:

$$f'(x_k) = \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}} = \frac{f[k] - f[k-1]}{h}$$

To compute derivative at "time" kh ,we need

- Value of signal at **current "time", *kh***
- Value of signal at **past "time", *(k-1)h***

Computation of derivative at current time **requires present and past values** only

(since we only use present and past values, the backward difference is a **causal** algorithm for differentiation)

Digital Differentiators

- From a ***signal processing*** perspective the differentiation operations can be implemented as a ***digital filter*** and is given by the ***difference equation***

$$D_{F_2}[n] = \frac{1}{h}\, f[n+1] - \frac{1}{h}\, f[n]$$

Not causal ☹

$$D_{B_2}[n] = \frac{1}{h}\, f[n] - \frac{1}{h}\, f[n-1]$$

Causal ☺

The ***output*** of this filter is the derivative of the input and is obtained as a ***weighted sum*** of the input samples. These filters are called **Finite Impulse Response (FIR) filters**

Summary

- *Differentiation* formulas based only on the value of the function at given points are called ***finite differences***
- Generally, a **small *h*** gives ***better*** approximation to the derivative
  - *i.e.*, lower truncation error
- The ***simplest formulas*** for the first derivatives are the *two point* ***forward*** ***and*** ***backward*** ***differences*** ($O(h)$)
- In ***DSP*** applications only the ***backward difference*** can be implemented due to *causality*.

Recall: Two Point Differences

(Two point) Forward Difference

$$f'(x_k) = \frac{f(x_{k+1}) - f(x_k)}{x_{k+1} - x_k} = \frac{f[k+1] - f[k]}{h}$$

(Two point) Backward Difference

$$f'(x_k) = \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}} = \frac{f[k] - f[k-1]}{h}$$

The forward and backward difference formulas both have error that is "***O(h)***"

- If ***h*** is fixed, what can we do to minimize the error?

Reducing The Truncation Error

- If we ***use more information*** about the function, we should be able to obtain *better results* (reduce the error)

- What information can we use ?

  The information that we have available is the **value of the function at neighboring points.**

- If instead of using only *two points if* we ***use more neighboring points*** to approximate the derivatives we may be able *to do better*

Higher Order Formulas

- Higher order (multiple point) finite difference formulas can be obtained (derived) in two ways:

  1. By ***numerical interpolation*** (e.g., Lagrange polynomials) *followed by symbolic differentiation and sampling*.

  2. By ***Taylor Series Expansions***. The advantage of this approach is that we ***also*** get the **truncation error estimates**

Higher Order Formulas

Taylor Series Approach:

- To derive **3 points** formulas proceed as follows:
  - Write *one Taylor Series Expansion* of the function (with remainder of order h^3) *per neighboring point*
  - Combine these expressions into one *eliminating terms in* h^2
  - Rewrite to obtain desired formula.

- This approach can be used to derive differentiation formulas of arbitrary number of points.

Reminder: General Taylor Expansion

- To estimate the error we perform a *Taylor series expansion in n terms* **with remainder** around the point x_i:

$$f(x_i + \Delta x) = f(x_i) + \Delta x f^{(1)}(x_i) + \frac{(\Delta x)^2}{2!} f^{(2)}(x_i) + \cdots + \frac{(\Delta x)^n}{n!} f^{(n)}(x_i) \boxed{+ \frac{(\Delta x)^{[n+1]}}{(n+1)!} f^{(n+1)}(\xi),}$$

Remainder

$$x_i < \xi < x_i + \Delta x$$

In this case n will be 3

3 pts Forward Differences

- 3 pts Forward Difference (D_{F3})

$$f'(x_k) \approx \frac{-3f(x_k) + 4f(x_{k+1}) - f(x_{k+2})}{x_{k+2} - x_k}$$

$$D_{F_3}[k] = \frac{-3f[k] + 4f[k+1] - f[k+2]}{2h}$$

$$= \frac{1}{h}\left(-\frac{3}{2}f[k] + 2f[k+1] - \frac{1}{2}f[k+2]\right)$$

- The error is $O(h^2)$

The sum of the “weights” must add up to zero: $-\frac{3}{2} + 2 - \frac{1}{2} = 0$

3 pts Backward Differences

- 3 pts Backward Difference (D_{B3})

$$f'(x_k) \approx \frac{3f(x_k) - 4f(x_{k-1}) + f(x_{k-2})}{x_k - x_{k-2}}$$

$$D_{B_3}[k] = \frac{3f[k] - 4f[k-1] + f[k-2]}{2h}$$

$$= \frac{1}{h}\left(\frac{3}{2}f[k] - 2f[k-1] + \frac{1}{2}f[k-2]\right)$$

- The error is $O(h^2)$

The sum of the "weights" must add up to zero (true for all finite difference formulas)

3 pts Central Differences

- 3 pts Central Difference (D_{C3})

$$f'(x_k) \approx \frac{f(x_{k+1}) - f(x_{k-1})}{x_{k+1} - x_{k-1}}$$

$$D_{C_3}[k] = \frac{f[k+1] - f[k-1])}{2h}$$

$$= \frac{1}{h}\left(\frac{1}{2} f[k+1] - \frac{1}{2} f[k-1]\right)$$

- The error is $O(h^2)$

Note: This is a **3 points** difference approximation that only **requires 2 function evaluations!**

Error Approximations

High Order Finite Difference Approximations

- *Three-point finite difference* formulas are of *order 2.*
  - The error is proportional to h^2

- ***N-point finite difference*** formulas are of ***order N-1***, $O(h^{N-1})$
  - *The error is proportional to* h^{N-1}

- ***Error can be controlled without changing the sampling period "h".***

Central Difference Advantage

Engineering Significance:

If there is **no causality** constraint, the ***central difference*** *is the best* because:

- For a given number of data points used in the approximation formula it leads to *smaller errors.*
- It only requires *two data points* for evaluation.
- This formulas is widely used in image processing

Digital Differentiators

- The 2nd order backward difference is described by the following difference equation

$$D_{B_3}[n] = \frac{3}{2h} f[n] - \frac{2}{h} f[n-1] + \frac{1}{2h} f[n-2]$$

- Compare to the 1st order backward difference

$$D_{B_2}[n] = \frac{1}{h} f[n] - \frac{1}{h} f[n-1]$$

- **Both can be implemented as FIR filters**

Choice of Approximation

- When ***causality*** is an issue (digital signal processing) ***backward approximations*** are our only choice.

- When ***causality is not required*** ***central differences*** are ***more accurate***

Caution:

Differentiation *exacerbates errors* (e.g., ***amplifies noise***). To use it effectively you need to ***first filter the noise***

Applied Programming

Numerical Integration

and

Digital Integrators

Why Numerical Integration ?

Analytical integration is:

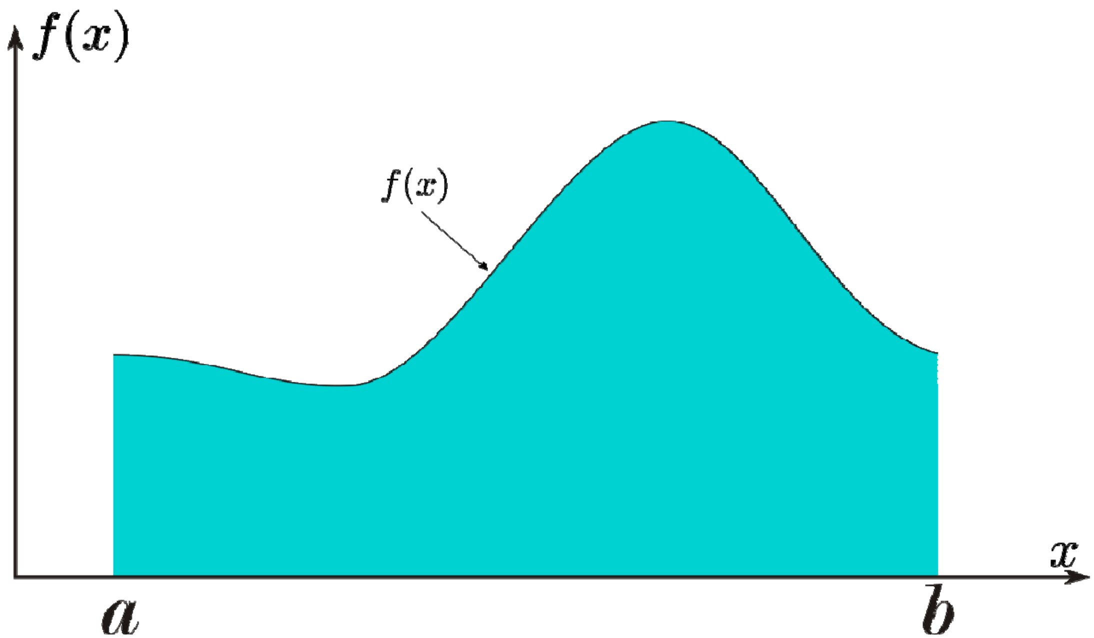
- Difficult:
  - requires symbolic computations.
- Not possible:
  - closed form solution is not known.
- Not practical:
  - in many situations the function is not know.
- Applications:
  - Digital Integrators are often used in control systems (e.g. PI control)

Integral: Geometric Interpretation

Area under a "curve" between

$a \le x \le b$

$$\boxed{\int_a^b f(x)dx}$$



From Calculus:

Limit of Riemann Sum as $\Delta x \to 0$

$$\int_a^b f(x)dx = \lim_{\Delta x \to 0} \sum_{k=0}^{n-1} f(x_k)\Delta x$$

Use this idea to develop integration algorithms

Numerical Integration

Objective

Find the integral of a function in a given interval *using only values of the function at a finite number of point*, *e.g.*, $y_i = f(x_i), i=1..n$

- Sometimes we are given a closed form *expression for the function* to integrate and are free to pick the points
- Normally we are only given pairs $(x_i,f(x_i))$ that describe the *values of the function at given points*

Numerical Integration

Engineering applications of numerical integration:

I. Estimation of the integral of a function available only at a discrete set of points, e.g., $y_i = f(x_i), i=1..n$
Example: Compute velocity from acceleration measurements.

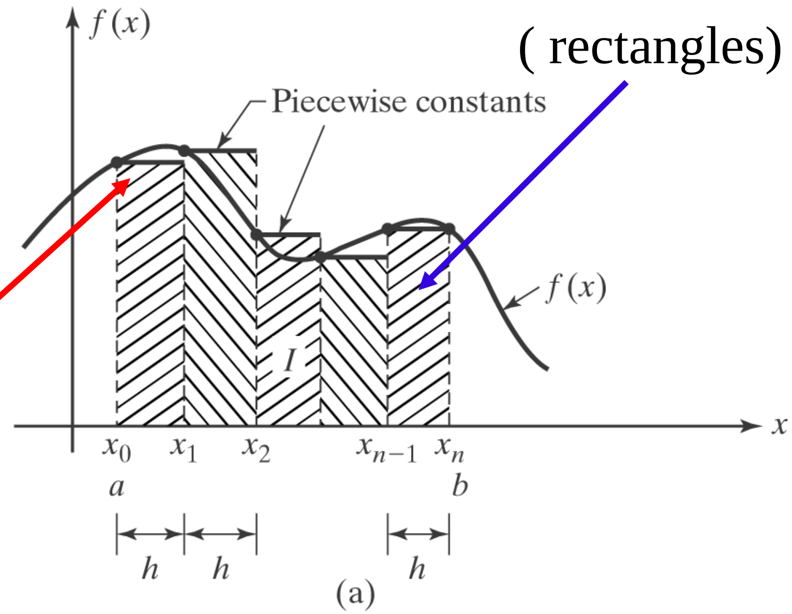
II. Simulation of dynamic systems described by differential equations
Example: Flight simulators, general numerical simulators control and signal processing systems, etc. (e.g., Simulink)

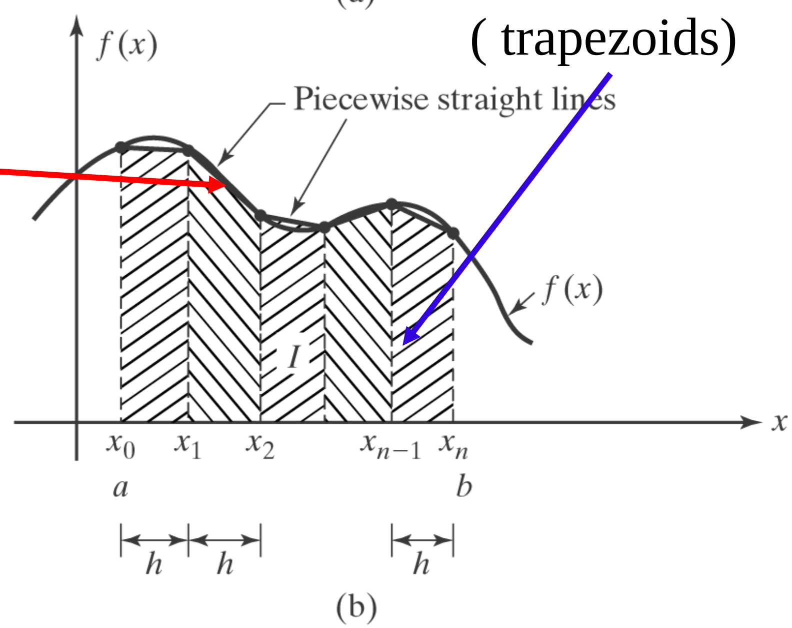
*III. **Control Systems** often use integrators in tracing systems to reduce errors.*

Numerical Integration

General approach:

1. Discretize independent variable ($\Delta x = x_{i+1} - x_i = h$)
2. ***Interpolate*** function between points x_i and x_{i+1} with a polynomial:
   1. Constant (0^{th} order)
   2. Line (1^{st} order)
   3. Parabola, …
3. ***Integrate interpolated function***, e.g., find area under polynomial between x_i and x_{i+1} and add all.





Figures used with permission from support materials for *Applied Numerical Methods for Engineers and Scientists*, S. S. Rao, Prentice Hall, 2002.

Signal Processing Perspective

- Discretization is uniform (equidistant pts) and is caused by sampling a input signal

$$f(x) \xrightarrow{t:=n\,h} f(kh) = f[k]$$

- ***Integral*** is the ***output of a digital filter*** (digital integrator)

- The integration algorithm defines the digital filter

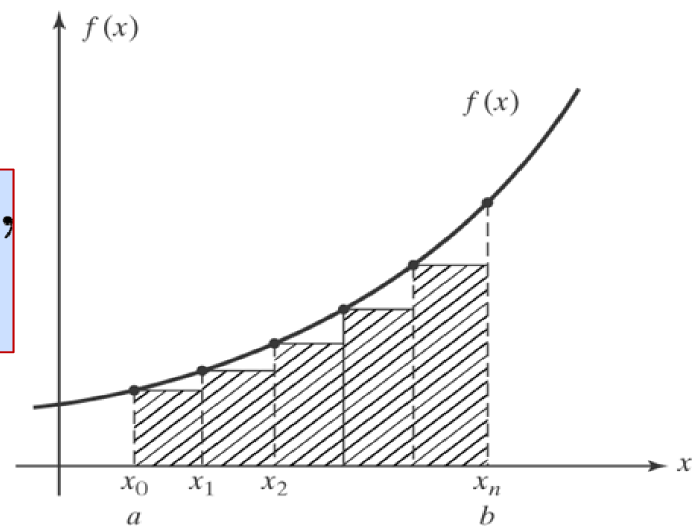
Forward Rectangular Integrator

1. Discretization: Uniform (equidistant pts)
2. Interpolation: Constant (0^{th} order poly)

- Derive algorithm

Integrator Algorithm

$$I_F[k] = I_F[k-1] + hf[k-1],$$
$$I_F[-1] = 0$$



The ***output*** of this filter is the **integral** of the input. Note that the ***output depends on past outputs*** (it is recursive). These are called **Infinite Impulse Response (IIR) filters**

Figure used with permission from support materials for *Applied Numerical Methods for Engineers and Scientists*, S. S. Rao, Prentice Hall, 2002.

Solving Difference Equations

- Example 1: Integrator 1 (F. Rect.)

$$I_F[k] = I_F[k-1] + hy[k-1], \quad I_F[-1] = 0$$

- *By direct evaluation:*

$$I_F[0] = I_F[-1] + hy[-1] = hy[-1]$$

$$I_F[1] = I_F[0] + hy[0] = h(y[-1] + y[0])$$

$$I_F[2] = I_F[1] + hy[1] = h(y[-1] + y[0] + y[1])$$

$$\vdots$$

$$I_F[n] = h\sum_{k=1}^{n} y[k-1]$$

A digital integrator is just an "accumulator"

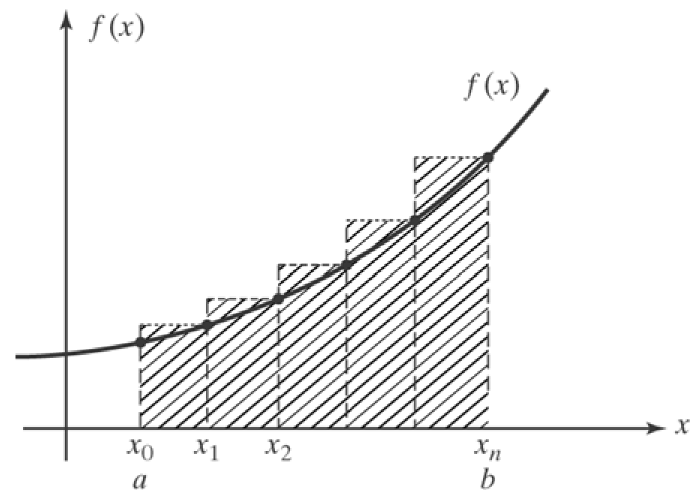
Backward Rectangular Integrator

1. Discretization: Uniform (equidistant pts)
2. Interpolation: Constant (0^{th} order poly)

- Derive algorithm

Integrator Algorithm

$$I_B[k] = I_B[k-1] + hf[k],$$
$$I_B[-1] = 0$$



Note that both integration algorithms (forward and backward rectangular) are causal.

Figure used with permission from support materials for *Applied Numerical Methods for Engineers and Scientists*, S. S. Rao, Prentice Hall, 2002.

Trapezoidal Rule

1. Discretization: Uniform (equidistant pts)
2. Interpolation: Linear (1st order poly)

- ***Integrator Algorithm:***

$$I_T[k] = I_T[k-1] + \overbrace{\frac{h}{2}(f[k-1] + f[k])}^{I_k}$$

$$I_T[-1] = 0$$

As before h is constant and $I_T[k]$ is the approx. integral up to the k^{th} segment

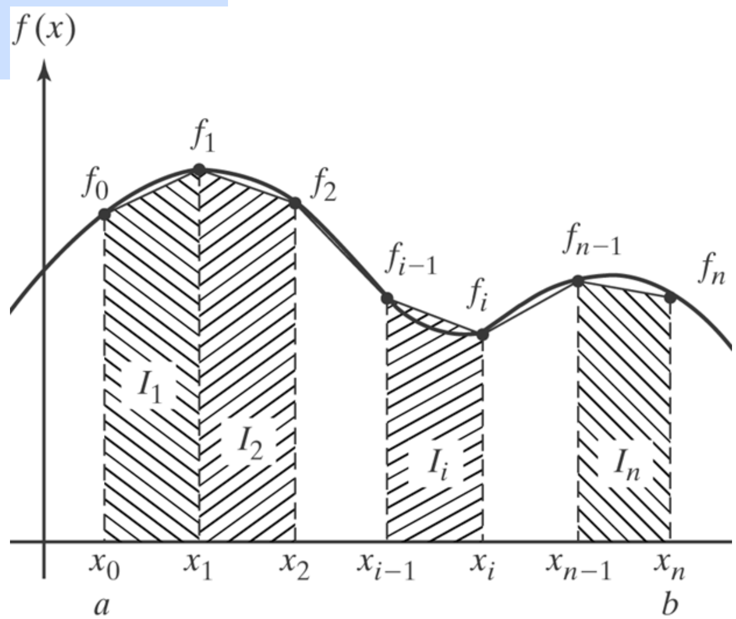


Figure used with permission from support materials for *Applied Numerical Methods for Engineers and Scientists*, S. S. Rao, Prentice Hall, 2002.

Numerical Integration and Filters

- The recursive ***integration*** formulas can be implemented in ***digital (IIR) filters***

  (IIR = Infinite Impulse Response)

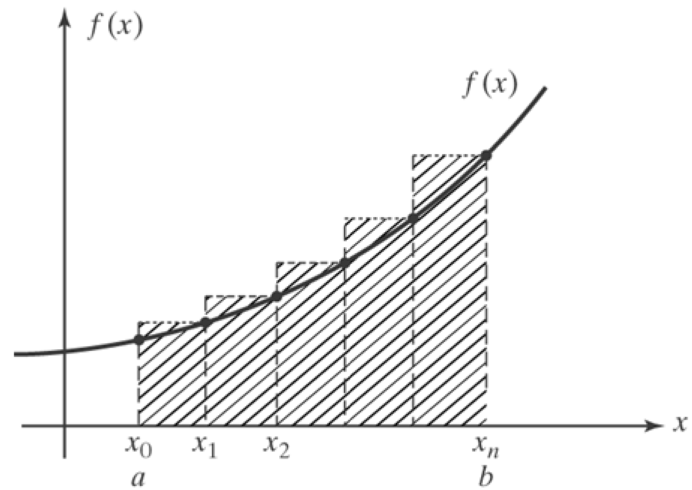
$$I_F[k] = I_F[k-1] + hy[k-1], \quad I_F[-1] = 0$$
$$I_B[k] = I_B[k-1] + hy[k], \quad I_B[-1] = 0$$
$$I_T[k] = I_T[k-1] + \frac{h}{2}(y[k-1] + y[k]), I_T[-1] = 0$$

- To "start" the filters we need to set their initial conditions (value of *I[k]* at *k=-1*)

- The output of the filter at "time" k is the integral of the function up to that time

Filters and Numerical Integration

What is the area under ***f(x)*** from x_o to x_n



- Given an algorithm (described in recursive form for filtering) the integral of the function over the interval of integration requires the **"solution of the difference equation"**
- The simplest way to solve these equations is by ***direct evaluation***

Solving Difference Equations

- Example 1: Integrator 1 (Forward Rect.)

$$I_F[k] = I_F[k-1] + hf[k-1], \quad I_F[-1] = 0$$

- *By direct evaluation:*

$$I_F[0] = I_F[-1] + hf[-1] = hf[-1]$$

$$I_F[1] = I_F[0] + hf[0] = h(f[-1] + f[0])$$

$$I_F[2] = I_F[1] + hf[1] = h(f[-1] + f[0] + f[1])$$

$$\vdots$$

$$I_F[n] = h\sum_{k=1}^{n} f[k-1]$$

A digital integrator is just an "accumulator"

Solving Difference Equations

- Example 2: Integrator 2 (Backward Rect)

$$I_B[k] = I_B[k-1] + hf[k], \quad I_B[-1] = 0$$

- *By direct evaluation:*

$$I_B[0] = I_B[-1] + hf[0] = hf[0]$$

$$I_B[1] = I_B[0] + hf[1] = h(f[0] + f[1])$$

$$I_B[2] = I_B[1] + hf[2] = h(f[0] + f[1] + f[2])$$

$$\vdots$$

$$I[n] = h\sum_{k=1}^{n} f[k]$$

Solving Difference Equations

- Example 3:Trapezoidal rule

$$I[k] = I[k-1] + \frac{h}{2}(f[k-1] + f[k]), I[-1] = 0$$

- ***<u>By direct evaluation:</u>***

$$I[0] = I[-1] + \frac{h}{2}(f[-1] + f[0]) = \frac{h}{2}(f[-1] + f[0])$$

$$I[1] = I[0] + \frac{h}{2}(f[0] + f[1]) = \frac{h}{2}(f[-1] + 2f[0] + f[1])$$

$$\vdots$$

$$I[n] = h\left(\frac{f[-1] + f[n]}{2} + \sum_{k=0}^{n-1} f[k]\right)$$

Other Integrators: Simpson (1/3) Rule

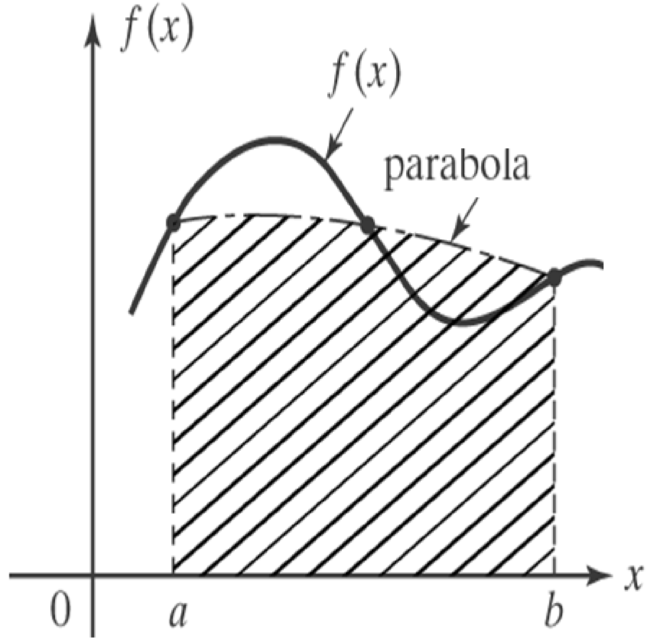
1. Discretization: Uniform (equidistant pts)
2. Interpolation: Quadratic (2^{st} order poly)

- ***Recursive formula***

$$I_S[2k] = I_S[k] + \frac{2h}{6}\left(y[2k-2] + 4y[2k-1] + y[2k]\right)$$

$$I_S[0] = 0$$

- Numerical integral up to $2k^{th}$ segment is $I_S[2k]$



Simpson (1/3) Rule

$$I_S[2k] = I_S[k] + \frac{2h}{6}\left(y[2k-2] + 4y[2k-1] + y[2k]\right)$$

$$I_S[0] = 0$$

- Simpson's (1/3) rule is often used in the derivation of a number of numerical Ordinary Differential Equation solvers

- This integrator is ***not used in DSP*** *applications* because it is *not* described by a *standard difference equation*

Choice of Integration Algorithm

If the purpose of numerical integration is:

- To design a ***filter or controller***, then the *rectangular or trapezoidal* rules are often sufficient.

- To integrate a function accurately, then ***more advanced algorithms*** such as *Romberg Integration* or *quadrature methods* are necessary (beyond scope of this course) .

This course will focus on integration algorithms for DSP and control applications.

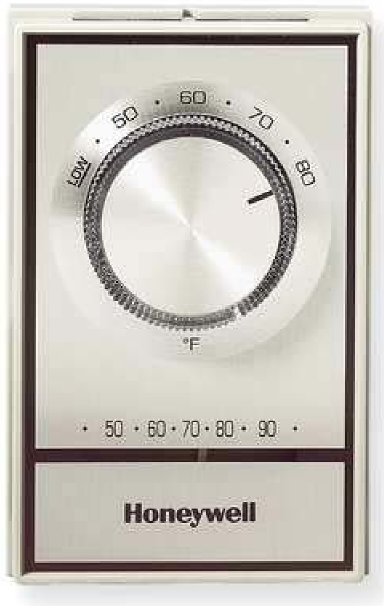
Automated Controls

- Engineers build systems that “control themselves”
  - Home heating systems
  - Car cruse controls
  - XYZ cutting systems
  - Etc

- All these systems share the same basic requirement:
  - Have a “goal”
  - Measure the “actual”
  - Generate a “correction”

Simplest Control

- Bang-bang
  - Either "Full ON"
  - Or "Full OFF"
  - Home furnace is a good example.



www.zoro.com

- Easy to build
  - Not "smooth"

PID Control

- Much “smoother” than bang-bang
  - Many small robots are driven by independent DC motors (sometimes stepper motors).

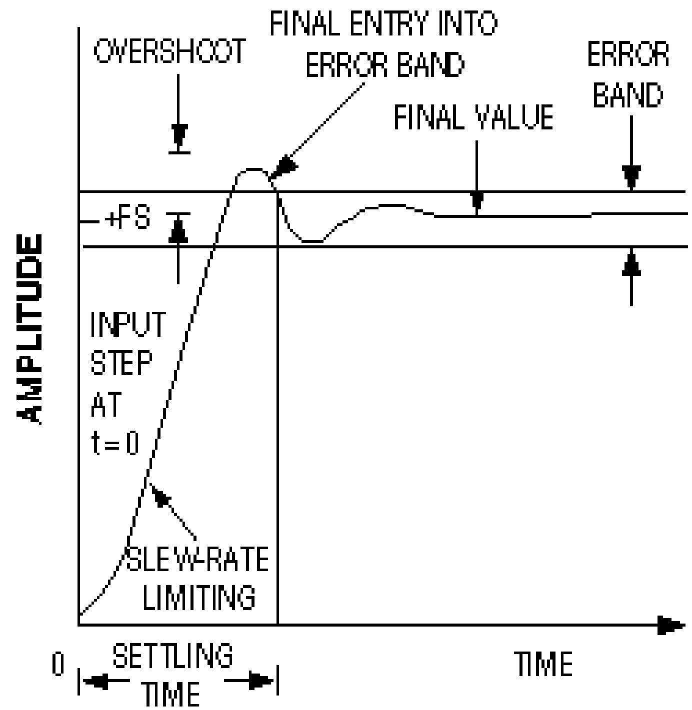
- To steer these robots the velocity of these motors must be precisely coordinated and controlled.

- This is commonly achieved using a PID (Proportional-Integral-Derivative) controller.

PID Control

- Proportional Integral Derivative (PID) Control is a popular method of controlling systems using closed-loop feedback.

- We would like to get our system to our destination value as soon as possible and then minimize oscillations.



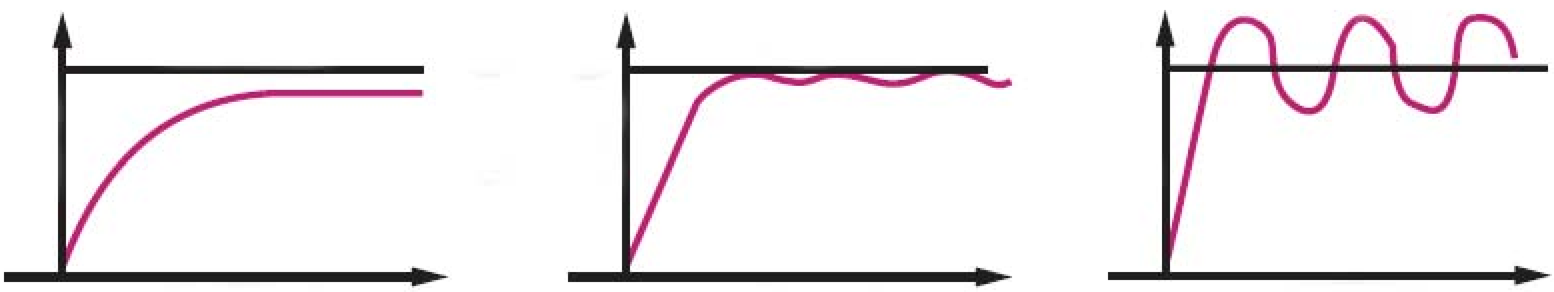
http://www.societyofrobots.com/programming_PID.shtml

- Lets take a look at Proportional, Integral, and Derivative controllers independently, then we'll put them all together.

Proportional Control

- When using proportional control alone, the equilibrium state is below Vdes (for all values of Kp).
  - This is because as Vdest approaches Vact, the difference becomes zero and the drive is off.

  - Increasing gain will decrease error but then cause oscillations



Too Low Kp (too slow response)

Kp about right

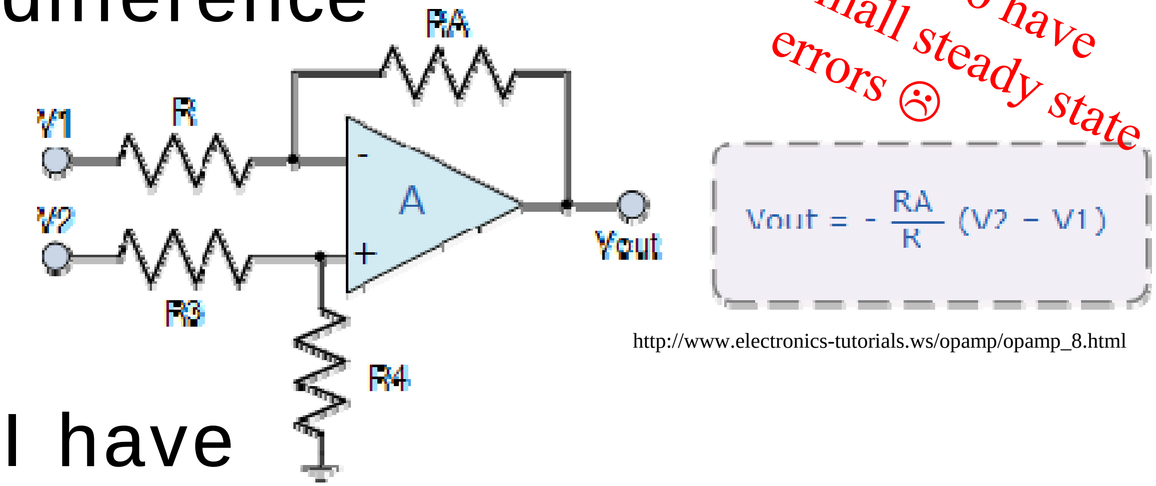
Kp too high (too much oscillation)

Becker, Hsu, Ptucha, RIT 2015

Analog Proportional Control

- Operational Amplifier (op amp)
  - Lots of configurations
  - Setup for “difference”

Tends to have small steady state errors ☹

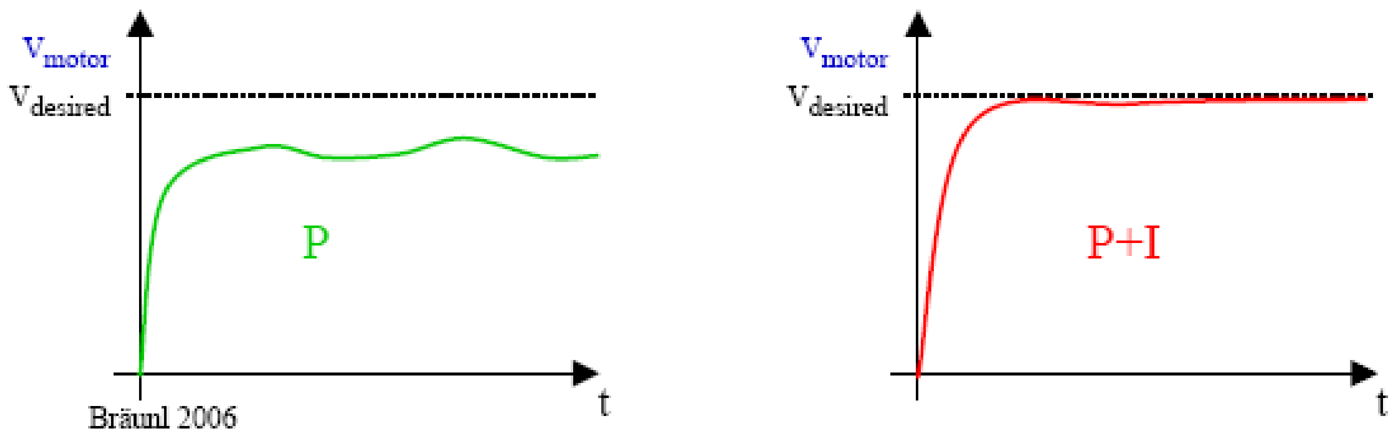


http://www.electronics-tutorials.ws/opamp/opamp_8.html

- 2 inputs
  - V1 – what I have
  - V2 – what I want
  - Vout – error (drive signal)
- In a digital world, this is subtraction

Integral Control

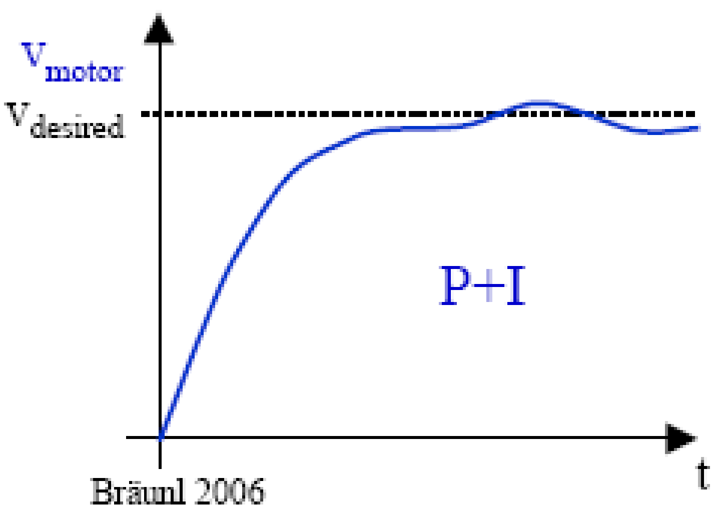
- The primary purpose of the Integral Controller is to minimize any steady-state error introduced by the Proportional Controller.
  - Problem: P-Controller may reach equilibrium without reaching the target value (steady-state error)
  - Solution: Introduce an integral mechanism to eliminate steady-state error.

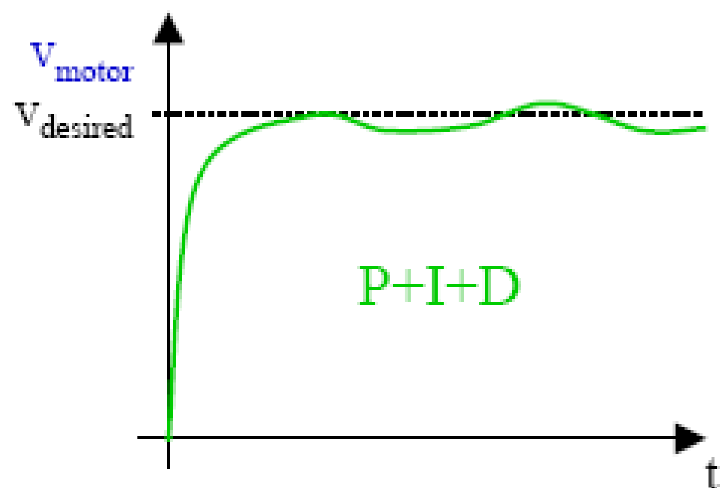


Becker, Hsu, Ptucha, RIT 2015

Derivative Control

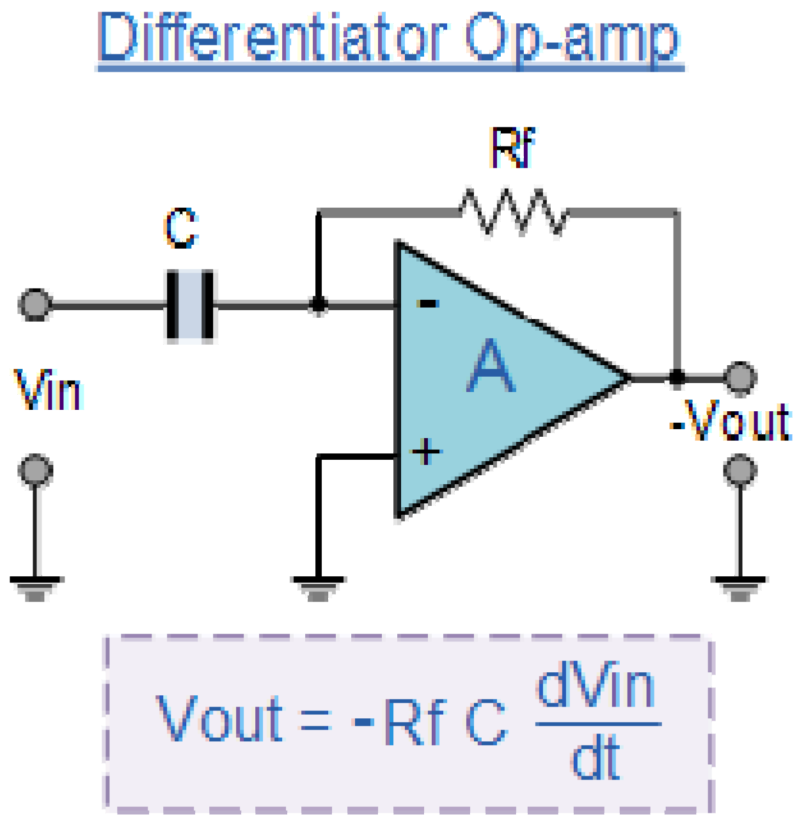
- Adding a Derivative Controller will allow us to bring our process into control faster
  - P-Controller responds slowly to change in input
  - P-Controller with high gain tends to oscillate
- Solution: Add a derivative term for response/dampening

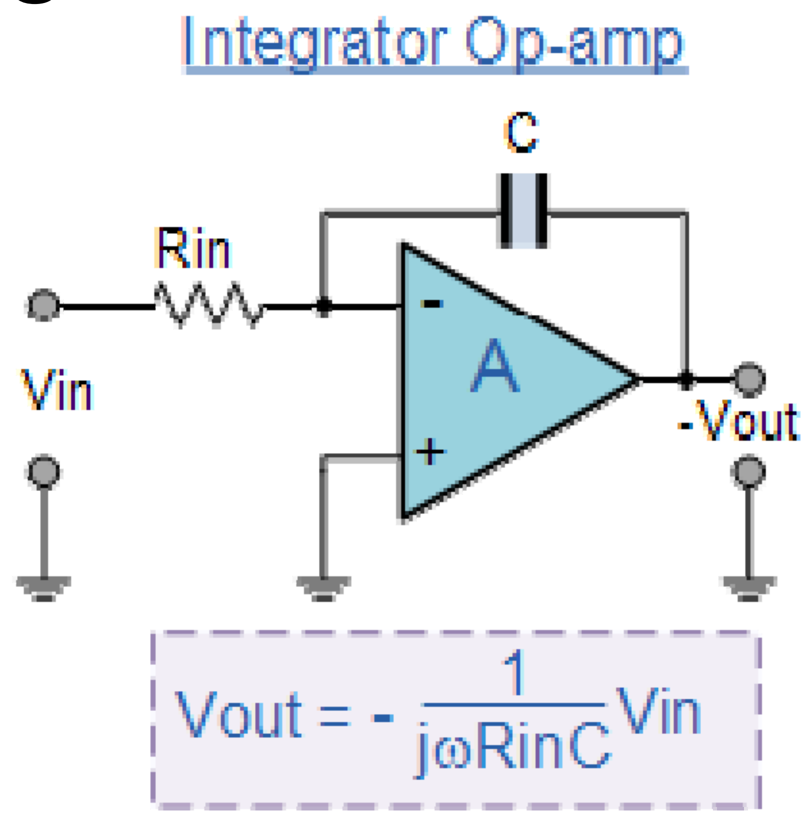




Becker, Hsu, Ptucha, RIT 2015

Analog





- Yes it is possible to do “math” in a pure analog space!
  - But keeping the voltage in the rails is hard

http://www.electronics-tutorials.ws/opamp/opamp51.gif?x98918

PID Control

- A PID controller takes an error signal, and generates a control output, *c(t)* , that is a *weighted sum* of the *error, e(t),*

$$c(t) = \overset{\text{P}}{K_p e(t)} + \overset{\text{I}}{K_i \int_0^t e(\tau) d\tau} + \overset{\text{D}}{K_d \frac{de(t)}{dt}}$$

- The "weights" (*gains*) K_p, K_i and K_d must be chosen to achieve a desired control action. They are often found by "trial and error", using "tuning algorithms" or using digital control techniques (Control Systems Theory)

Digital PID from Analog PID

- How can we implement the PID control

  $$c(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$$

  in a computer (e.g., microcontroller) ?

- Need to "discretize" the above equations:
  - Perform differentiation and integration numerically
  - The increment *h* represents the *sampling period* (how often we read the sensors and update the control)

PID Control: Discretization

$$c(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$$

- First, need to select a sampling period *h*. This depends on the performance objectives, hardware used, etc. (faster is NOT better !!)

  $t = kh, k = \ldots,-1,0,1,\ldots]$

- Discretize term by term (*h* becomes implicit)

  $c[k] = P[k] + I[k] + D[k]$

PID Control: Discretization

$$c(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$$

- Proportional Term: *P[k]*

$$P(kh) = K_p e(kh)$$

$$\boxed{P[k] = K_p e[k]}$$

The proportional gain for the digital PID and analog PID are the same

PID Control: Discretization

$$c(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$$

- Integral term: *I[k]*

Use trapezoid rule

$$I(kh) \approx K_i \left(\int_0^{(k-1)h} e(\tau)d\tau + \int_{(k-1)h}^{kh} e(\tau)d\tau \right)$$

$$\approx K_i \left(I[k-1] + \frac{h}{2}\left(e[k] + e[k-1]\right) \right)$$

The integral term cannot be simplified further
(the "output" depends on previous outputs)

PID Control: Discretization

$$c(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$$

- Derivative term: *D[k]*

Use backward difference

$$D(hk) \approx \frac{K_d}{h}\left(e(hk) - e(h(k-1))\right)$$

$$\boxed{D[k] = \frac{K_d}{h}\left(e[k] - e[k-1]\right)}$$

The derivative gain for the digital PID and analog PID are different (scaled by sampling period)

Digital PID Control

- The digital PID control law can be computed term by term.

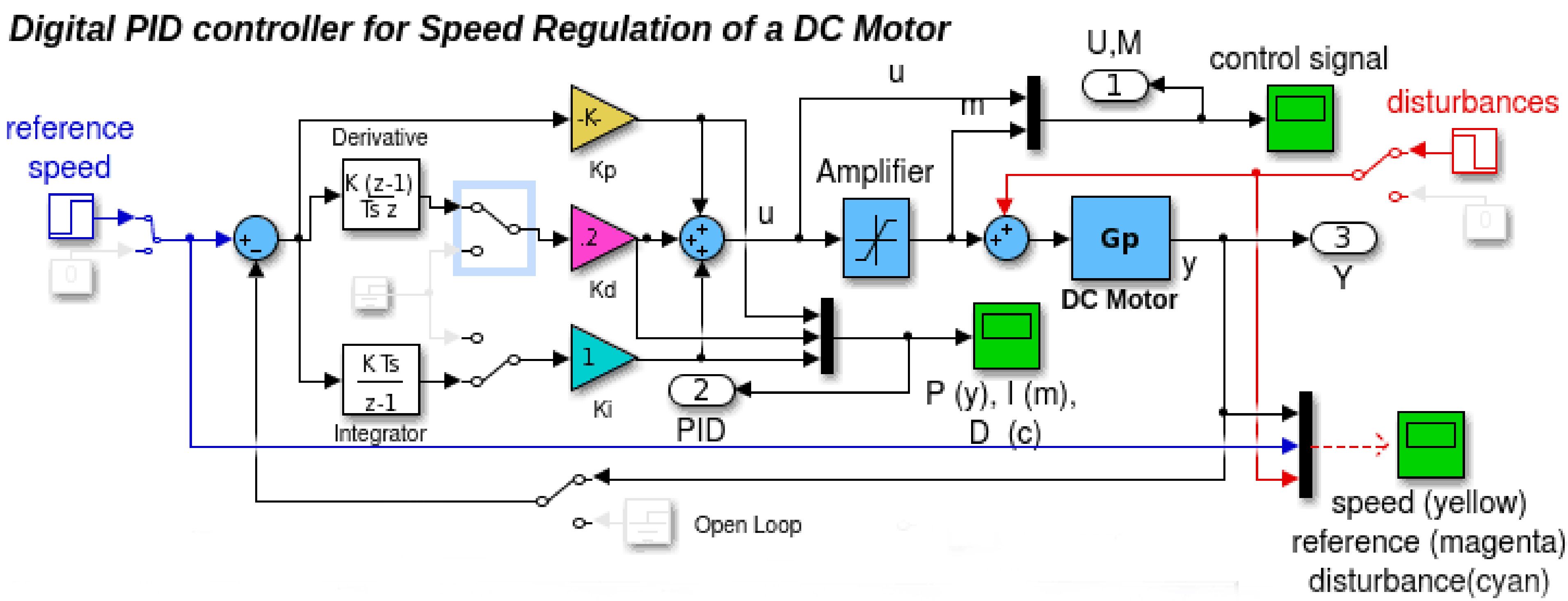
- It is possible to write the PID controller using transfer functions (more in DSP)

$$C(z) = K_p E(z) + \frac{K_i h}{2}\left(\frac{z+1}{z-1}\right) E(x) + \frac{K_d}{h}\left(\frac{z+1}{z}\right) E(z)$$

The integrator is an IIR filter and the differentiator an FIR filter

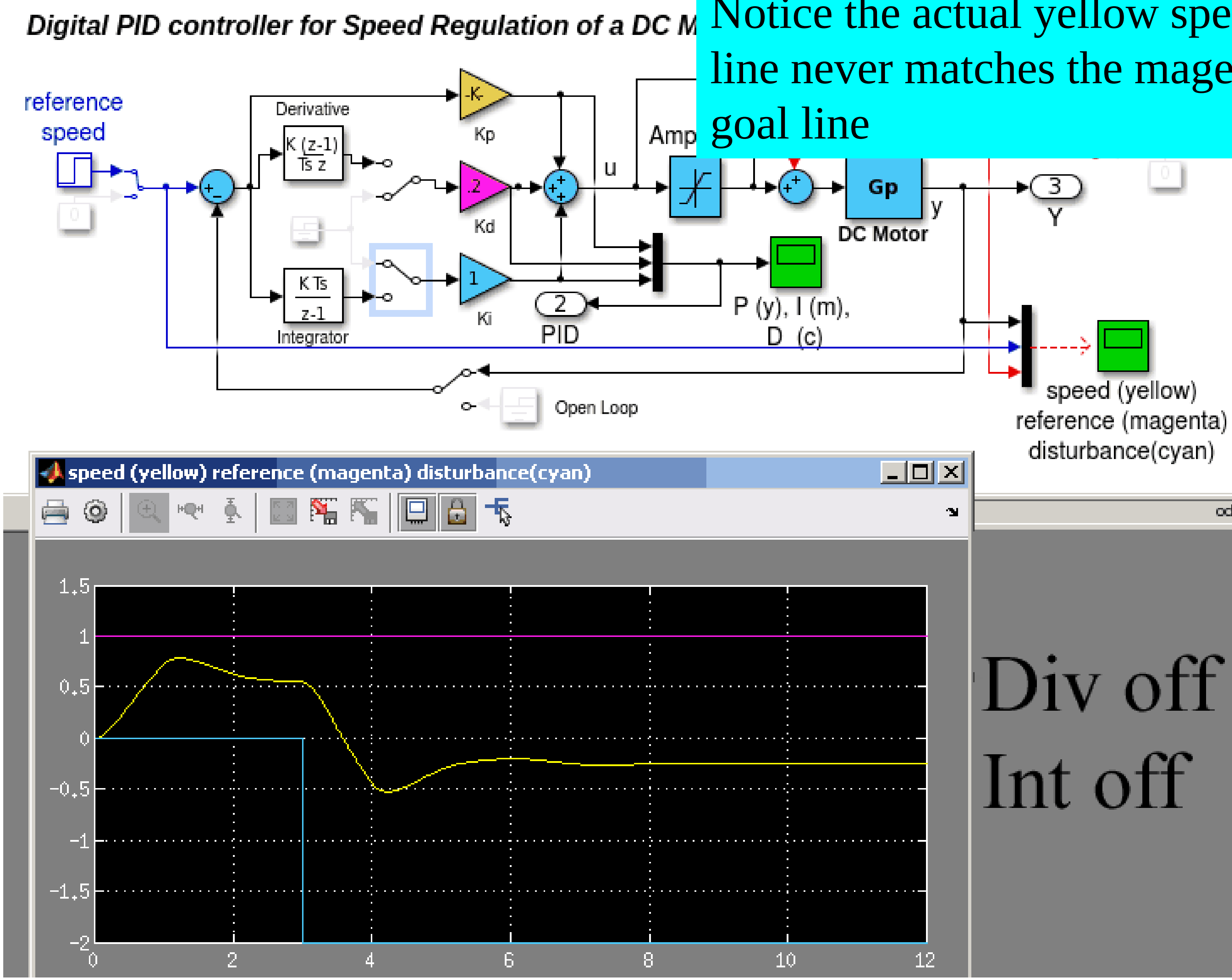
Digital PID Control - Simulation

- Demo: Speed Regulation of DC Motor



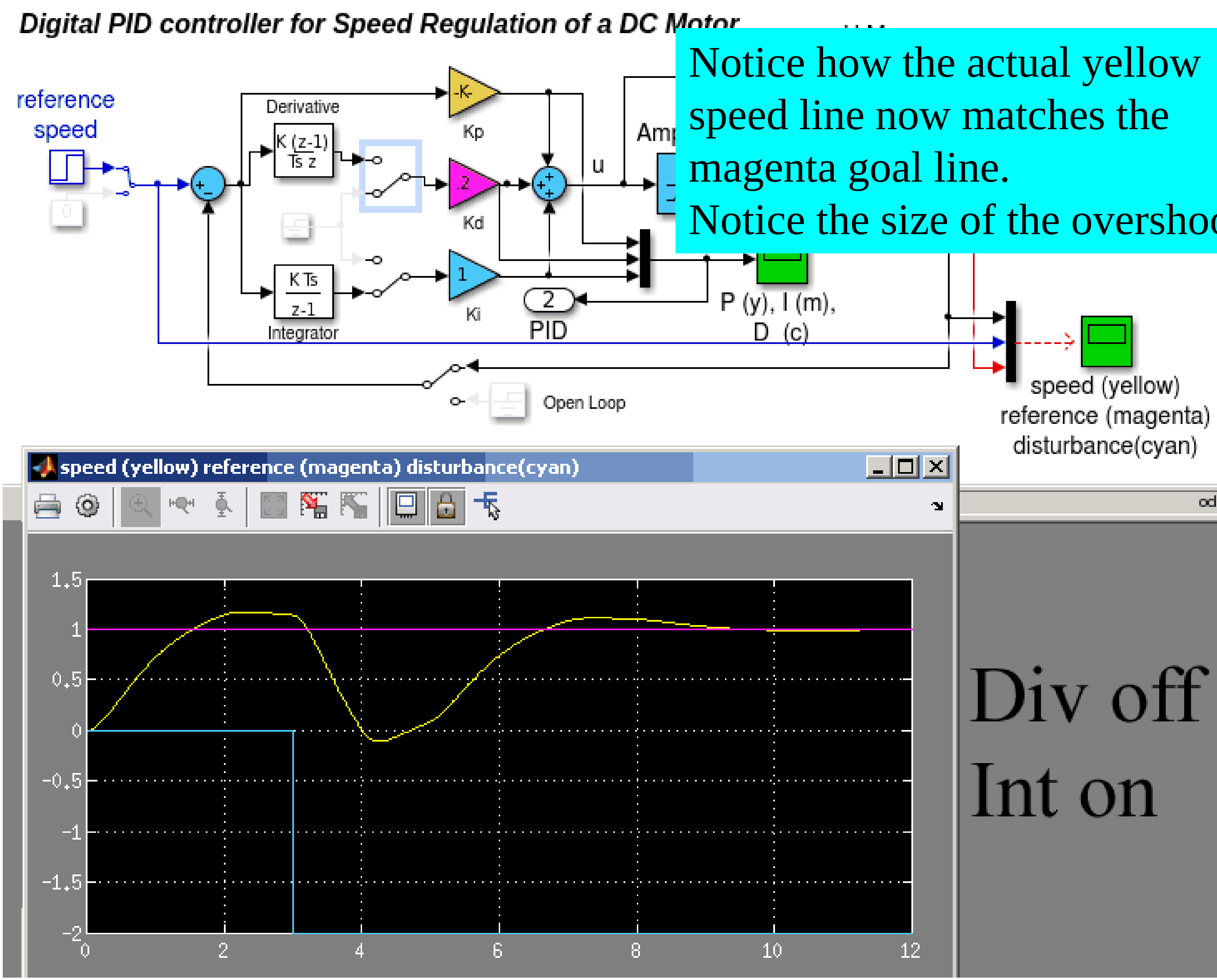
Simulink Block Diagram of a Digital PID Control System

Notice the actual yellow speed line never matches the magenta goal line



Div off

Int off

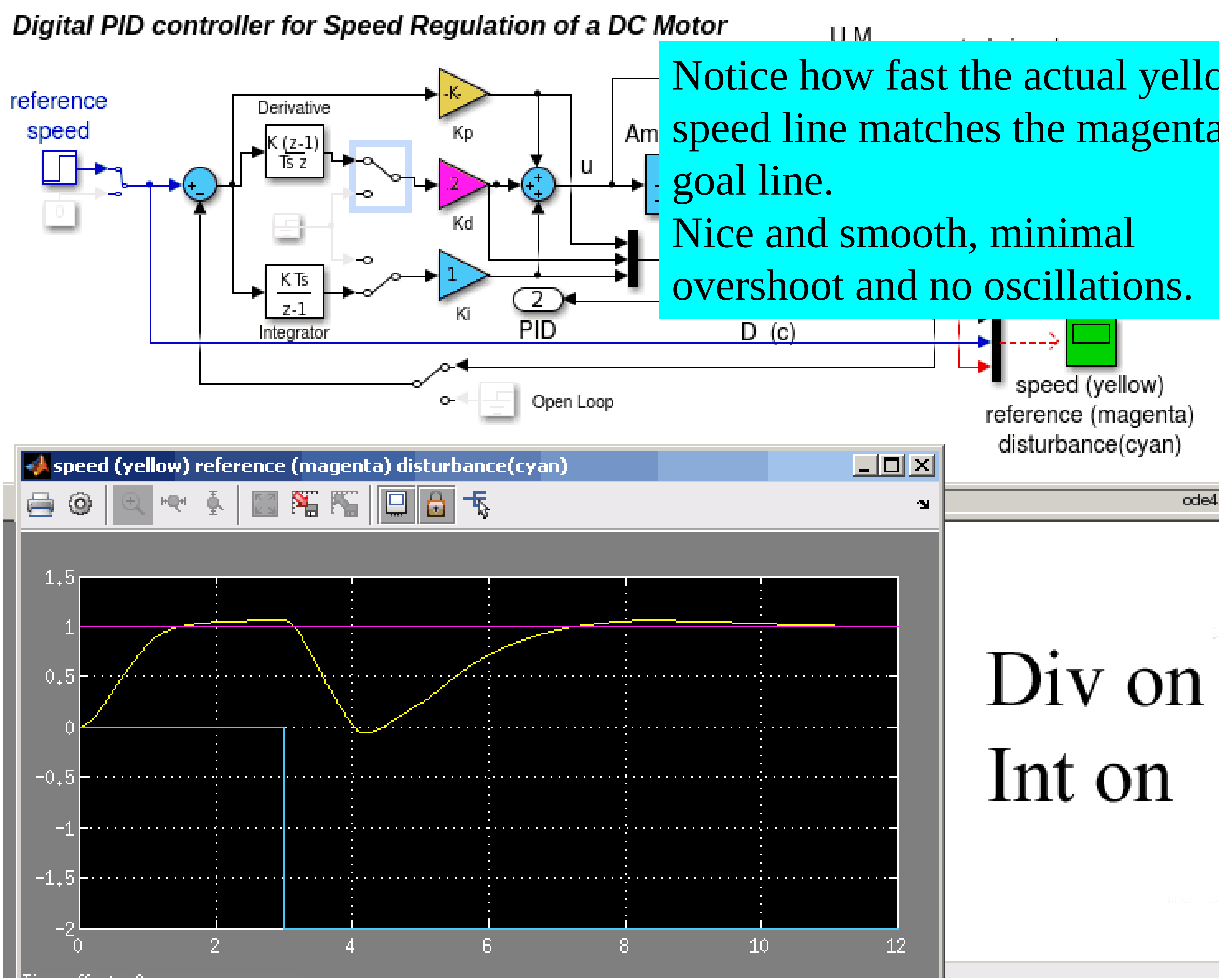


Notice how the actual yellow speed line now matches the magenta goal line.

Notice the size of the overshoot

Div off

Int on



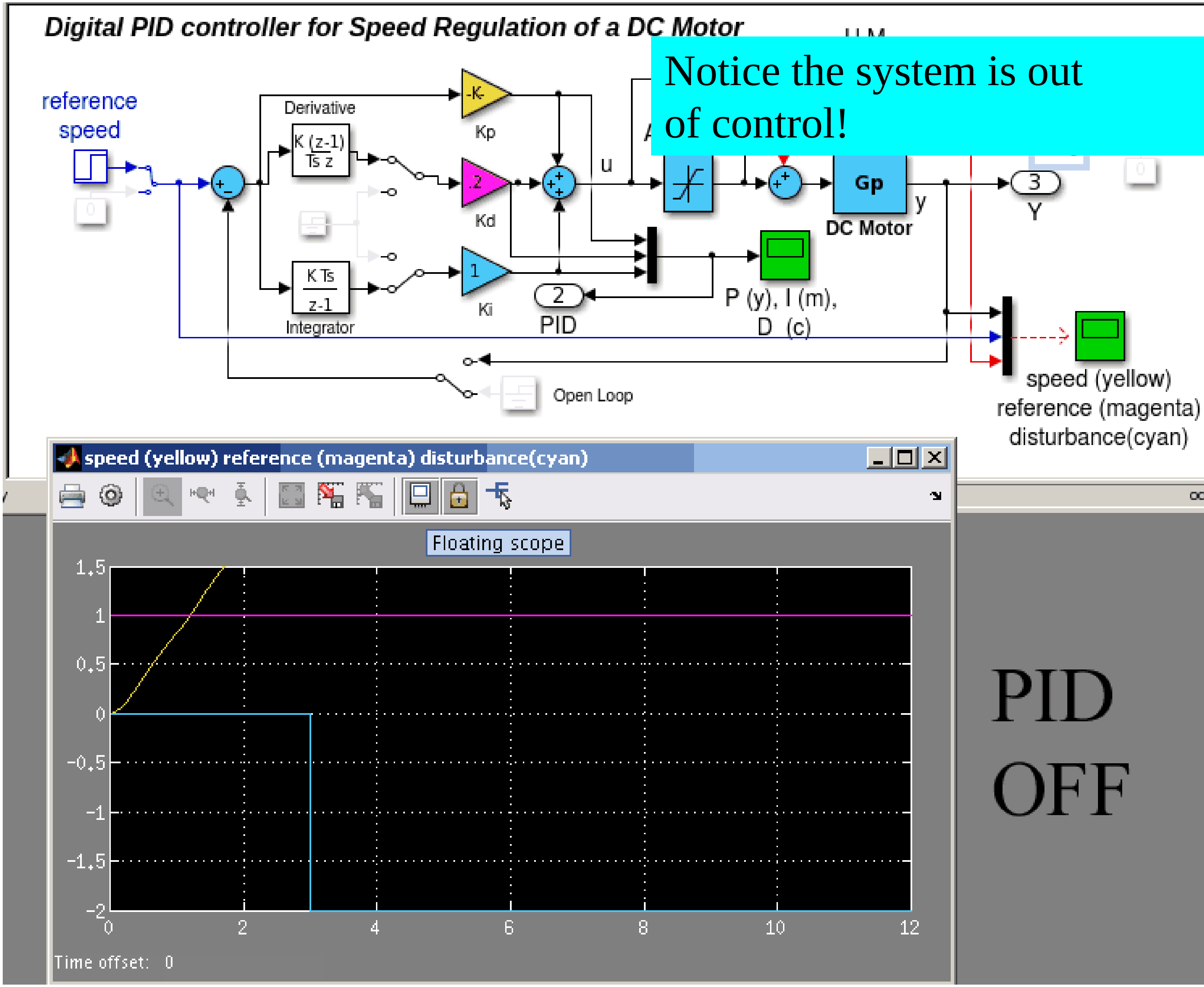
Notice how fast the actual yellow speed line matches the magenta goal line.

Nice and smooth, minimal overshoot and no oscillations.

Div on

Int on

Notice the system is out of control!



PID OFF