Ottimizzazione dei Sistemi Complessi

G. Liuzzi[1]

Martedì 17 Maggio 2016

[1] Istituto di Analisi dei Sistemi ed Informatica IASI - CNR

Programmazione con incertezza

Una società di autonoleggio dispone (attualmente, oggi) di 50 macchine tutte dislocate nella città di New York. È noto che domani la società avrà richieste di autovetture per viaggi da New York (N) a Boston (B) e da Boston a New York. Supponiamo siano possibili due scenari e che le richieste con rispettive probabilità siano quelle riportate nella tabella che segue.

p(s)	d_{NB}	d_{BN}
1/3	20	30
2/3	30	10

Determinare un piano di allocazione tenendo presente che:

- spostare una vettura (vuota) oggi da N a B costa 0.5
- un cliente che viaggia da N a B (o da B a N) paga 1.5

AMPL

Scrivere nella sintassi di AMPL il seguente modello lineare

$$\begin{aligned} \min\ & 3x_A + 4x_B + 5x_C + 6x_D \\ s.t.\ & 1.1x_A + 0.7x_B + x_C + x_D \leq 7 \\ & x_A + x_D \leq 3 \\ & x_B + 3x_C \leq 5 \\ & x_A, x_B, x_C, x_D \geq 0 \end{aligned}$$

tenendo conto che:

1. l'utilizzo di una istruzione `set` da **1 punto**;
2. l'utilizzo di una istruzione `param` a 1 dimensione da **1 punto**;
3. l'utilizzo di una istruzione `param` a 2 dimension1 da **1 punto**;
4. l'utilizzo di una istruzione `sum` da **1 punto**;
5. la definizione di un vettore di vincoli da **1 punto**;
6. la definizione di un vettore di variabili da **1 punto**;

AMPL

```
param pi := 4*atan(1);
param beta;                                                    1
minimize B: pi*r^2;
s.t.  vs:  r^2 + h^2 = s^2;
s.t.  V: pi*r^2*h/3 >= 200;                                    2
s.t.  S: pi*r*s <= beta;
    printf "S =%lf B = %lf\n",pi*r*s,pi*r^2;
    printf "%lf %lf\n",pi*r*s,pi*r^2 > cone.out;               3
}
reset;
var r >= 2;                                                    4
var h >= 2;
var s >= 0;
printf "S = %lf B = %lf\n",pi*r*s,pi*r^2;                      5
printf "%lf %lf\n",pi*r*s,pi*r^2 > cone.out;
data;                                                          6
param beta := Infinity;
for{i in 1..1000}{
    let beta := 300  0.1*i;                                    7
    solve;
let r := 2;
let h := 2;
let s := sqrt(4);                                              8
option solver minos;
solve;
```

Julia

`@addNLConstraint(m,2.0*x[1]^2 + ... <= 0 )`
`@addNLConstraint(m,10.0*x[3]^2 + ... <= 0 )`
`@addNLConstraint(m,x[2]^2 + ... <= 0 )`
`@addNLConstraint(m,4.0*x[1]^2 + ... <= 0 )`
	1

`@defVar(m, x[1:n])`	2
`m = Model(solver=NLoptSolver()) n = 7`	3
`@setNLObjective(m, Min,(x[1]-10.0)^2 + ...)`	4

`fstar = JuMP.getObjectiveValue(m)`
`xstar = zeros(n)`
`for i = 1:n`
`  xstar[i] = getValue(x[i])`
`end`
	5

`xstart = Float64[1.0, 2.0, 0.0, 4.0, 0.0, 1.0, 1.0]`
`for i = 1:n`
`  setValue(x[i], xstart[i])`
`end`
	6

`using JuMP` `using NLopt`	7
`status = solve(m)`	8

Programmazione con incertezza

Una industria deve decidere in quali città (1,2,3) aprire magazzini per lo stoccaggio di un prodotto finito prima della vendita. L'apertura di ciascun magazzino comporta un costo unitario. I clienti sono posizionati in ulteriori tre città che indichiamo con A, B e C. I costi unitari di trasporto tra i magazzini e i clienti dipendono dalle origini e destinazioni e sono riportati in tabella

	1	2	3
A	0.1	0.2	0.3
B	0.2	0.1	0.3
C	0.3	0.2	0.1

Riguardo alle domande da soddisfare, sono prevedibili due scenari, come riportato in tabella

p(s)	d_A	d_B	d_C
1/3	100	200	150
2/3	110	190	160

Controllo Ottimo

$$
\begin{aligned}
\min \ & \frac{1}{2}x_3(T)^2 + \frac{1}{2}\int_0^T \left(x_1(t)^2 + x_2(t)^2 + u_1(t)^2 + u_2(t)^2\right) dt \\
& \dot{x_1}(t) = -x_1(t) - x_2(t) - x_3(t) + u_1(t) \\
& \dot{x_2}(t) = x_3(t) + u_2(t) \\
& \dot{x_3}(t) = x_2(t) \\
& x_1(0) = x_2(0) = x_3(0) = 1
\end{aligned}
$$

Controllo Ottimo – AMPL

Vediamo la formulazione AMPL mediante estensioni `TACO` (Toolkit for AMPL Control Optimization)

```
option presolve 0;

suffix scale IN >= 0;
suffix interp_to IN;
suffix type symbolic IN;
suffix slope_min IN;
suffix slope_max IN;

option type_table '\
  1    u0          piecewise constant control\
  2    u1          piecewise linear control\
  3    u1c         piecewise linear continuous control\
  4    u3          piecewise cubic control\                                    ';
  5    u3c         piecewise cubic continuous control\
  6    dae         DAE algebraic state variable\
  7    Lagrange    Force a least-squares objective to be treated as a Lagrange type one\
  8    dpc         Treat ODE initial value constraint as decoupled point constraint\

function diff;
function eval;
function integral;
```

Controllo Ottimo – AMPL

```
param T := 10;

var t;
var x1;
var x2;
var x3;

var w1 suffix type "u3c";
var w2 suffix type "u3c";

minimize obj2:  integral (x1^2+x2^2+w1^2+w2^2,T);
minimize obj1:  eval(x3^2/2,T);

subject to

ODE1:  diff (x1,t) = - x1 - x2 - x3 + w1;
ODE2:  diff (x2,t) = x3 + w2;
ODE3:  diff (x3,t) = x2;

init1:  eval(x1,0) = 1;
init2:  eval(x2,0) = 1;
init3:  eval(x3,0) = 1;
```

Controllo Ottimo – Soluzione con MUSCOD-II

All'indirizzo:

`https://neos-server.org/neos/solvers/index.html`

nella categoria "Mixed-Integer Optimal Control Problems" (MIOCP), è disponibile il solutore: `MUSCOD-II`

Il risolutore scrive un file `ampl.sol` in cui è possibile leggere:

1. valore ottimo di funzione obiettivo (1 valore)
2. errore di ammissibilità (1 valore)
3. tempo finale, nel nostro caso $T = 10$ (1 valore)
4. numero di istanti di tempo di integrazione n_i (1 valore)
5. istanti di tempo di integrazione (n_i valori)
6. traiettorie $x_i(t)$ ($n_i \times n_x$ valori)
7. traiettorie dei controlli $u_i(t)$ ($n_i \times n_u$ valori)

Controllo Ottimo (2)

$$
\begin{aligned}
\min \ & \frac{1}{2}\left(x_1(T)^2 + x_2(T)^2\right) \\
& \dot{x_1}(t) = x_1(t) + x_2(t) + u_1(t) \\
& \dot{x_2}(t) = x_2(t) + x_3(t) \\
& \dot{x_3}(t) = x_1(t) + x_2(t) + u_2(t) \\
& x_1(0) = x_2(0) = x_3(0) = 1 \\
& x_3(T) = 0
\end{aligned}
$$

Controllo Ottimo (2) – AMPL

```
param T := 10;

var t;
var x1;
var x2;
var x3;

var w1 suffix type "u3c";
var w2 suffix type "u3c";

minimize obj1:  eval(x1^2/2 + x2^2/2,T);

subject to

ODE1:  diff (x1,t) = x1 + x2 + w1;
ODE2:  diff (x2,t) = x2 + x3;
ODE3:  diff (x3,t) = x1 + x2 + w2;

init1:  eval(x1,0) = 1;
init2:  eval(x2,0) = 1;
init3:  eval(x3,0) = 1;
fini1:  eval(x3,T) = 0;
```