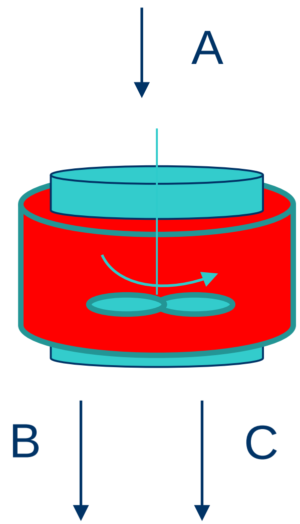
Dynamic Optimization

Prof. Cesar de Prada
ISA –UVA
prada@autom.uva.es

Outline

- Dynamic optimization problems
- Parameterization
- Sequential approach
- Simultaneous approach
- Path constraints
- Applications
- Software

A dynamic system: batch reactor



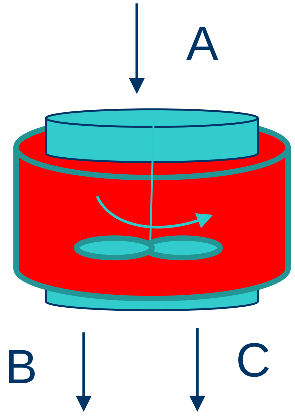
The operation of an endothermic batch reactor last for one hour. It loads an amount A, which reacts according to the parallel reactions A $\rightarrow$ B and A $\rightarrow$ C, but only the B product has commercial value. The speeds of reaction are given by:

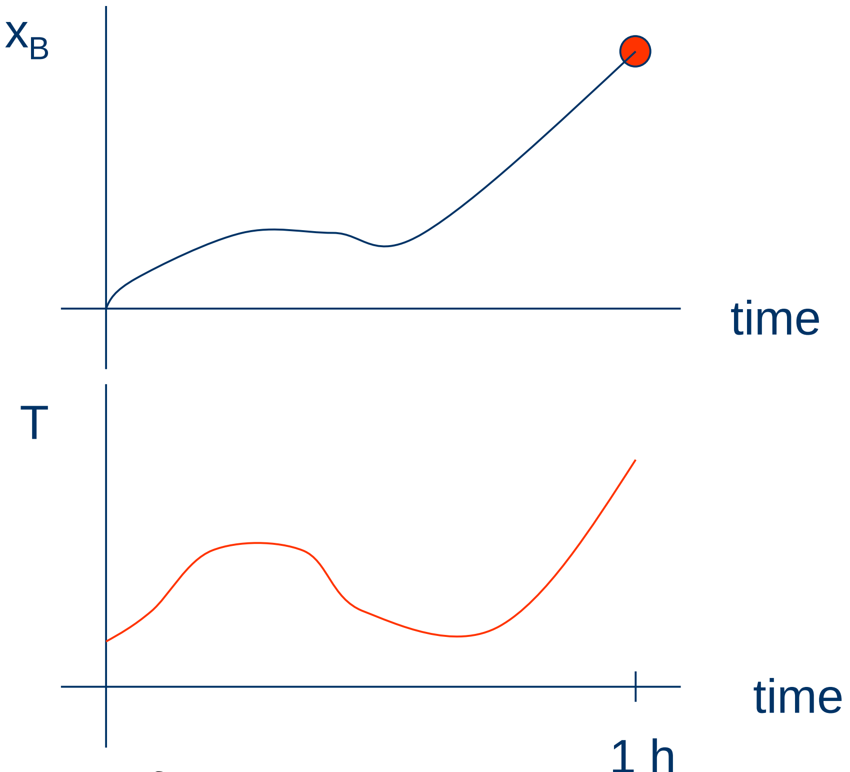
$$k_B = 10^6 \exp(10000 / RT)$$

$$k_C = 5 * 10^{11} \exp(20000 / RT)$$

Find the temperature profile that maximizes the final production of B, if the temperature must always be bellow 139 ºC

Dynamic Optimization (DO)





$$\max_{T(t)} \quad x_B(1)$$

$$\frac{dx_A}{dt} = -(k_B + k_C)x_A \qquad x_A(0) = A_0$$

$$\frac{dx_B}{dt} = k_B x_A \qquad x_B(0) = 0$$

$$T(t) \le 139$$

$$k_B = 10^6 \exp(10000 / RT)$$

$$k_C = 5*10^{11} \exp(20000 / RT)$$

Prof. Cesar de Prada ISA-UVA

Dynamic Optimization

$$\min_{u(t),x(t),x_0,t_f} \quad J(\mathbf{u}) = \int_{t_0}^{t_f} C(\mathbf{x},\mathbf{u})dt$$

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x},\mathbf{u},\mathbf{z}), \quad \mathbf{x}(t_0) = \mathbf{x}_0$$

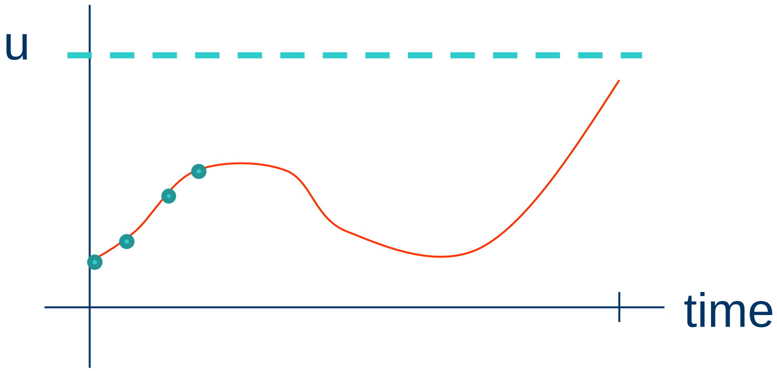
$$\mathbf{h}(\mathbf{x},\mathbf{u},\mathbf{z}) = \mathbf{0}$$

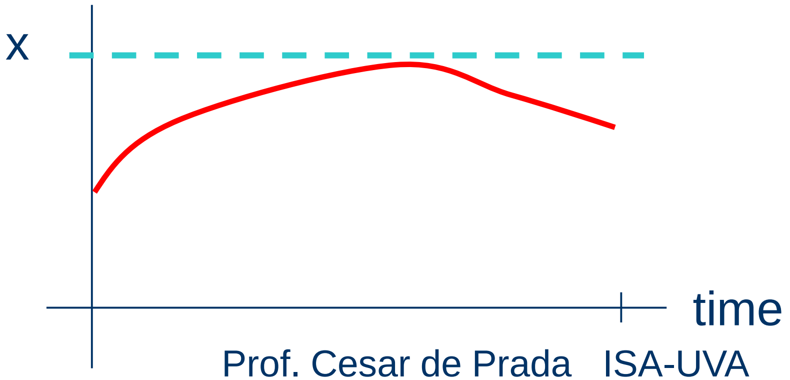
DAE

$$\mathbf{g}(\mathbf{x},\mathbf{u},\mathbf{z}) \le \mathbf{0}$$

- ✓ Many types:
  - ✓ Integral or algebraic cost
  - ✓ Initial value problems
  - ✓ TPBV problems
  - ✓ Final time problems
  - ✓ DAE or ODE
  - ✓ ….

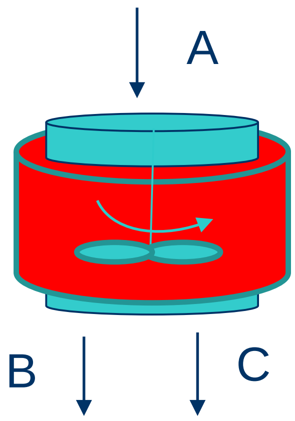
Problem: infinite number of decision variables and constraints





Prof. Cesar de Prada ISA-UVA

Decision variables parameterization

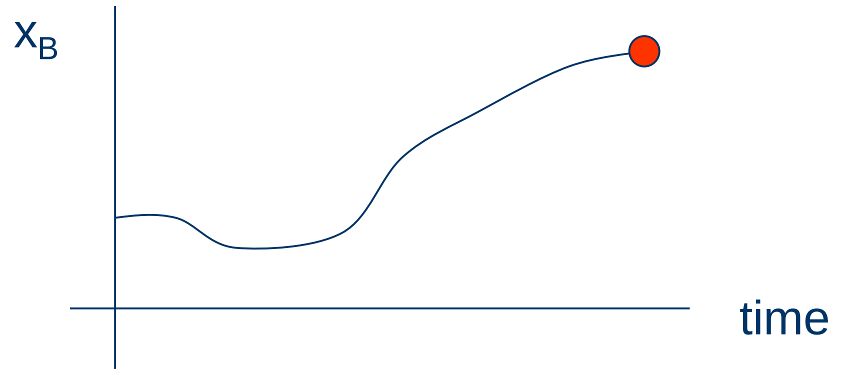


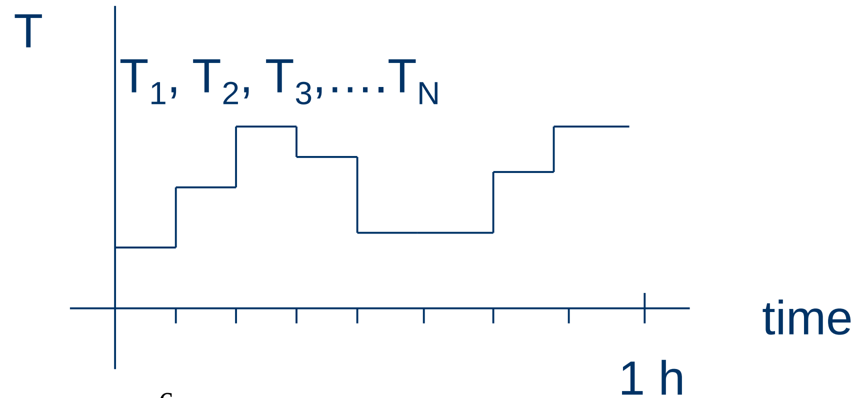
$$\max_{T_i} \quad x_B(1)$$

$$\frac{dx_A}{dt} = -(k_B + k_C)x_A \qquad x_A(0) = A_0$$

$$\frac{dx_B}{dt} = k_B x_A \qquad x_B(0) = 0$$

$$T_i \leq 139 \quad i = 0,1,\ldots,N$$

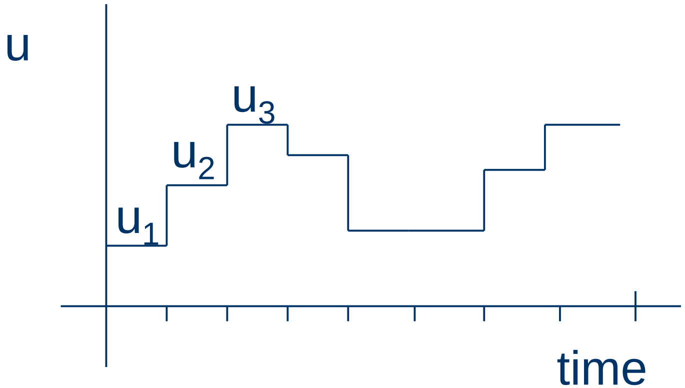


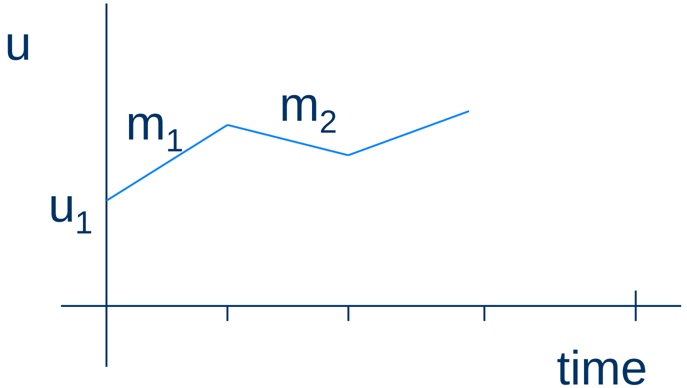


$$k_B = 10^6 \exp(10000 / RT)$$

$$k_C = 5 * 10^{11} \exp(20000 / RT)$$

Control Vector Parameterization CVP



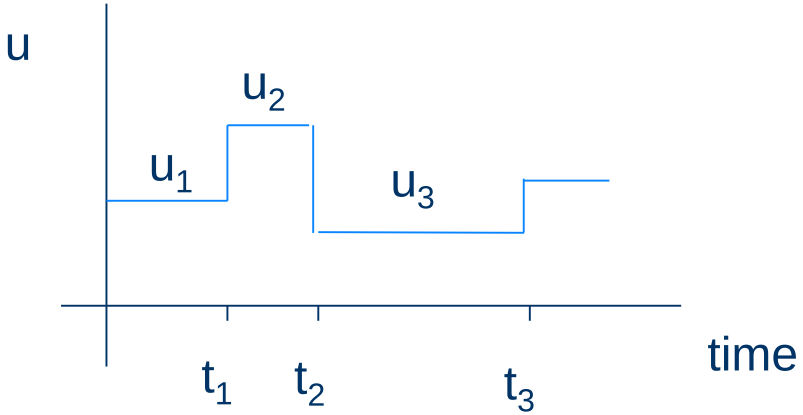


New decision variables

$u_1, u_2, u_3, \ldots . u_N$

$u_1, m_1, m_2, \ldots . m_N$

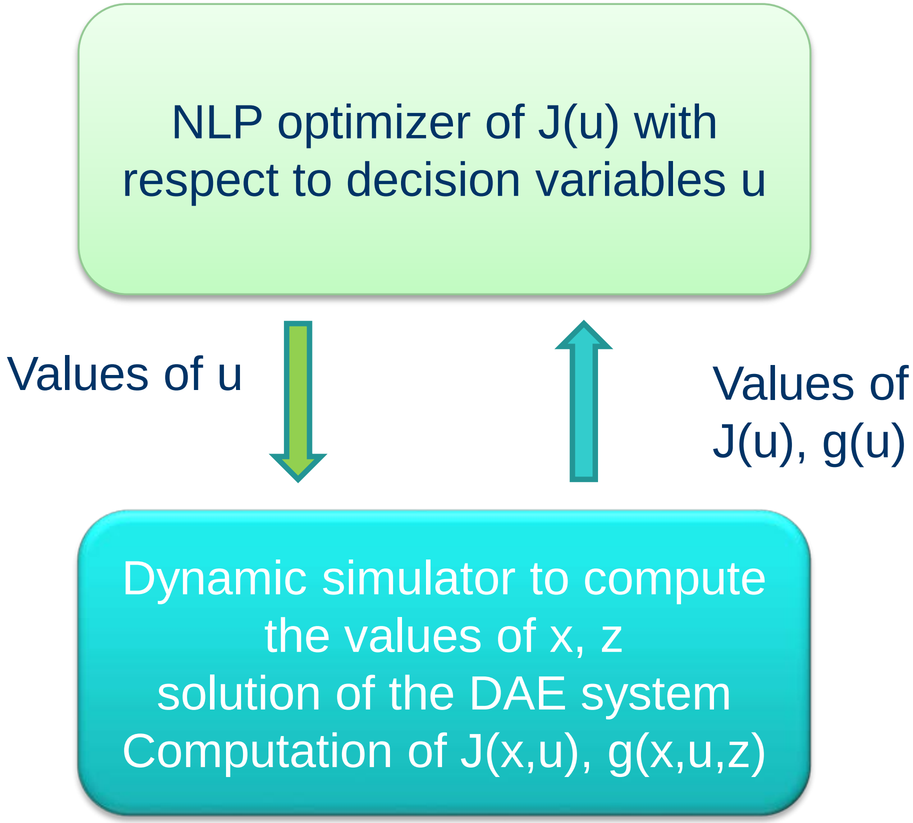
$u_1, t_1, u_2, t_2, \ldots . u_N, t_N$



Solving DO problems

- Two main approaches:
  - Solving the differential equations with a dynamic simulator (Sequential approach) CVP
  - Discretizing the dynamic system to convert it into an algebraic one (Simultaneous approach)
- They are more computational intensive than standard NLP problems

Sequential approach



$$\min_{u(t),x_0,t_f} \quad J(\mathbf{u}) = \int_{t_0}^{t_f} C(\mathbf{x},\mathbf{u})dt$$

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x},\mathbf{u},\mathbf{z}), \quad \mathbf{x}(t_0) = \mathbf{x}_0$$

$$\mathbf{h}(\mathbf{x},\mathbf{u},\mathbf{z}) = \mathbf{0}$$

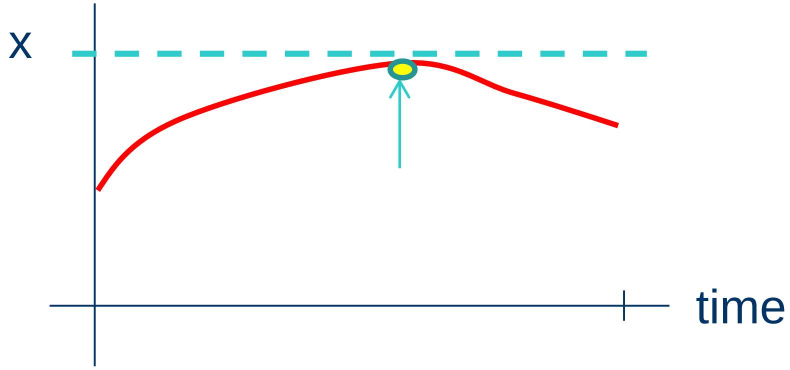
$$\mathbf{g}(\mathbf{x},\mathbf{u},\mathbf{z}) \leq \mathbf{0}$$

The DO problem is converted into a NLP one, with a number of u variables equal to the problem degrees of freedom

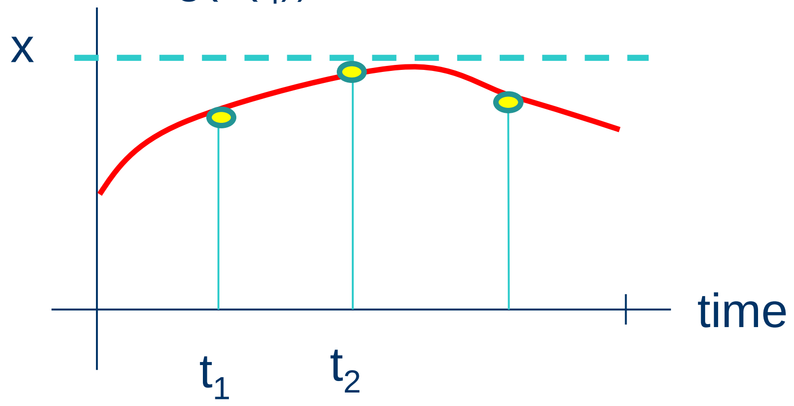
Multiple calls to the simulator from the NLP code

Path constraints

Max g((x(t)) ≤ M



$g(x(t_i)) \leq M$ I = 1, 2,…



The sequential approach uses a smaller number of decision variables (the CVP of u) than the simultaneous one, but imposing constraints over time on states and algebraic variables is more difficult as they are computed inside the simulation

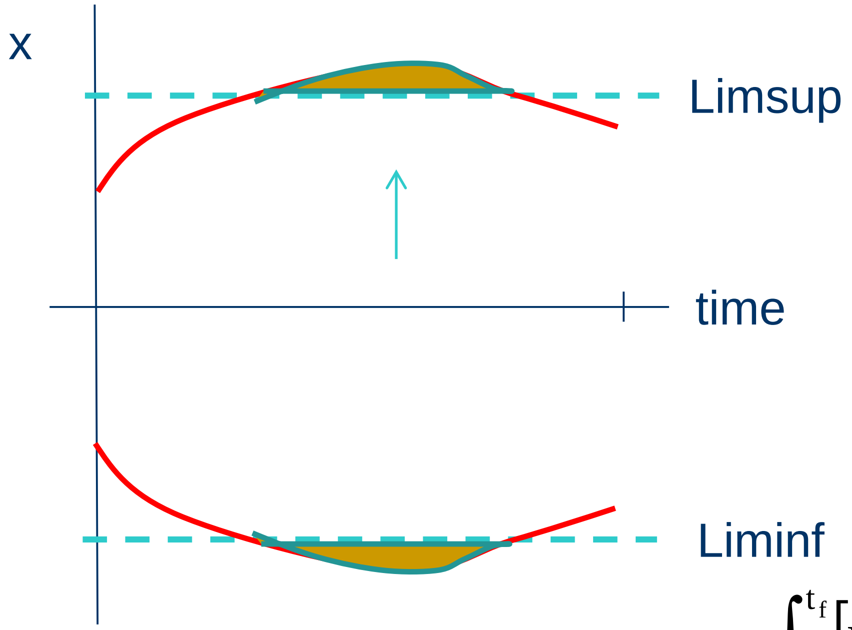
Solutions:

- ✓ Choose several internal points and impose the constraint at this points
- ✓ Compute the max or min of x(t) and impose the constraint on it

Path constraints

Another option

$$\int_{t_0}^{t_f} \left[\max(\text{Lim}\,\text{sup}, \mathbf{x}(t)) - \text{Lim}\,\text{sup}\right] dt \leq 0$$

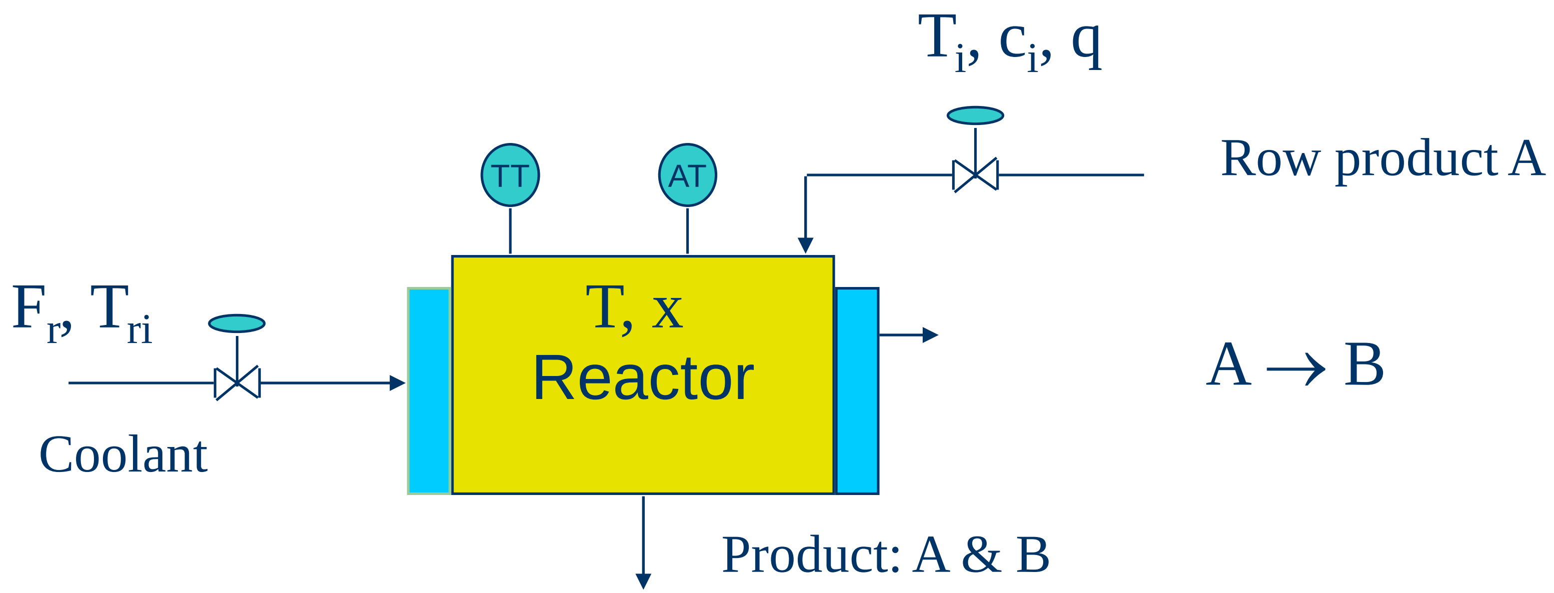


The surface under the trayectory above the upper (or bellow the lower) limit is constrained to be zero

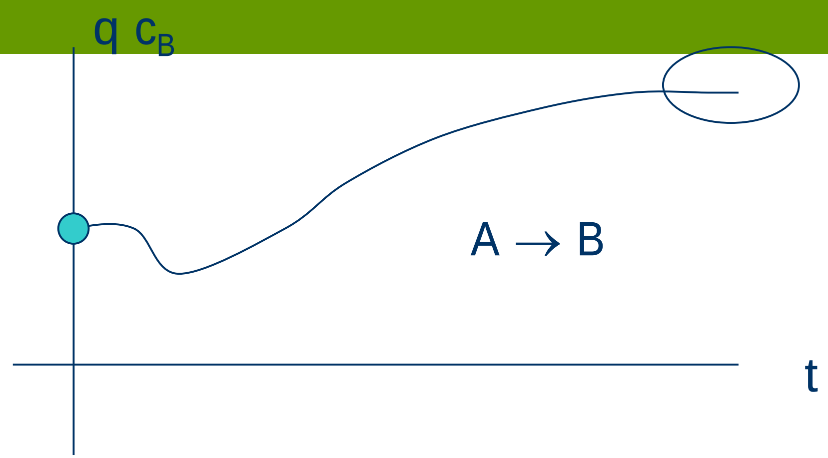
$$-\int_{t_0}^{t_f} \left[\min(\text{Liminf}, \mathbf{x}(t)) - \text{Liminf}\right] dt \leq 0$$

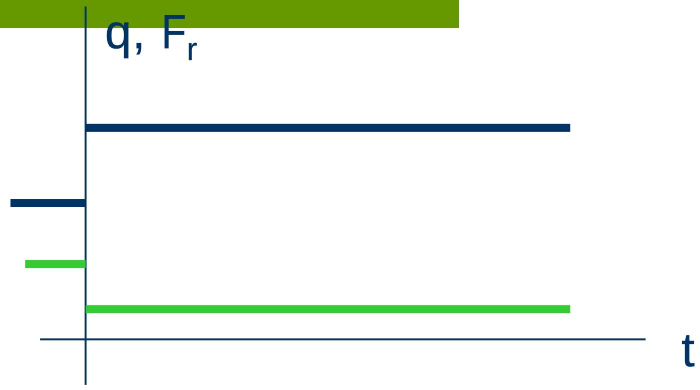
Dynamic Optimization (DO) example

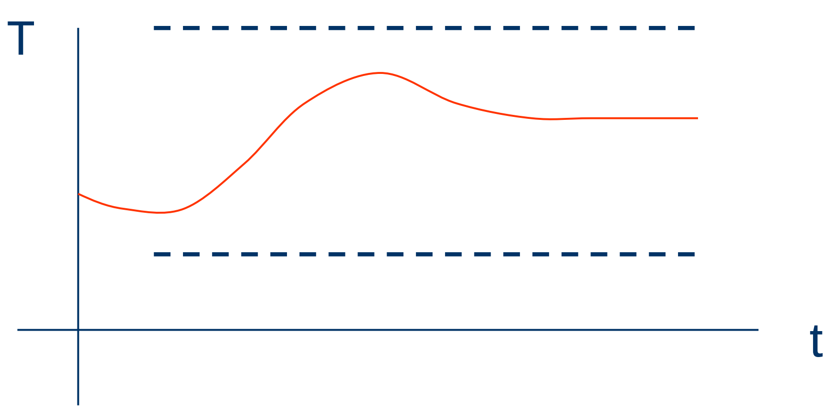
Starting from a certain operation point, and performing a single change in the process MVs, bring the process to the maximum production point respecting a set of constraints over the transient.



Dynamic Optimization







$$\max_{q,F_r} \int q c_B \, dt$$

Dynamic model equations

$$\ldots \leq q \leq \ldots$$

$$\ldots \leq F_r \leq \ldots$$

$$T_{min} \leq T(t) \leq T_{max}$$

$x > 0.7$

DO in EcosimPro

EXPERIMENT

Constraints in the decision variables z

$$L_1 \le u_1 \le U_1$$
$$L_2 \le u_2 \le U_2$$
$$\vdots$$

COMPONENT

PROCESS MODEL

$$\frac{dx(t)}{dt} = f(x(t), u)$$
$$y(t) = g(x(t), u)$$

CONSTRAINTS

$$g_1(x(t), u) \le 0$$
$$g_2(x(t), u) \le 0$$
$$\vdots$$

COST FUNCTION

$$J(u)$$

Decision variables : u_1, u_2, ...

Value of the cost function *J* and constraints g

OPTIMIZATION ALGORITHM

NLP

u^*

The optimization algorithm obtains the values of the cost function J and constraints g when it needs them by calling the dynamic simulation module

Dynanic simulator

Component / Variables

Component name

```
COMPONENT reactor_ab_dynamic_max

DATA
        REAL ....
DECLS
        REAL q = 2.832    “Volumetric Inflow  [m3/h]“
        REAL Fr           “Coolant flow [m3/h]"
        REAL ....
        -- Indice a minimizar
        REAL J_cost
        -- Vector de las restricciones no lineales
        REAL F_optim[4]
........
```

Decision Variables

Cost Function

Constraints

Constraints / Cost function

```
CONTINUOUS
    ... ecuaciones del modelo
    -- balance de masa de A
    V*cA' = q*(cA0 - cA)- V*k*cA
    -- balance de energia en el reactor
    V*rho*Cp*T' = q*rho*Cp*(Te - T) - V*k*cA*DH - Q
    .........
    -- calculo de la funcion de costo a minimizar
    J_costo' = - q*cB

    -- calcular las restricciones no lineales (expresiones g(x) <= 0)
    F_optim[1] = J_costo
    F_optim[2] = 0.75 – x
    F_optim[3] = MATH.min(T,LiminfT) - LiminfT
    F_optim[4] = MATH.max(T,LimsupT) - LimsupT
END COMPONENT
```

Dynamic model

Cost function

End point constraint

Path constraint
Max(T(t)) < LimsupT

Experiment /Functions

```
USE OPTIM_METHODS
EIDAS calculoSens

CONST INTEGER numC = 3          -- numero de restricciones del problema
CONST INTEGER numU = 2          -- numero de variables de decisión
--CONST INTEGER numX = 4        -- número de variables de estado + 1 (costo)

FUNCTION INTEGER coste_y_restricciones (IN REAL esnopt_x[], IN
INTEGER needF, OUT REAL esnopt_F[], IN INTEGER explicit_derivatives,
IN INTEGER needG, OUT REAL esnopt_G[])

…..
END FUNCTION
```

Experiment / Functions

```
FUNCTION NO_TYPE funcionResiduos( …… )
        …….
END FUNCTION

FUNCTION NO_TYPE funcionQuadraturas( ……. )
 ……
        -- funciones de cuadraturas, fijar la dimension de F_optim
        FOR( i IN 1,4 )
                quad[i] = F_optim[i]
        END FOR
END FUNCTION

FUNC_PTR<ptrFunRes> ptrRes = funcionResiduos
FUNC_PTR<ptrFunRes> ptrQuad = funcionQuadraturas
```

Experiment / Variables

```
EXPERIMENT optim ON reactor_ab_dynamic_max.open_loop

  DECLS
        REAL dec_var[numU]          -- valor inicial de las variables de decision
        REAL xlow[numU]             -- valor inferior de las variables de decision
        REAL xupp[numU]             -- valor superior de las variables de decision
        REAL Flow[numC + 1]         -- valor inferior de la funcion objetivo y las
                                               restricciones
        REAL Fupp[numC + 1]         -- valor superior de la funcion objetivo y las
                                               restricciones
        INTEGER calcularSens = 1    --  igual a 1 si se calculan sensibilidades
        INTEGER infoESnopt = 0      --  informacion interna de SNOPT

 OBJECTS
        VECTOR_STRING nombresX   -- variable auxiliar para la inicialización de
                                               las sensibilidades
```

Experiment / INIT

```
INIT
   -- initial values for state variables
   cA = 2.891
   Tr = 51.5
   T = T0
   J_coste = 0

 BOUNDS
   -- Set equations for boundaries: boundVar = f(TIME;...)
   Fr = 50
   q = 2
```

Experiment

```
BODY
…….
                -- inicialización de las variables de decisión, y los límites
                -- addU( nombre de la variable, valor inicial de la misma,
booleano que indica si el parámetro es un valor inicial )
                calculoSens.addU( "q", q, FALSE )
                calculoSens.addU( "Fr", Fr, FALSE )

                dec_var[1] = q
                dec_var[2] = Fr
                xlow[1] = 0.5
                xupp[1] = 5
                xlow[2] = 5
                xupp[2] = 90
```

Specifying the decision variables and their range

Experiment

TIME = 0
TSTOP = 10
CINT = 0.1

-- Inicializacion del algoritmo de integracion y cálculo de sensibilidades

…….

-- Configurar la tolerancia en el cálculo de sensibilidades (opcional)
calculoSens.setTol(1e-5)
-- inicialización de los límites de las restricciones y la función de coste
Flow[1] = -1.0e6
Fupp[1] = 1.0e6
Flow[2] = Liminfx -- End point constraint
Fupp[2] = 1 -- End point constraint
Flow[3] = 0 -- Path constraint
Fupp[3] = 0 -- Path constraint
Flow[4] = 0
Fupp[4] = 0

Experiment /SNOPT

```
esnopt_init (numU, numC)
esnopt_set_variables_bounds_and_initial_values (xlow, xupp, dec_var)
esnopt_set_constraints_bounds_and_initial_values (Flow, Fupp, F_optim)
esnopt_set_cost_function_and_constraints (coste_y_restricciones)
esnopt_set_explicit_derivatives ( calcularSens )
esnopt_set_function_precision ( 1.0e-5 )
esnopt_set_iterations_limit (200)
infoESnopt = esnopt
-- Final de la optimización, obtención de los resultados para la simulacion.
        setSilentMode(FALSE)
        SET_REPORT_ACTIVE("#MONITOR",TRUE)
        esnopt_print_data ()
        esnopt_get_variables_values(dec_var)
        esnopt_free ()
```

Calling the optimizer

Getting the solution

Experiment

```
-- Llamada al integrador
        RESET()
-- Modificación de los parámetros con los nuevos valores obtenidos

        q = dec_var[1]
        Fr = dec_var[2]

        TIME = 0
        CINT = 0.1
        INTEG()
…..
END EXPERIMENT
```

Cost and constraints

```
FUNCTION INTEGER coste_y_restricciones (IN REAL esnopt_x[], , IN
INTEGER needF, OUT REAL esnopt_F[], IN INTEGER explicit_derivatives, IN
INTEGER needG, OUT REAL esnopt_G[])

        DECLS
                ……
        BODY
                ……
-- Actualizar las variables de decisión a los valores que propone el optimizador
                calculoSens.setU( "q", esnopt_x[1] )
                calculoSens.setU( "Fr", esnopt_x[2] )
…….
```

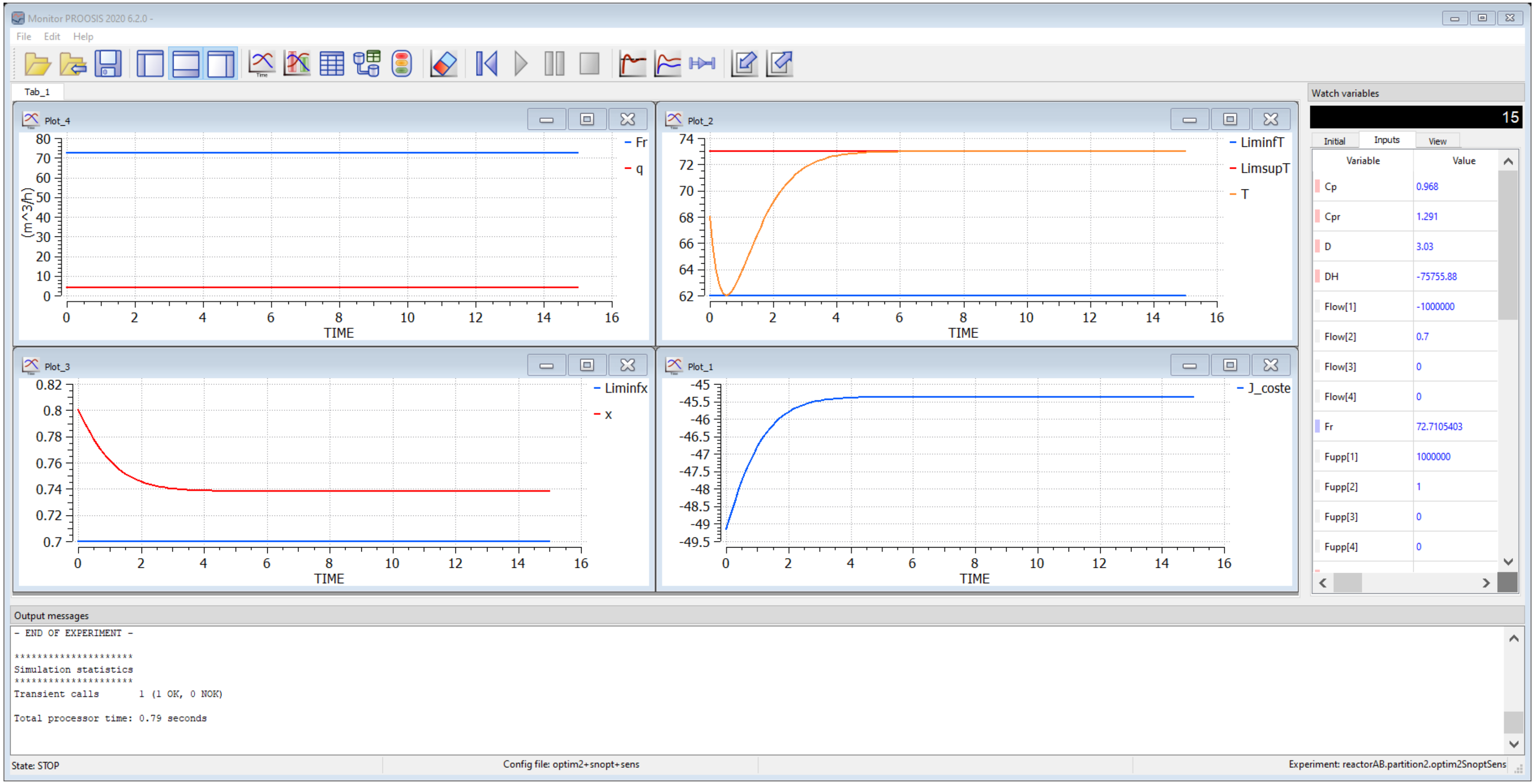
Cost and Constraints

```
--Introduccion de los gradientes de la funcion de costo y restricciones a SNOPT
--si la funcion es de camino, hay que introducir la derivada parcial de la cuadratura
-- ( arr_quadsen ),  en caso contrario, si la restricción es de punto final, al usar
-- arr_quadsen_p se usa la derivada parcial del valor de la función.
                IF ( explicit_derivatives == 1) THEN
-- derivadas de las 4 F_optim respecto a las 2 variables de decision
                esnopt_G[1] = arr_quadsen[ 1 ]    -- der(F-optim[1]/d q
                esnopt_G[2] = arr_quadsen[ 2 ]     -- der(F-optim[1]/d Fr

                esnopt_G[3] = arr_quadsen_p[ 3 ] -- der(F-optim[2]/d q
                esnopt_G[4] = arr_quadsen_p[ 4 ] -- der(F-optim[2]/d Fr

                esnopt_G[5] = arr_quadsen[ 5 ]
                esnopt_G[6] = arr_quadsen[ 6 ]
                esnopt_G[7] = arr_quadsen[ 7 ]
                esnopt_G[8] = arr_quadsen[ 8 ]
```

Cost and Constraints

```
-- Introducción de los valores de las funciones a SNOPT
-- si la funcion es de camino, hay que introducir la cuadratura ( arr_quad )
-- en caso contrario, si la restricción es de punto final, al usar F_optim se
-- usa el valor de la función.
        esnopt_F[1] = arr_quad[1]
        esnopt_F[2] = F_optim[2]
        esnopt_F[3] = arr_quad[3]
        esnopt_F[4] = arr_quad[4]

        RETURN 0
END FUNCTION
```

Results in EcosimPro (EcoMonitor)



Simultaneous approach

- It is based in the discretization of the equations

$$\min_{u(t),x(t),x_0,t_f} \quad J(\mathbf{u}) = \int_{t_0}^{t_f} C(\mathbf{x},\mathbf{u})dt$$

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x},\mathbf{u},\mathbf{z}), \quad \mathbf{x}(t_0) = \mathbf{x}_0$$

$$\mathbf{h}(\mathbf{x},\mathbf{u},\mathbf{z}) = \mathbf{0}$$

$$\mathbf{g}(\mathbf{x},\mathbf{u},\mathbf{z}) \leq \mathbf{0}$$

$\Rightarrow$

$$\min_{\mathbf{x}_k,\mathbf{u}_k,\mathbf{z}_k} J = \sum_{j=1}^{N} C(\mathbf{x}_k,\mathbf{u}_k)\Delta_k$$

$$\mathbf{x}_{k+1} = \tilde{\mathbf{f}}(\mathbf{x}_k,\mathbf{u}_k,\mathbf{z}_k)$$

$$\mathbf{h}(\mathbf{x}_k,\mathbf{u}_k,\mathbf{z}_k) = 0$$

$$\mathbf{g}(\mathbf{x}_k,\mathbf{u}_k,\mathbf{z}_k) \leq 0$$

$$k = 0,1,2,\ldots.N$$

The discretized model has only algebraic equations and can be solved with NLP methods.

Discretization

One important problem associated with the simultaneous approach is the discretization of the differential equations

$$\min_{u(t),x(t),x_0,t_f} \quad J(\mathbf{u}) = \int_{t_0}^{t_f} C(\mathbf{x},\mathbf{u})dt$$

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x},\mathbf{u},\mathbf{z}), \quad \mathbf{x}(t_0) = \mathbf{x}_0$$

$$\mathbf{h}(\mathbf{x},\mathbf{u},\mathbf{z}) = \mathbf{0}$$

$$\mathbf{g}(\mathbf{x},\mathbf{u},\mathbf{z}) \leq \mathbf{0}$$

Simple methods, such as the Euler discretization are not robust and lead to numerical problems with stiff systems

$$\frac{d\mathbf{x}}{dt} \approx \frac{\mathbf{x}(t+\Delta_t)-\mathbf{x}(t)}{\Delta_t} = \frac{\mathbf{x}_{k+1}-\mathbf{x}_k}{\Delta_t}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{f}(\mathbf{x}_k,\mathbf{u}_k,\mathbf{z}_k)\Delta_t$$

Other methods such as higher order implicit integration ones or collocation methods should be used

Simultaneous approach

$$\min_{\mathbf{x}_k,\mathbf{u}_k,\mathbf{z}_k} J = \sum_{j=1}^{N} C(\mathbf{x}_k,\mathbf{u}_k)\Delta_k$$

$$\mathbf{x}_{k+1} = \tilde{\mathbf{f}}(\mathbf{x}_k,\mathbf{u}_k,\mathbf{z}_k)$$

$$\mathbf{h}(\mathbf{x}_k,\mathbf{u}_k,\mathbf{z}_k) = 0$$

$$\mathbf{g}(\mathbf{x}_k,\mathbf{u}_k,\mathbf{z}_k) \leq 0$$

$$k = 0,1,2,\ldots.N$$

$$\min_{\mathbf{x}_k,\mathbf{u}_k,\mathbf{z}_k} J = \sum_{j=1}^{N} C(\mathbf{x}_k,\mathbf{u}_k)\Delta_k$$

$$\mathbf{x}_1 = \tilde{\mathbf{f}}(\mathbf{x}_0,\mathbf{u}_0,\mathbf{z}_0)$$

$$\mathbf{x}_2 = \tilde{\mathbf{f}}(\mathbf{x}_1,\mathbf{u}_1,\mathbf{z}_1)$$

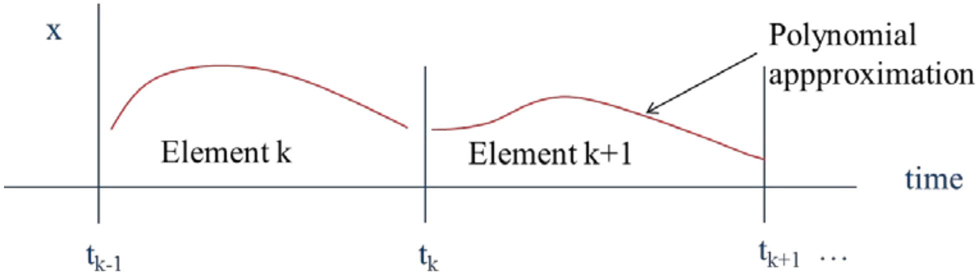
$$\mathbf{x}_3 = \tilde{\mathbf{f}}(\mathbf{x}_2,\mathbf{u}_2,\mathbf{z}_2)$$

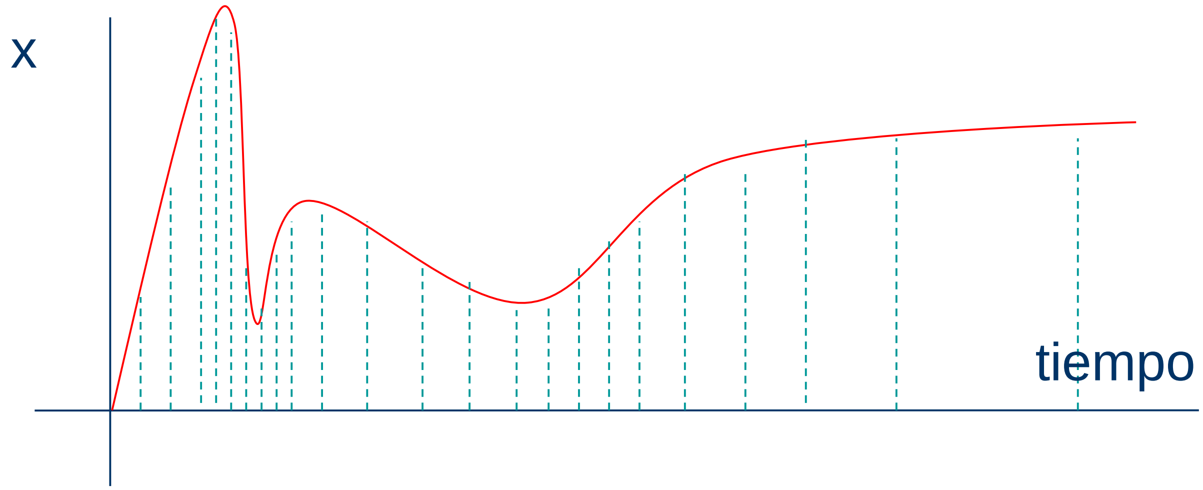
$$\ldots..$$

The number of equations increases by a factor of N and the number of decision variables increases from the CVP of u to u_k, x_k, z_k with respect to the sequential approach

But it is easier to impose constraints on the time evolution of the states and algebraic variables (path constraints) by limiting , x_k, z_k at the discretization points

Discretización





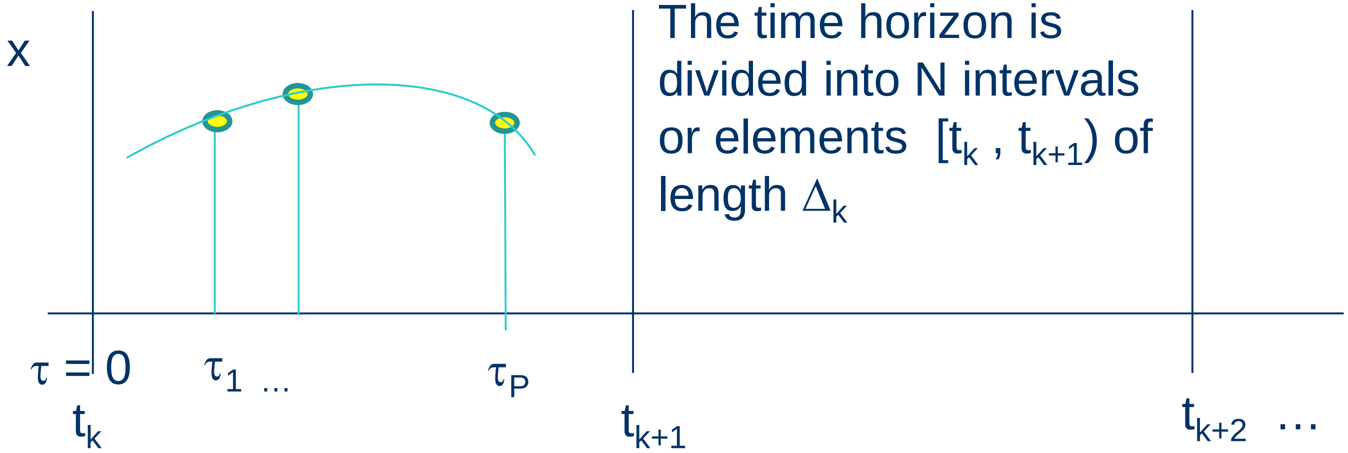
Colocación ortogonal por elementos finitos

¿nº de elementos?

La integración de sistemas stiff usa métodos de paso y estructura variable para mantener el error de integración bajo cotas.

El uso de métodos de paso fijo obliga a usar un gran número de intervalos, resultando en un alto número de ecuaciones y variables y no garantiza la calidad

Collocation on finite elements



The time horizon is divided into N intervals or elements $[t_k , t_{k+1})$ of length Δ_k

The time evolution of the variables is approximated by polynomial interpolation on the values of the variable on P+1 collocation points located at fixed positions τ_j in every element k. Different methods exist

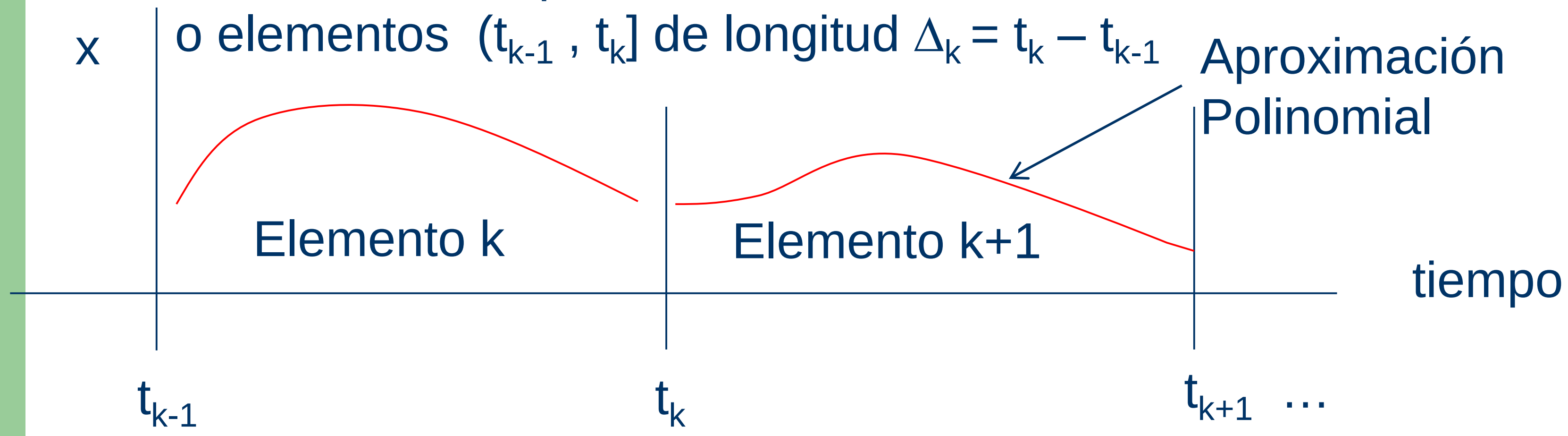
$$\mathbf{x}(t) \approx \sum_{j=0}^{P} P_j(\tau)\mathbf{x}_{kj}$$

$$t = t_k + \tau\Delta_k \qquad \tau \in [0,1)$$

$$\dot{\mathbf{x}}(t) \approx \sum_{j=0}^{P} \frac{\dot{P}_j(\tau)\mathbf{x}_{kj}}{\Delta_k}$$

Colocación en elementos finitos

El horizonte temporal se divide en K intervalos o elementos $(t_{k-1}, t_k]$ de longitud $\Delta_k = t_k - t_{k-1}$

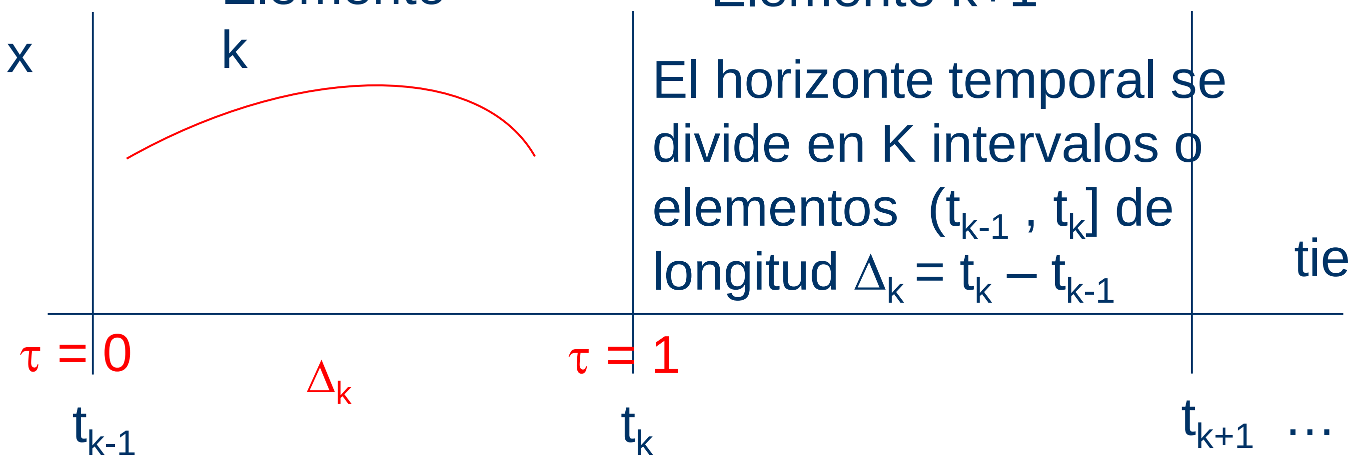


En cada intervalo $(t_{k-1}, t_k]$ la solución x se aproxima por una fórmula polinómica. Esto proporciona una aproximación suave en elemento, al tiempo que permite discontinuidades en la señal de control.

Pueden usarse muchos tipos de aproximaciones polinómicas

El número de elementos r K no tiene por que ser grande

Colocación en elementos finitos



El horizonte temporal se divide en K intervalos o elementos $(t_{k-1}, t_k]$ de longitud $\Delta_k = t_k - t_{k-1}$

x_{kj} parámetros a calcular

Una posibilidad es aproximar la evolución temporal de las variables por una combinación lineal de polinomios conocidos $P_j(\tau)$ de orden P. Típicamente se usan polinomios de interpolación de Lagrange.

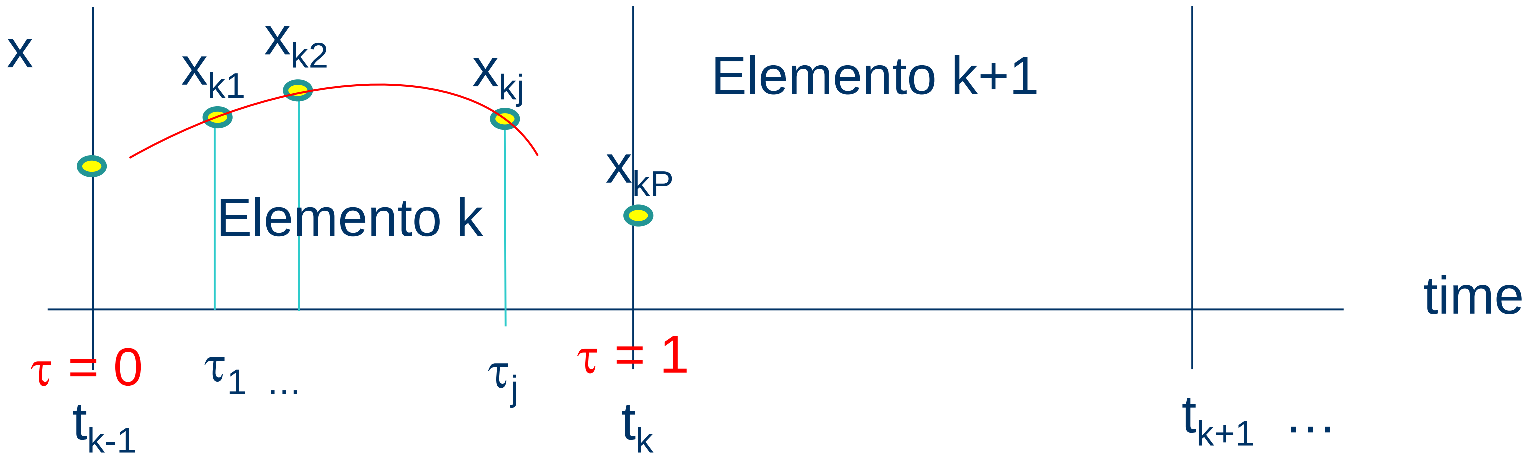
$$\mathbf{x}(t) \approx \sum_{j=0}^{P} P_j(\tau)\mathbf{x}_{kj}$$

$$t = t_{k-1} + \tau\Delta_k \qquad \tau \in (0,1] \qquad k = 1,\ldots,K$$

$$\dot{\mathbf{x}}(t) \approx \sum_{j=0}^{P} \frac{\dot{P}_j(\tau)\mathbf{x}_{kj}}{\Delta_k}$$

τ Tiempo normalizado

Polinomios de interpolación de Lagrange



Se seleccionan P+1 puntos de interpolación $\tau_0 = 0, \tau_1, .., \tau_P$

$\tau_i < \tau_{i+1}$

$$\mathbf{x}(t) \approx \sum_{j=0}^{P} P_j(\tau)\mathbf{x}_{kj}$$

$$P_j(\tau) = \prod_{i=0, i\neq j}^{P} \frac{\tau - \tau_i}{\tau_j - \tau_i}$$

$$t = t_{k-1} + \tau\Delta_k \qquad \tau \in (0,1]$$

$$x(t_{kj} = \tau_j) = x(t_{k-1} + \tau_j\Delta_k) = x_{kj}$$

$$\dot{\mathbf{x}}(t) \approx \sum_{j=0}^{P} \frac{\dot{P}_j(\tau)\mathbf{x}_{kj}}{\Delta_k}$$

Los parámetros x_{kj} tienen un significado claro cuando se usan los polinomios de Lagrange

Polinomios de Lagrange

$$P_j(\tau) = \prod_{i=0, i\neq j}^{P} \frac{\tau - \tau_i}{\tau_j - \tau_i}$$

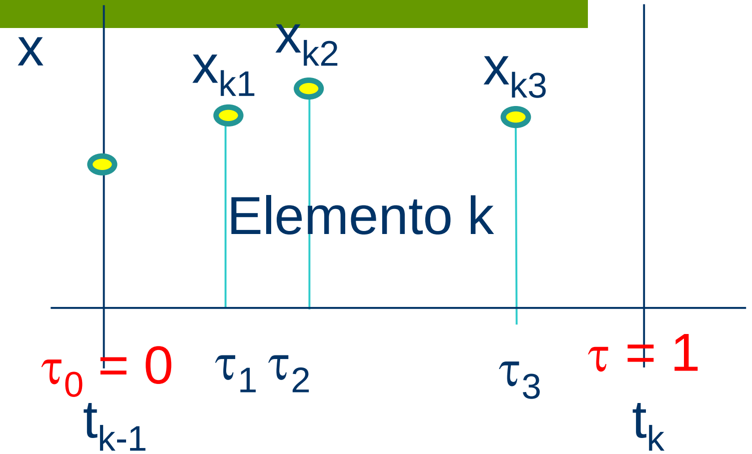
$$P_0 = \frac{\tau - \tau_1}{\tau_0 - \tau_1} \frac{\tau - \tau_2}{\tau_0 - \tau_2} \frac{\tau - \tau_3}{\tau_0 - \tau_3}$$

$$P_1 = \frac{\tau - \tau_0}{\tau_1 - \tau_0} \frac{\tau - \tau_2}{\tau_1 - \tau_2} \frac{\tau - \tau_3}{\tau_1 - \tau_3}$$

$$P_2 = \frac{\tau - \tau_0}{\tau_2 - \tau_0} \frac{\tau - \tau_1}{\tau_2 - \tau_1} \frac{\tau - \tau_3}{\tau_2 - \tau_3}$$

$$P_3 = \frac{\tau - \tau_0}{\tau_3 - \tau_0} \frac{\tau - \tau_1}{\tau_3 - \tau_1} \frac{\tau - \tau_2}{\tau_3 - \tau_2}$$

$$x(t_{k-1} + \tau_j \Delta_k) = x_{kj}$$



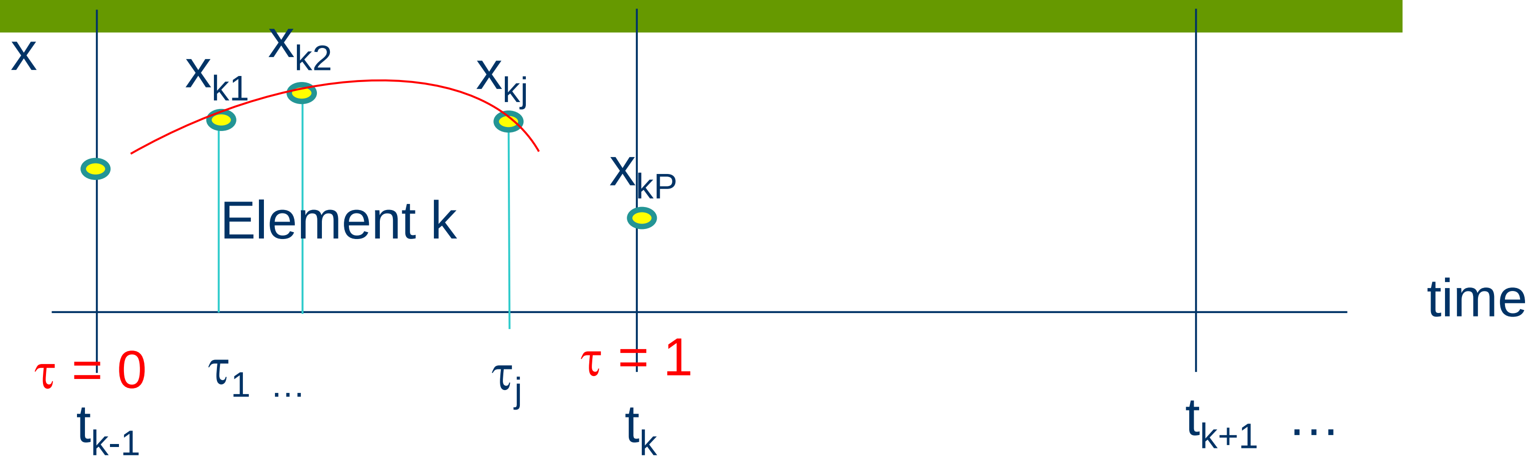
Ejemplo con P= 3 $\quad \mathbf{x}(t) \approx \sum_{j=0}^{P} P_j(\tau)\mathbf{x}_{kj}$

Para $\tau = \tau_1$ $\quad P_0 = P_2 = P_3 = 0 \qquad P_1 = 1$

$x(t_{k-1} + \tau_1 \Delta_k) =$

$= P_0\, x_{k0} + P_1\, x_{k1} + P_2\, x_{k2} + P_3\, x_{k3} = x_{k1}$

Colocación en elementos finitos



$$F(\dot{x}, x, u(p)) = 0$$

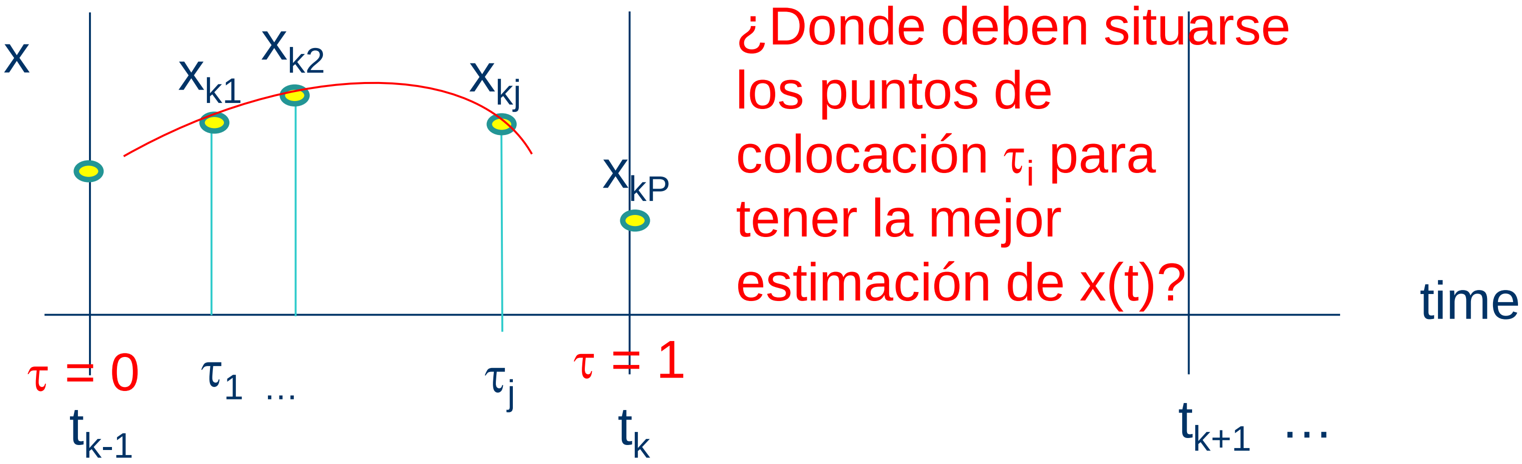
$$F\left(\sum_{j=0}^{P} \frac{\dot{P}_j(\tau_i)\mathbf{x}_{kj}}{\Delta_k}, \mathbf{x}_{ki}, u(p)\right) = 0 \qquad k = 1,..K$$

Se impone que se satisfagan las ecuaciones DAE en los puntos de colocación.

Esta condición proporciona un conjunto de ecuaciones que permiten calcular los coeficientes x_{ki} desconocidos

Los P+1 puntos de colocación se sitúan en posiciones fijas τ_i en cada elemento k. Existen diferentes métodos para situarlos

Colocación Ortogonal



$$F\left(\sum_{j=0}^{P} \frac{\dot{P}_j(\tau_i)\mathbf{x}_{kj}}{\Delta_k}, \mathbf{x}_{ki}, u(p)\right) = 0$$

$k = 1,..K$
$i = 1, \ldots P$

Para reducir el número de polinomios (P) se escogen polinomios ortogonales

$$\int_0^1 P_j(\tau) P_i(\tau)\, d\tau = 0 \quad i \neq j$$

Colocación Ortogonal

Shifted Gauss–Legendre and Radau roots as collocation points.

De P K	Legendre Roots	Radau Roots
1	0.500000	1.000000
2	0.211325	0.333333
	0.788675	1.000000
3	0.112702	0.155051
	0.500000	0.644949
	0.887298	1.000000
4	0.069432	0.088588
	0.330009	0.409467
	0.669991	0.787659
	0.930568	1.000000
5	0.046910	0.057104
	0.230765	0.276843
	0.500000	0.583590
	0.769235	0.860240
	0.953090	1.000000

τ_0 es siempre = 0

Los puntos de colocación τ_i, i = 1,…,P se seleccionan como las raíces de polinomios de tipo Gauss-Jacobi, típicamente:

$$P_P^{Legendre}(\tau) = \sum_{j=0}^{P} (-1)^{P-j} \tau^j \gamma_j$$

$$\gamma_0 = 1$$

$$\gamma_j = \frac{(P-j+1)(P+j)}{j^2}$$

Dan mas exactitud

$$P_P^{Radau}(\tau) = \sum_{j=0}^{P} (-1)^{P-j} \tau^j \gamma_j$$

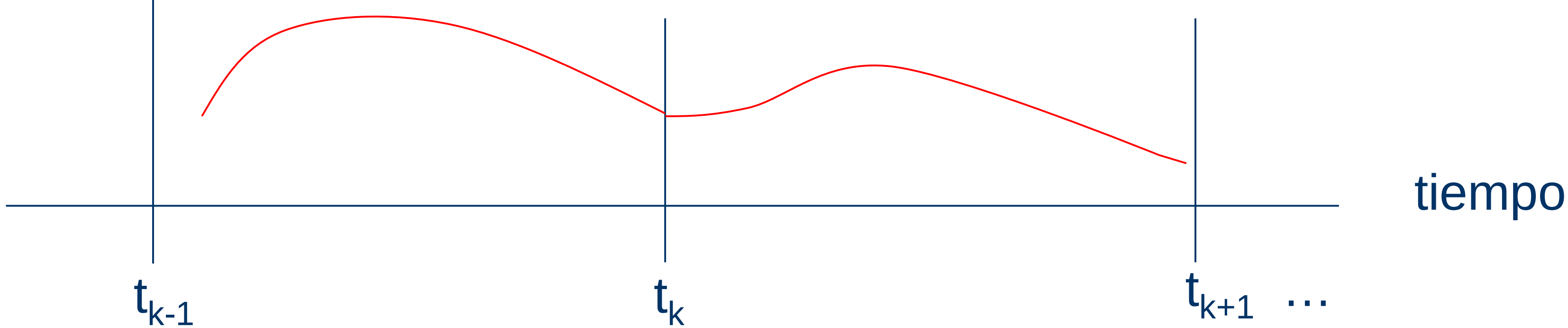
$$\gamma_0 = 1$$

$$\gamma_j = \frac{(P-j+1)(P+j+1)}{j^2}$$

Dan mas robustez

Colocación Ortogonal

Se impone la continuidad de las trayectorias a lo largo de los elementos finitos $(t_{k-1}, t_k]$ mediante:



$$F\left(\sum_{j=0}^{P} \frac{\dot{P}_j(\tau_i)\mathbf{x}_{kj}}{\Delta_k}, \mathbf{x}_{ki}, u(p)\right) = 0$$

k = 1,..K
i = 1, …P

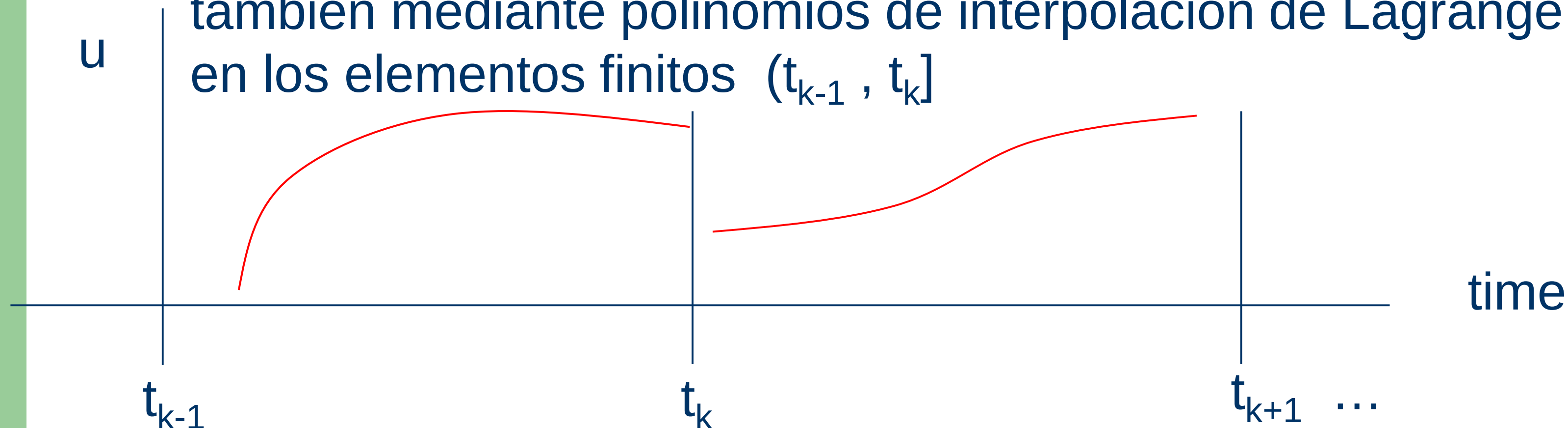
En lugar de estas ecuaciones, en los puntos $\tau_0 = 0$ se usa la continuidad de los estados, y en t = 0 las condiciones iniciales para generar ecuaciones que las sustituyan y que garanticen soluciones acorde a lo deseado

$$\mathbf{x}(t_k) = \mathbf{x}_{k+1,0} = \mathbf{x}_{k,P}$$

$$\mathbf{x}(t_0) = \mathbf{x}_{10} = \mathbf{x}_0$$

Colocación Ortogonal

Si se desea, las variables de control pueden representarse también mediante polinomios de interpolación de Lagrange en los elementos finitos $(t_{k-1}, t_k]$



$$\mathbf{u}(t) \approx \sum_{j=1}^{P} \overline{P}_j(\tau)\mathbf{u}_{kj}$$

$$\overline{P}_j(\tau) = \prod_{i=1, i\neq j}^{P} \frac{\tau - \tau_i}{\tau_j - \tau_i}$$

$$t = t_{k-1} + \tau\Delta_k \qquad \tau \in (0,1]$$

No se impone la continuidad de las trayectorias de control en los elementos finitos $(t_{k-1}, t_k]$

Pueden usarse métodos simultáneos de optimización con sistemas inestables

Ejemplo

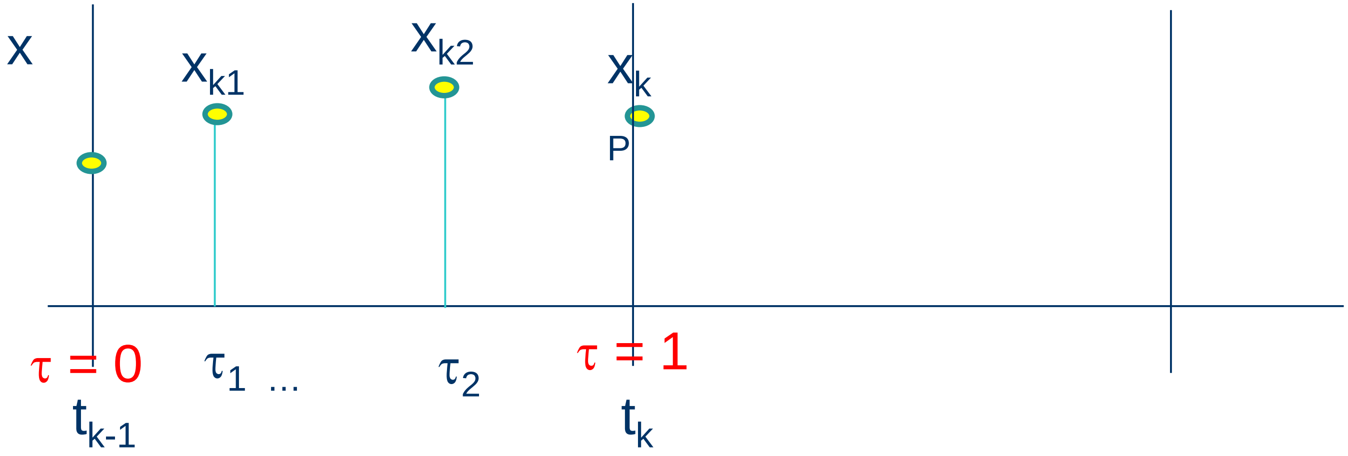
Integrar entre t = 0 y 1

$$\dot{x} = x^2 - 2x + 1 \qquad x(0) = -3$$

Se seleccionan K = 2 elementos finitos de igual tamaño

Δ_k = (1 – 0)/2 = 0.5

P = 3 puntos de colocación



Los puntos de colocación de Radau para P =3 son:

$\tau_0 = 0 \quad \tau_1 = 0.155051 \quad \tau_2 = 0.644949 \quad \tau_3 = 1$

Ejemplo

Los puntos de colocación de Radau para P =3 son:

$\tau_0 = 0 \quad \tau_1 = 0.155051 \quad \tau_2 = 0.644949 \quad \tau_3 = 1$

$$P_j(\tau) = \prod_{i=0, i\neq j}^{P} \frac{\tau - \tau_i}{\tau_j - \tau_i}$$

$$P_0 = \frac{\tau - \tau_1}{\tau_0 - \tau_1}\frac{\tau - \tau_2}{\tau_0 - \tau_2}\frac{\tau - \tau_3}{\tau_0 - \tau_3} = -10\tau^3 + 18\tau^2 - 9\tau + 1$$

$$P_1 = \frac{\tau - \tau_0}{\tau_1 - \tau_0}\frac{\tau - \tau_2}{\tau_1 - \tau_2}\frac{\tau - \tau_3}{\tau_1 - \tau_3} = 15.5808\,\tau^3 - 25.6296\tau^2 + 10.0488\tau$$

$$P_2 = \frac{\tau - \tau_0}{\tau_2 - \tau_0}\frac{\tau - \tau_1}{\tau_2 - \tau_1}\frac{\tau - \tau_3}{\tau_2 - \tau_3} = -8.9141\tau^3 + 10.2963\tau^2 - 1.3821\tau$$

$$P_3 = \frac{\tau - \tau_0}{\tau_3 - \tau_0}\frac{\tau - \tau_1}{\tau_3 - \tau_1}\frac{\tau - \tau_2}{\tau_3 - \tau_2} = 3.3333\tau^3 - 2.6667\tau^2 + 0.3333\tau$$

$$x(t_{k-1} + \tau_j \Delta_k) = x_{kj} \qquad \mathbf{x}(t) \approx \sum_{j=0}^{P} P_j(\tau)\mathbf{x}_{kj} \qquad t = t_{k-1} + \tau\Delta_k \qquad \tau \in (0,1]$$

Ejemplo

Los puntos de colocación de Radau para P =3 son:

$\tau_0 = 0$ $\tau_1 = 0.155051$ $\tau_2 = 0.644949$ $\tau_3 = 1$

$$\dot{\mathbf{x}}(t) \approx \sum_{j=0}^{P} \frac{\dot{P}_j(\tau)\mathbf{x}_{kj}}{\Delta_k}$$

$$\dot{P}_0(\tau) = -30\tau^2 + 36\tau - 9$$

$$\dot{P}_1(\tau) = 46.7423\tau^2 - 51.2592\tau + 10.0488$$

$$\dot{P}_2(\tau) = -26.7423\tau^2 + 20.5925\tau - 1.3821$$

$$\dot{P}_3(\tau) = 10\tau^2 - 5.3333\tau + 0.3333$$

$$\mathbf{x}(t_{k-1} + \tau_j \Delta_k) = \mathbf{x}_{kj}$$

$$t = t_{k-1} + \tau\Delta_k \qquad \tau \in (0,1]$$

$$\dot{x} = x^2 - 2x + 1 \qquad x(0) = -3$$

$$\sum_{j=0}^{3} \frac{\dot{P}_j(\tau)\mathbf{x}_{kj}}{0.5} = \mathbf{x}^2 - 2\mathbf{x} + 1$$

k = 1,2

Ejemplo

$$\dot{x} = x^2 - 2x + 1 \qquad x(0) = -3 \quad \Rightarrow \quad \sum_{j=0}^{3} \frac{\dot{P}_j(\tau)\mathbf{x}_{kj}}{0.5} = \mathbf{x}^2 - 2\mathbf{x} + 1$$

k = 1,2

En los puntos de colocación τ_i :

$$\sum_{j=0}^{3} \frac{\dot{P}_j(\tau_i)\mathbf{x}_{kj}}{0.5} = \mathbf{x}_{ki}^2 - 2\mathbf{x}_{ki} + 1$$

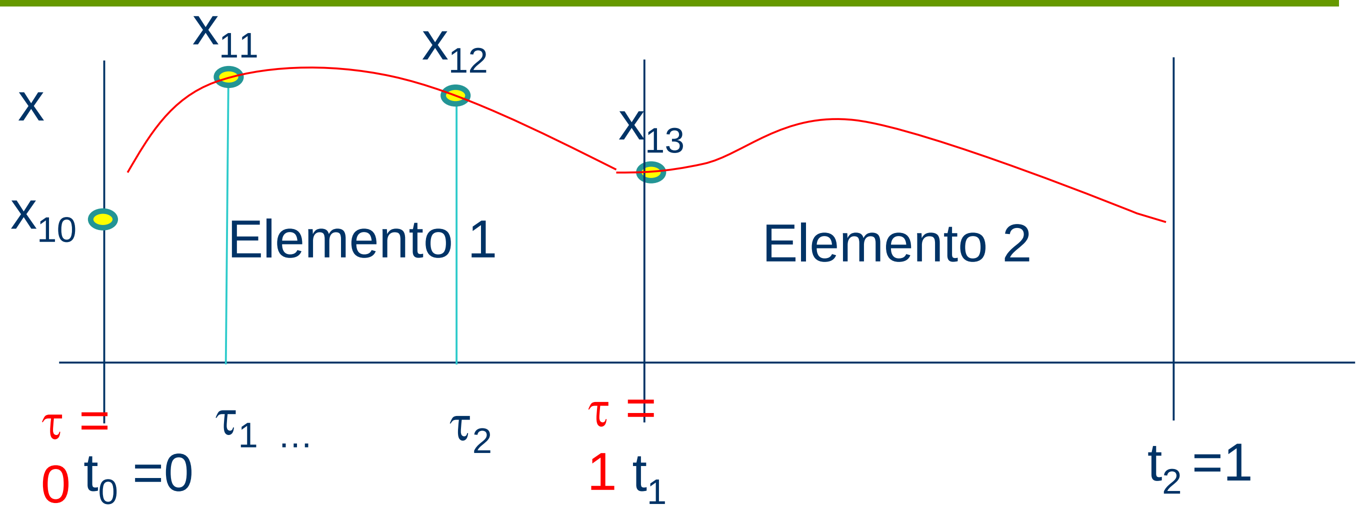
k = 1,2
i = 1, …3

$$(-30\tau_i^2 + 36\tau_i - 9)x_{10} + (46.7423\tau_i^2 - 51.2592\tau_i + 10.0488)x_{11} +$$
$$+ (-26.7423\tau_i^2 + 20.5925\tau_i - 1.3821)x_{12} + (10\tau_i^2 - 5.3333\tau_i + 0.3333)x_{13} =$$
$$= 0.5(x_{1i}^2 - 2x_{1i} + 1) \qquad i = 1,2,3$$

$$(-30\tau_i^2 + 36\tau_i - 9)x_{20} + (46.7423\tau_i^2 - 51.2592\tau_i + 10.0488)x_{21} +$$
$$+ (-26.7423\tau_i^2 + 20.5925\tau_i - 1.3821)x_{22} + (10\tau_i^2 - 5.3333\tau_i + 0.3333)x_{23} =$$
$$= 0.5(x_{2i}^2 - 2x_{2i} + 1) \qquad i = 1,2,3$$

8 incógnitas, 6 ecuaciones

Ejemplo



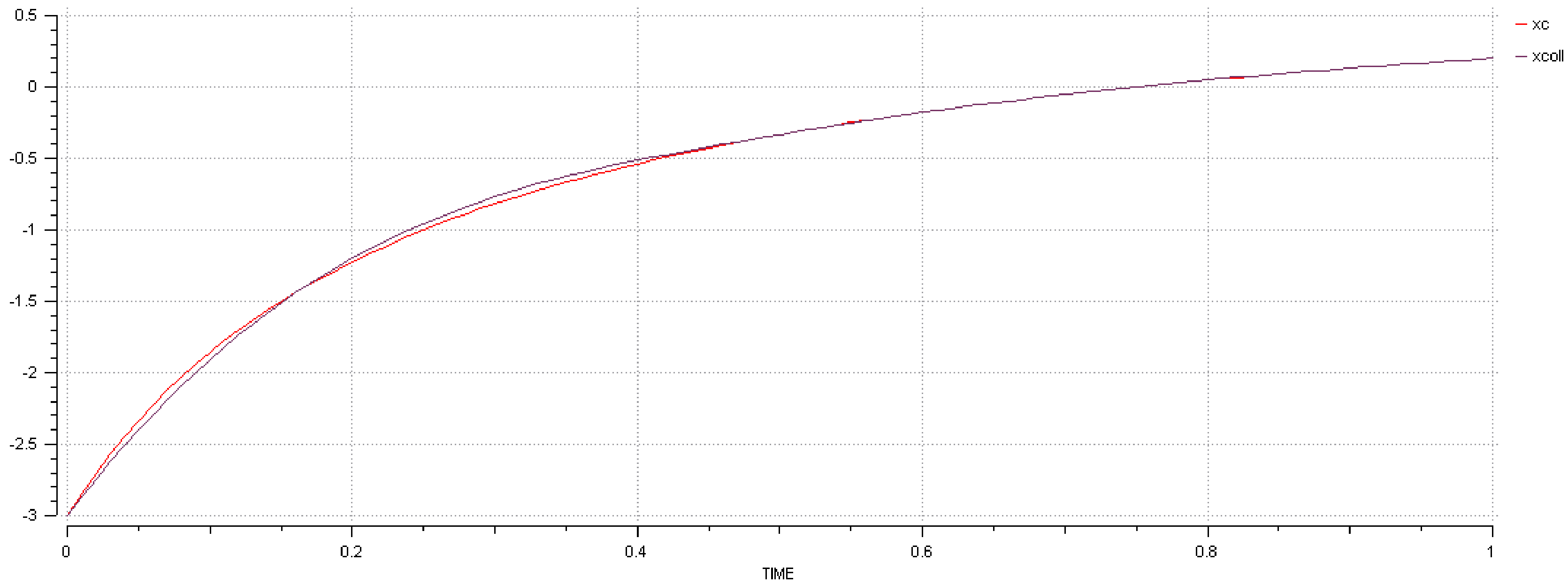
$$\mathbf{x}(t_k) = \mathbf{x}_{k+1,0} = \mathbf{x}_{k,P} = \sum_{j=0}^{P} P_j(1)\mathbf{x}_{k,j} \qquad \mathbf{x}(0.5) = \mathbf{x}_{20} = \mathbf{x}_{13} = \sum_{j=0}^{3} P_j(1)\mathbf{x}_{1j}$$

$$\mathbf{x}(t_0) = \mathbf{x}_{10} = \mathbf{x}_0 \qquad \mathbf{x}(0) = \mathbf{x}_{10} = -3$$

8 incógnitas, 8 ecuaciones

Las condiciones iniciales y de continuidad proporcionan las otras dos ecuaciones

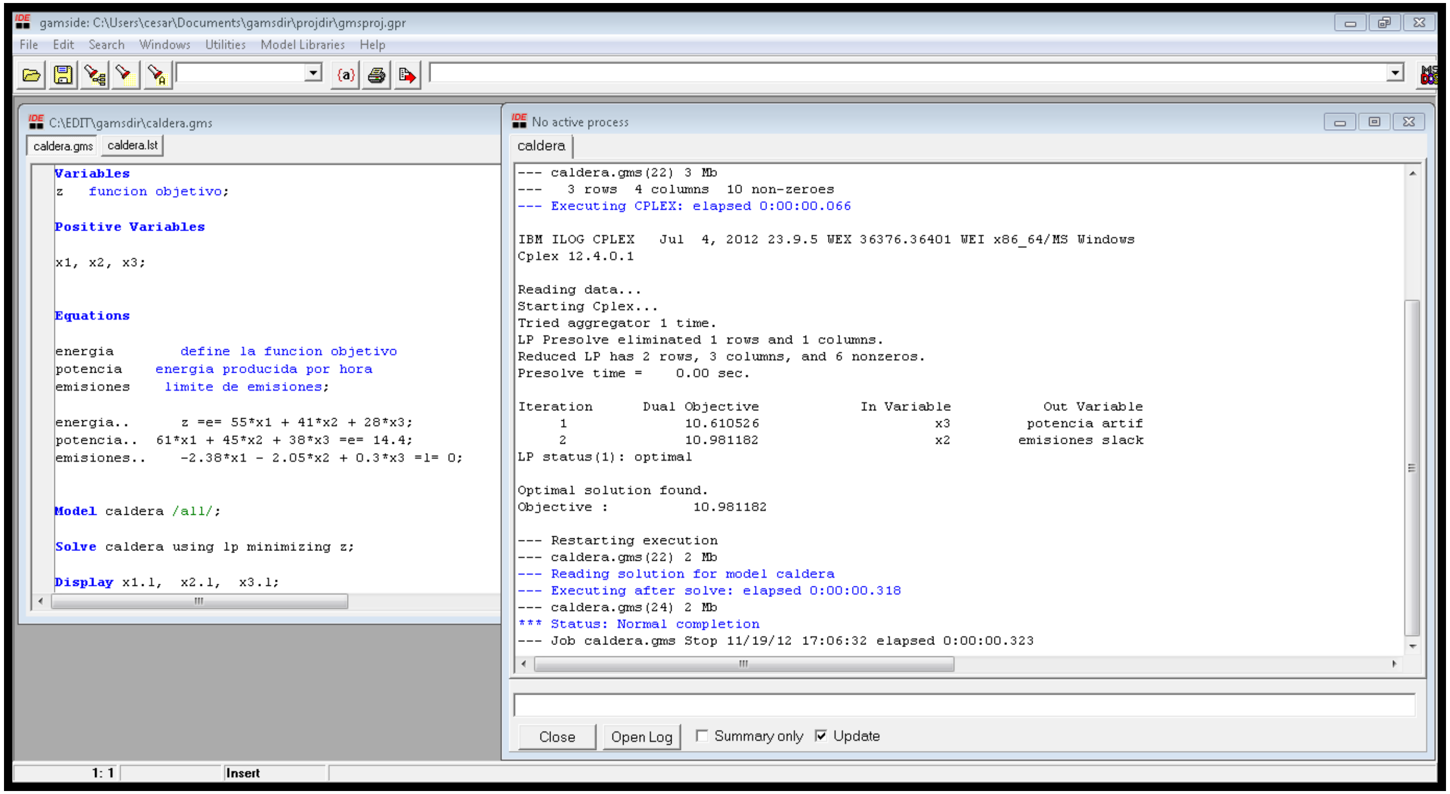
Ejemplo



$$\dot{x} = x^2 - 2x + 1 \qquad x(0) = -3$$

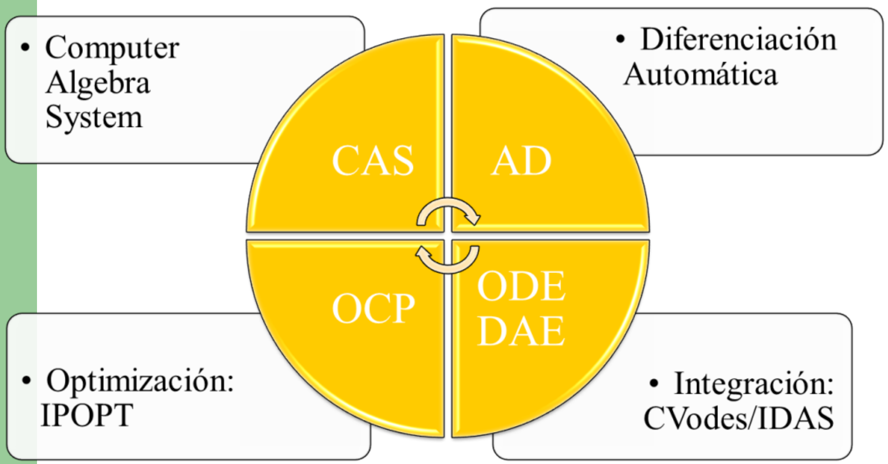
Respuestas analítica y obtenida por colocación ortogonal

Software: GAMS



Entornos de modelado y optimización como GAMS, AIMMS, XPRESS, Gurobi,… pueden usarse tras la discretización

Software



CasADi es un entorno simbólico para optimización numérica que facilita la discretización e implementa diferenciación automática (gradientes y Hesianos).

Genera código C e implementa interfaces a códigos DAE y de optimización como SUNDIALS, IPOPT etc.

Se gestiona desde una interfaz con Python/Matlab

Solución eficiente de problemas de gran escala

Pero no soporta:

- Discontinuidades
- Optimización mixta-entera

Problemas de memoria

Entorno pobre de modelado

Computational Infrastructure for Operations Research (COIN-OR) Open source codes

Sensibilidades paramétricas

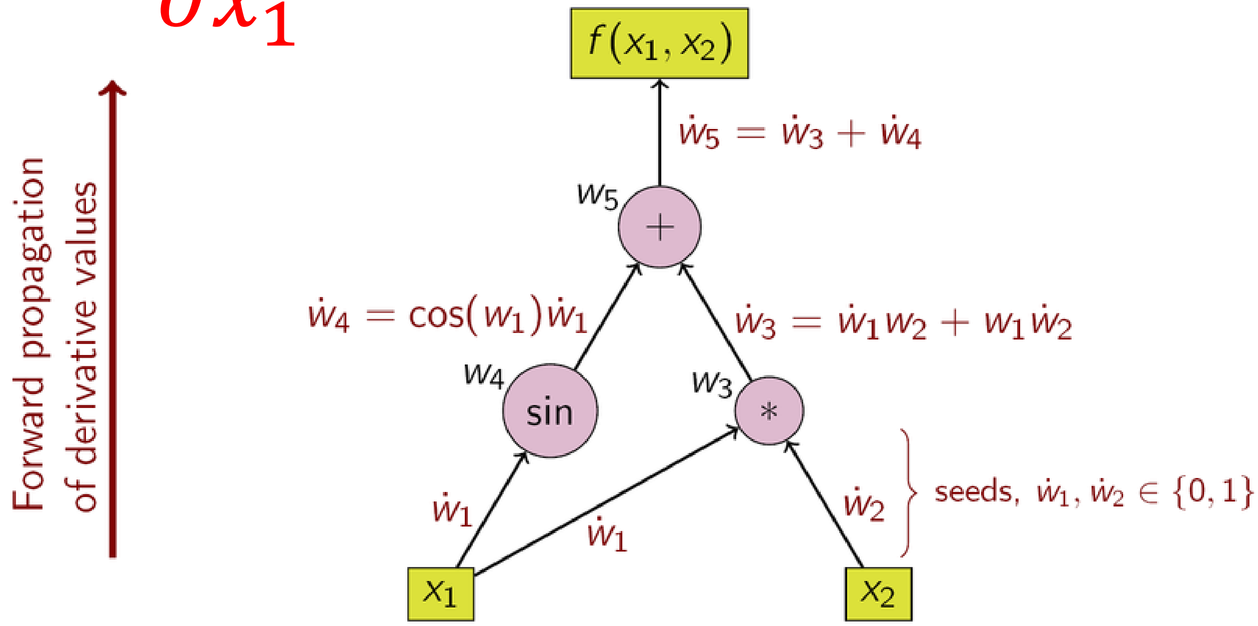
Diferenciación Automática

Assignation	Derivatives
$w_1 = x_1$	$w_1' = 1\ (seed)$
$w_2 = x_2$	$w_2' = 0\ (seed)$
$w_3 = w_1 w_2$	$w_3' = w_1' w_2 + w_1 w_2' = x_2$
$w_4 = \sin(w_1)$	$w_4' = \cos(w_1) w_1' = \cos(x_1)$
$w_5 = w_3 + w_4$	$w_5' = w_3' + w_4' = x_2 + \cos(x_1)$

Ejemplo::

$$f = x_1 x_2 + \sin(x_1)$$

$$¿\frac{\partial f}{\partial x_1}?$$



Optimal control

Classical optimal control theory has its roots in the calculus of Variations. Typical problems are formulated as:

$$\min_{u(t)} \quad J(\mathbf{u}) = \phi(\mathbf{x}(t_f)) + \int_{t_0}^{t_f} C(\mathbf{x}, \mathbf{u})dt$$

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t), \quad \mathbf{x}(t_0) = \mathbf{x}_0$$

$$t_0, t_f, \mathbf{x}_0 \quad \text{specified}$$

The Lagrange multipliers approach and variational principle can be used to solve the problem. As the constraints are dynamic, the Lagrange multiplier vector λ(t) is a function of time.

$$\min_{u(t),\lambda(t)} \quad \bar{J} = \phi(\mathbf{x}(t_f)) + \int_{t_0}^{t_f} \{C(\mathbf{x}, \mathbf{u}) + \lambda(t)'[\mathbf{f}(\mathbf{x}, \mathbf{u}, t) - \dot{\mathbf{x}}]\}dt$$

Hamiltonian

Using the **Hamiltonian**, defined as:

$$H(t) = C(\mathbf{x},\mathbf{u}) + \lambda(t)'\mathbf{f}(\mathbf{x},\mathbf{u},t)$$

$$\bar{J} = \phi(\mathbf{x}(t_f)) + \int_{t_0}^{t_f}\left\{C(\mathbf{x},\mathbf{u}) + \lambda(t)'\left[\mathbf{f}(\mathbf{x},\mathbf{u},t) - \dot{\mathbf{x}}\right]\right\}dt =$$

$$= \phi(\mathbf{x}(t_f)) + \int_{t_0}^{t_f}\left\{H(t) - \lambda(t)'\dot{\mathbf{x}}\right\}dt = (\text{integrating by parts } \lambda\dot{x})$$

$$= \phi(\mathbf{x}(t_f)) - \lambda(t_f)x(t_f) + \lambda(t_0)x(t_0) + \int_{t_0}^{t_f}\left\{H(t) + \dot{\lambda}(t)'\mathbf{x}\right\}dt$$

NOC

$$\bar{J} = \phi(\mathbf{x}(t_f)) - \lambda(t_f)' x(t_f) + \lambda(t_0)' x(t_0) + \int_{t_0}^{t_f} \left\{ H(t) + \dot{\lambda}(t)' \mathbf{x} \right\} dt$$

for optimality :

$$\left.\frac{\partial \phi}{\partial x}\frac{\partial \mathbf{x}}{\partial \mathbf{u}}\right|_{t_f} - \lambda(t_f)' \left.\frac{\partial \mathbf{x}}{\partial \mathbf{u}}\right|_{t_f} + \int_{t_0}^{t_f} \left\{ \frac{\partial H}{\partial x}\frac{\partial \mathbf{x}}{\partial \mathbf{u}} + \frac{\partial H}{\partial \mathbf{u}} + \dot{\lambda}(t)' \frac{\partial \mathbf{x}}{\partial \mathbf{u}} \right\} dt = 0$$

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t)$$

Now, by choising:

$$\dot{\lambda}(t)' = -\frac{\partial H}{\partial \mathbf{x}} \qquad \lambda(t_f)' = \left.\frac{\partial \phi}{\partial \mathbf{x}}\right|_{t_f} \qquad \frac{\partial H}{\partial \mathbf{u}} = 0 \qquad x(t_0) \text{ given}$$

The NOC are always satisfied

NOC

The NOC in terms of the Hamiltonian are given by:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x},\mathbf{u},t) \qquad \mathbf{x}(t_0) = \mathbf{x}_0 \qquad H = C + \boldsymbol{\lambda}'\mathbf{f}(\mathbf{x},\mathbf{u},t)$$

$$\dot{\lambda}(t)' = -\frac{\partial H}{\partial \mathbf{x}} \qquad \lambda(t_f)' = \left.\frac{\partial \phi}{\partial \mathbf{x}}\right|_{t_f} \qquad \frac{\partial H}{\partial \mathbf{u}} = 0$$

This is a TPBVP as part of the boundary conditions of the differential equations are given at t_f and part at t_0

Positive definite Hessian is required for sufficiency

Similar equations result for other formulations such as terminal constraints, free final time, etc.