COM361 — Introdução a Otimização — 2022, Prof. Amit

Event triggered social media chatter model - Julia implementation

Gabriel José Souza e Silva (gabriel.jsss@poli.ufrj.br), Matheus Marinatto (marinattomatheus@poli.ufrj.br)

Contents

1. Introduction and problem description

This project's goal is to describe and explore a social marketing-based model for information spread, testing different control techniques so to determine which one provides better results. An important constraint for most marketing campaigns nowadays is the advertising cost, therefore a key goal for dispersing a message is to do that as efficiently and possible respecting budget and time constraints. From that, many optimization problems can be formulated. The following question, described in [1] gives the reason for such models to exist:

> At a high level, how does an organization sell to someone? Typically, individuals or groups (like marketing agencies) are enlisted as message spreaders who broadcast that information in a variety of formats including billboards, social media posts, and television advertisements.

So in order to sell, it's necessary to spread information and the further question would be **how to maximize sells (and information spread) while minimizing costs**.

In [1], the approach to design an advertising model starts from the well established Vidale-Wolfe marketing model with some tweaks to better handle the social media dynamics. Quoting [2]:

> The Vidale-Wolfe marketing model is a first-order, linear, nonhomogeneous ordinary differential equation (ODE) where the forcing term is proportional to advertising expenditure. With an initial response in sales as the initial

> condition, the solution of the initial value problem is straightforward for a first undergraduate ODE course.

Mathematically, this model is described as follows:

$\frac{dS(t)}{dt} = \beta u(t)[M(t) - S(t)] - \delta S(t)$

where

- S(t) → Sales at time t;
- M(t) → Market size at time t;
- β → Advertising constant;
- δ → Rate of brand sale decay;
- $u(t)$ → Control action at time t.

Most of the terms above are self-explanatory but β, which is described in [1] as "[...] the rate of decay of brand sale given no active advertising."

From that, the Event-triggered Social Media Chatter Model is derived by doing the following:

1. Normalizing Vidale-Wolfe's model by setting $M(t) = 1$;
2. Breaking β into two other constants:
   A. β_1 → the social marketing campaign constant;
   B. β_2 → the social interaction constant;
3. Generalizing the sales term $S(t)$ to an information spread value X_t.

In [1] the meaning of these constants is explained in the following paragraph

> The effectiveness of social marketing is affected by dynamic resource spending and promotion over the network to convince people to purchase a product, uphold a social or political movement, or join in an activity. The social marketing constant can be associated with a traditional advertising campaign or an event that triggers similar social media interest. Once people become exposed to an advertisement and decide to share the message, the social interaction constant may be viewed as the natural tendency of the social media network to advertise internally through posts, tweets, and likes without external influences and advertising.

The final mathematical model is

$$dX_t = \beta_1 u(t)[1 - X_t] + \beta_2[1 - X_t]X_t - \delta X_t$$

or visually

Model

The final definition to be presented is the concept of socio-equilibrium threshold. It basically describes the equilibrium level of social media chatter after the control (promotion) goes to 0 or mathematically

$$X_{eqb} = 1 - \frac{\delta}{\beta_2}$$

In [1], β_2 is multiplied by a factor k, but in this project the value of k will be embedded in β_2. Finally, the "[...] goal of social media marketing is to increase peoples' attention and interest beyond the natural equilibrium point through the control action of spending resources on ads..." and at the same time minimizing the associated costs.

2. Mathematical model

During a marketing campaign, **the goal is to achieve social craze status as fast as possible while keeping costs as low as possible**. From that the optimization problem will be derived. The cost function for the Event-triggered Social Media Chatter Model is defined in [1] as

$$J = \int_0^{t_f} [u^2(t) + (x - x_d)^2 + \lambda] dt$$

where

- λ → weight placed on time. Meaning the importance of how long it takes to get to equilibrium;
- x_d → desired amount of activity.

when there's no need to maintain x_d, $u(t) = 0$. Now the mentioned equations will be discretized, so they can be used in future simulations:

- Event-triggered Social Media Chatter Model:

$$x(t+1) - x(t) = \beta_1 u(t)[1 - x(t)] + \beta_2[1 - x(t)]x(t) - \delta x(t)$$

- Cost function:

$$J = \sum_{t=0}^{T_F} (u^2(t) + (x(t) - x_d(t))^2 + \lambda)$$

- Equilibrium point:

$$X_{eqb} = 1 - \frac{\delta}{\beta_2}$$

Gathering these equations, the optimization problem can be formulated as

$$
\begin{array}{lll}
\underset{u}{\text{minimize}} & J(u,x) & \\
\text{subjected to:} & u_i \geq 0 & i = 1, \ldots, m \\
& x_{i+1} = x_i + \beta_1 u_i[1-x_i] + \beta_2[1-x_i]x_i - \delta x_i & \\
& x_i \geq 0 &
\end{array}
$$

More constraints may be added according to the control technique used. The constants used are defined in the cell below, following the same as used in the snipet on page 151 of [1].

The first formulation proposed tries to get to the equibilibrium minimizing the associated expenses. However, another goal could be to get to the X_{eqb} as fast as possible. For this second goal proposed in this project, the objective function can be reduced to

$$
J = \sum_{t=0}^{T_F} (x(t) - X_{eqb})^2
$$

In [8]:

```
#dependencies

import Pkg;
Pkg.add("JuMP");
Pkg.add("Clp");
Pkg.add("PyPlot");
Pkg.add("Ipopt");

@time using Clp;
@time using JuMP;
@time using PyPlot;
@time using Ipopt;
```

```
   Resolving package versions...
  No Changes to `~/.julia/environments/v1.8/Project.toml`
  No Changes to `~/.julia/environments/v1.8/Manifest.toml`
   Resolving package versions...
  No Changes to `~/.julia/environments/v1.8/Project.toml`
  No Changes to `~/.julia/environments/v1.8/Manifest.toml`
   Resolving package versions...
  No Changes to `~/.julia/environments/v1.8/Project.toml`
  No Changes to `~/.julia/environments/v1.8/Manifest.toml`
   Resolving package versions...
  No Changes to `~/.julia/environments/v1.8/Project.toml`
  No Changes to `~/.julia/environments/v1.8/Manifest.toml`
  0.000369 seconds (180 allocations: 13.742 KiB)
  0.000285 seconds (180 allocations: 13.742 KiB)
  0.000284 seconds (180 allocations: 13.742 KiB)
  0.000296 seconds (180 allocations: 13.742 KiB)
```

In [57]:

```
δ = 0.1; #forgetting factor
β1 = 0.1; #spreading constant
β2 = 0.15; #social spreading
λ = 1; #time cost weight
xEquilibrium = 1-(δ/β2);
```

```
ρβ1 = 0.02;
ρβ2 = 0.03;
ρδ = 0.025;

function getTimeXeqReached(x)
    for t in 1:numberOfIterations
        if x[t] >= 0.33
            return t;
        end
    end
    return 0;
end;
```

3.1 Optimal control

In [115…

```
function optimalControl()
    m = Model(Ipopt.Optimizer);
    set_silent(m);
    numberOfIterations = 100;
    @variable(m, xOptimalControl[1:numberOfIterations] >= 0);
    @variable(m, uOptimalControl[1:numberOfIterations] >= 0);

    for t in 1:numberOfIterations-1
        @constraint(m, xOptimalControl[t+1] == xOptimalControl[t] + β1*uOpti
    end

    @constraint(m, sum(xOptimalControl) >= 0);
    @constraint(m, sum(uOptimalControl) >= 0);
    @constraint(m, xOptimalControl[1] == 0) #initial point


    @objective(m, Min, sum(
        uOptimalControl .^2 .+ (xOptimalControl .- xEquilibrium).^2 .+ λ
    ));
    optimize!(m);
    return JuMP.objective_value.(m), JuMP.value.(xOptimalControl), JuMP.valu
end;
```

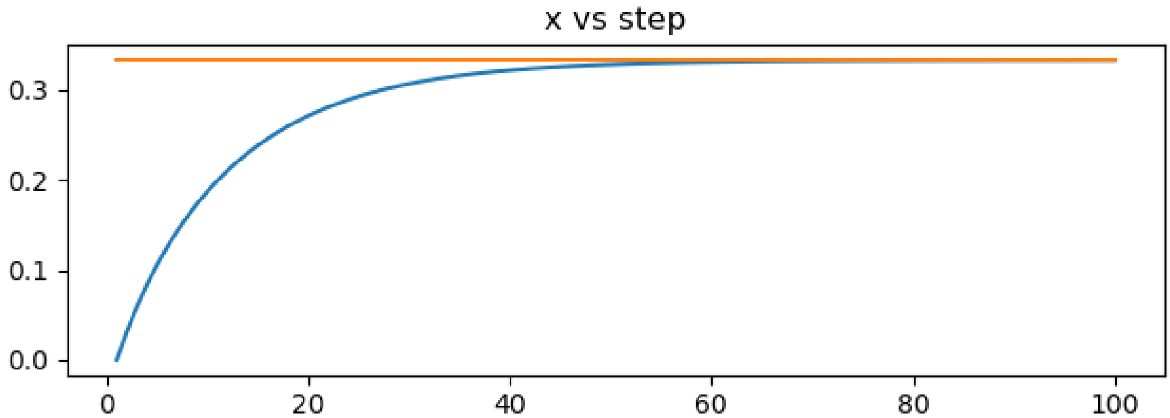
In [116…

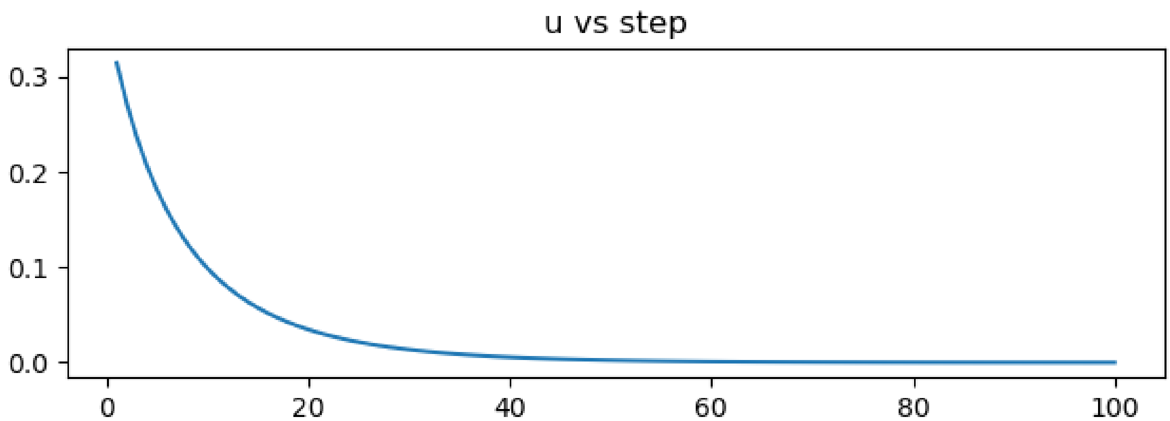
```
####### exporting variables for further analysis ########
costOptimalControlResult, xOptimalControlResult, uOptimalControlResult, cont
println(costOptimalControlResult);
println(xOptimalControlResult);
println(uOptimalControlResult);
println(controlAmountOptimalControlResult);

##### simple plot #######
clf(); #required for vscode on mac
fig = plt.figure();

ax1 = fig.add_subplot(2, 1, 1);
ax1.title.set_text("x vs step");
plt.plot(range(1, numberOfIterations), transpose(JuMP.value.(xOptimalControl
plt.plot(range(1, numberOfIterations), xEquilibrium .* ones(numberOfIteratic
```

```
ax2 = fig.add_subplot(2, 1, 2);
ax2.title.set_text("u vs step");
plt.plot(range(1, numberOfIterations), transpose(JuMP.value.(uOptimalControl

fig.tight_layout();
show()
gcf()
```

101.08874588996284
[-2.818722160738183e-40, 0.03143742192612366, 0.05916806739835122, 0.083804
67839066263, 0.10582232339851363, 0.12559605468638763, 0.1434268599765247,
0.15955993436529997, 0.1741977870907215, 0.18750979834242631, 0.19963928951
106827, 0.21070882249925718, 0.22082421915895611, 0.23007764372431538, 0.23
85499913558608, 0.24631275755364787, 0.25342951558751126, 0.259957095445632
5, 0.2659465337092212, 0.2714438463052314, 0.27649066331001065, 0.281124755
53555605, 0.28538047559871765, 0.2892891309004579, 0.29287930196352224, 0.2
9617711655897844, 0.299206487752443, 0.30198932224134356, 0.304545704003438
6, 0.3068940572357245, 0.30905129175803814, 0.31103293343145, 0.31285324165
57598, 0.31452531563096203, 0.3160611907699571, 0.3174719264153173, 0.31876
768582721826, 0.3199578092616985, 0.321050880839748, 0.3220547898118308, 0.
32297678674430996, 0.3238235350899966, 0.32460115855171345, 0.3253152846030
484, 0.32597108449259987, 0.32657331002560824, 0.3271263273888703, 0.327634
14826040815, 0.3281004584238811, 0.3285286440886742, 0.3289218160995826, 0.
3292828322047218, 0.32961431753648296, 0.32991868344781944, 0.3301981448347
3797, 0.3304547360654388, 0.3306903256269985, 0.3309066295917279, 0.3311052
2399727926, 0.3312875562271718, 0.33145495547157766, 0.33160864234192416,
0.3317497377070687, 0.3318792708134403, 0.33199818674657705, 0.332107353286
8584, 0.33220756720789524, 0.3322995600620653, 0.3323840034944981, 0.332461
51412538216, 0.33253265804196674, 0.3325979549462141, 0.33265788200930263,
0.3327128774847709, 0.3327633441219477, 0.3328096523977952, 0.3328521435526
21, 0.33289113238426543, 0.3329269097395174, 0.3329597446493035, 0.33298988
608490115, 0.33301756435583946, 0.3330429922114204, 0.3330663657341198, 0.3
3308786511864846, 0.33310765541719806, 0.33312588730658865, 0.3331426979052
617, 0.33315821164402876, 0.3331725411777297, 0.3331857883160465, 0.3331980
4494933275, 0.33320939394735327, 0.3332199100132267, 0.33322966048003705,
0.33323870604249545, 0.33324710142014846, 0.3332548959517609, 0.33326213412
26591, 0.3332688560281344]
[0.31437421926123654, 0.2716088944179608, 0.23599684367583193, 0.2060793616
9306237, 0.1807512104171038, 0.15916165490017414, 0.14064672882490362, 0.12
468184625921833, 0.11084799856633373, 0.09880722138316361, 0.08828451106491
081, 0.07905430713601916, 0.07093025872185214, 0.06375738704789127, 0.05740
6019262634535, 0.051767047609415306, 0.046748191326232265, 0.04227102500320
202, 0.03826859838338012, 0.034683516586180366, 0.03146638169316373, 0.0285
74520105391504, 0.025970937490137454, 0.023623456170561104, 0.0215039996608
35108, 0.019587996553227374, 0.017853881726681432, 0.01628267730578189, 0.0
14857639274096961, 0.013563958372157691, 0.012388506062645348, 0.0113196180
54655336, 0.010346909243964236, 0.009461115022096836, 0.008653954791088941,
0.007918014237387451, 0.007246643501589901, 0.006633868857421475, 0.0060743
15904432073, 0.005563142600892553, 0.005095980729375926, 0.0046688846079593
034, 0.004278286043227639, 0.003920954674013753, 0.0035939629824911553, 0.0
03294655356239091, 0.0030206206747965185, 0.002769667969930174, 0.002539804
7727501695, 0.0023292178148771187, 0.002136255796713999, 0.0019594139748392
49, 0.0017973203537207804, 0.0016487232952717562, 0.0015124803839921814, 0.
0013875484062032812, 0.0012729743197178777, 0.001167887105647732, 0.0010714
904073034874, 0.000983055872604377, 0.0009019171263287901, 0.00082746430709
00489, 0.0007591391112523693, 0.0006964302921811781, 0.0006388695682248521,
0.0005860278968476978, 0.0005375120771559336, 0.0004929616541148071, 0.0004
5204612358137506, 0.00041446248169687856, 0.00037993321087991075, 0.0003482
048092837361, 0.0003190469056707886, 0.0002922518421785282, 0.0002676343999
632642, 0.00024503118838665284, 0.00022429922158777324, 0.00020531340817932
613, 0.00018796302276775657, 0.0001721475776493584, 0.0001577727270426996,
0.00014474683697280306, 0.00013297866111043754, 0.00012237627102662212, 0.0
001128471118271814, 0.00010429887125070552, 9.664078709957261e-5, 8.9785053
20728821e-5, 8.364807522669118e-5, 7.815143269576349e-5, 7.32224949735377e-

```
5, 6.879470317730906e-5, 6.480756729999491e-5, 6.12064426071944e-5, 5.79421
4946457827e-5, 5.4970492500389894e-5, 5.225172330098284e-5, 4.9749978676063
8e-5, 4.743271544548287e-5, 4.52701532114146e-5]
2.776115114183729
```





3.2 Optimal control with max instantaneous and total control input

In [79]:
```
function optimalControlWithCostConstraints()
    m = Model(Ipopt.Optimizer);
    set_silent(m);
    numberOfIterations = 100;
    @variable(m, xOptimalControlLimitedCosts[1:numberOfIterations] >= 0);
    @variable(m, uOptimalControlLimitedCosts[1:numberOfIterations] >= 0);

    for t in 1:numberOfIterations-1
        @constraint(m, xOptimalControlLimitedCosts[t+1] == xOptimalControlLi
    end

    @constraint(m, uOptimalControlLimitedCosts[numberOfIterations] == 0);
    @constraint(m, sum(xOptimalControlLimitedCosts) >= 0);
    @constraint(m, sum(uOptimalControlLimitedCosts) >= 0);

    @constraint(m, xOptimalControlLimitedCosts[1] == 0) #prevent explosion
    @constraint(m, uOptimalControlLimitedCosts[1] <= 1) #prevent explosion

    ##### introduce max possible control input ###########################
    @constraint(m, sum(uOptimalControlLimitedCosts) <= 1.8)
```

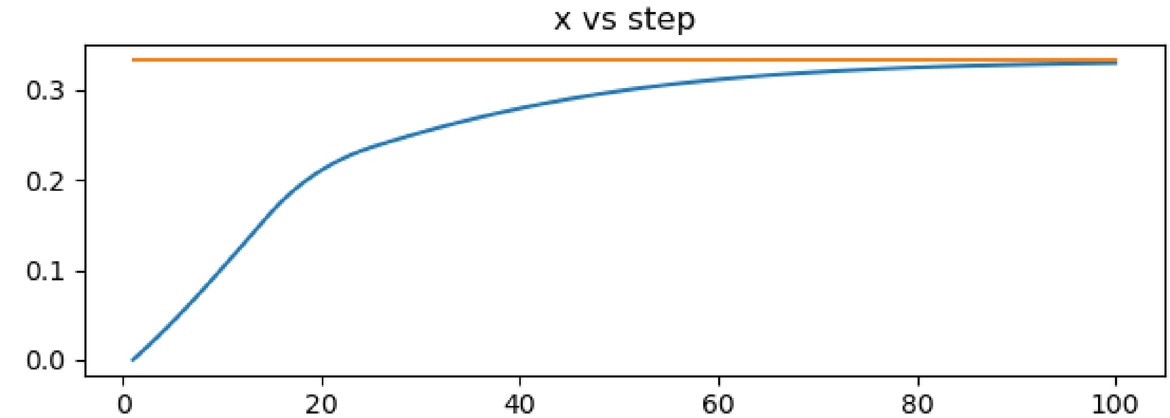
```julia
    @constraint(m, uOptimalControlLimitedCosts .<= 0.1)
    ##### introduce max possible control input ###########################

    advertisingCost = @expression(m, sum(uOptimalControlLimitedCosts .^2 .+
    @objective(m, Min, advertisingCost);
    optimize!(m);

    return JuMP.objective_value.(m), JuMP.value.(xOptimalControlLimitedCosts
end;
```

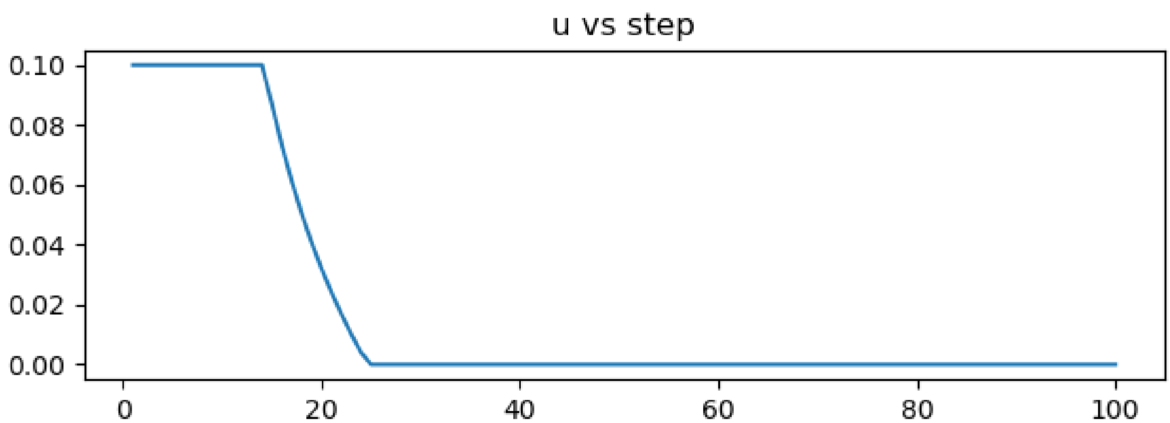
In [80]:

```julia
####### exporting variables for further analysis ########
costOptimalControlLimitedCostsResult, xOptimalControlLimitedCostsResult, uOp
println(costOptimalControlLimitedCostsResult);
println(xOptimalControlLimitedCostsResult);
println(uOptimalControlLimitedCostsResult);
println(controlAmountOptimalControlLimitedCostsResult);

##### simple plotting #######
clf(); #required for vscode on mac
fig = plt.figure();

ax1 = fig.add_subplot(2, 1, 1);
ax1.title.set_text("x vs step");
plt.plot(range(1, numberOfIterations), transpose(JuMP.value.(xOptimalControl
plt.plot(range(1, numberOfIterations), xEquilibrium .* ones(numberOfIteratio

ax2 = fig.add_subplot(2, 1, 2);
ax2.title.set_text("u vs step");
plt.plot(range(1, numberOfIterations), transpose(JuMP.value.(uOptimalControl

fig.tight_layout();
show()
gcf()
```

```
101.43442942075757
[5.807152217107358e-40, 0.010000000736685617, 0.020385001454448617, 0.03113
806990674576, 0.04223815636670494, 0.053660073840459586, 0.0653745666688119
7, 0.0773484745256171, 0.08954499566016587, 0.1019240494669187, 0.114442734
27181618, 0.12705587182708886, 0.13971662564281115, 0.1523771758383593, 0.1
649894028539156, 0.17642557730372865, 0.18661852043766727, 0.19570480918030
417, 0.20380072101018226, 0.21100578983753202, 0.21740556785558346, 0.22307
37923643937, 0.2280741017637171, 0.23246140622889294, 0.2362829964803067,
0.23972286603215928, 0.24308896373191924, 0.24637958211435992, 0.2495931311
2969472, 0.25272828154520005, 0.2557839607833515, 0.2587593459475254, 0.261
65385533658014, 0.2644671387920271, 0.26719906708730967, 0.2698497205366293
7, 0.27241937698782626, 0.2749084993546372, 0.27731772283462525, 0.27964784
19493319, 0.2818997975326613, 0.28407466378231044, 0.2861736354774327, 0.28
81980154538918, 0.2901492024166531, 0.2920286791572631, 0.2938380012331530
7, 0.29557878615481137, 0.2972527031168079, 0.2988614632993086, 0.300406810
7581427, 0.3018905139137166, 0.30331435764211867, 0.3046801359656253, 0.305
9896453344795, 0.3072446784872477, 0.3084470188732161, 0.3095984356171281,
0.31070067900403814, 0.3117554764601011, 0.31276452900368873, 0.31372950814
02572, 0.31465205317383704, 0.31553376890782237, 0.31637622370784946, 0.317
1809478999352, 0.3179494324776379, 0.31868312809278515, 0.3193834443052266,
0.3200517490681075, 0.32068936842626883, 0.3212975864065517, 0.321877645079
99417, 0.3224307447771289, 0.3229580444388174, 0.32346066208626767, 0.32393
967539507024, 0.32439612235924503, 0.3248310020324032, 0.32524527533420355,
0.32563986591129956, 0.3260156610429451, 0.32637351258234093, 0.32671423792
56637, 0.3270386210015273, 0.3273474132743766, 0.3276413347560119, 0.327921
07502009343, 0.32818729421506815, 0.32844062407151603, 0.3286816689004151,
0.3289110065792876, 0.329129189523611, 0.32933674564126075, 0.3295341792680
998, 0.3297219720831423, 0.3299005840020047, 0.33007045404760976, 0.3302320
011973377, 0.33038562520602205]
[0.10000000736685617, 0.10000000697480436, 0.10000000650014426, 0.100000005
91742692, 0.10000000518996652, 0.10000000426293484, 0.10000000305066503, 0.
100000001411585, 0.09999999909400334, 0.09999999560372358, 0.09999998982043
794, 0.0999999785496008, 0.09999994759851925, 0.09999953445676822, 0.087063
92218269232, 0.07334567163680976, 0.061217690234716994, 0.0504258544236426
4, 0.04075917856352338, 0.032040856497429784, 0.024121347876163268, 0.01687
2996326004794, 0.010185806177634028, 0.003964175177268788, 2.19856939226203
34e-6, 1.614272141027146e-7, 9.035509041866083e-8, 6.212726295857176e-8, 4.
697562600129977e-8, 3.7533927584887014e-8, 3.109451563552387e-8, 2.64280161
41498037e-8, 2.2895596288169472e-8, 2.0132381091953993e-8, 1.79148132758834
73e-8, 1.6098274049693217e-8, 1.4585065782390219e-8, 1.3306809031098621e-8,
1.2214211330596761e-8, 1.1270841042630829e-8, 1.0449189179535074e-8, 9.7280
9489450926e-9, 9.091013797197666e-9, 8.524823840288226e-9, 8.01898365800567
5e-9, 7.564927676350705e-9, 7.155623763606119e-9, 6.785244646899256e-9, 6.4
48920359886965e-9, 6.142549326410349e-9, 5.862652490101202e-9, 5.6062594638
530426e-9, 5.3708187869732966e-9, 5.154126536137532e-9, 4.954269054064997e-
9, 4.7695766415835785e-9, 4.598585839336657e-9, 4.4400084951707796e-9, 4.29
27062336395816e-9, 4.155669257354029e-9, 4.02799864558454e-9, 3.90889149437
1477e-9, 3.7976283792532235e-9, 3.6935627272552065e-9, 3.596111766764938e-
9, 3.5047487880466917e-9, 3.4189964976429972e-9, 3.338421289915574e-9, 3.26
262829086196e-9, 3.1912570548978504e-9, 3.1239778158853365e-9, 3.0604882103
62531e-9, 3.0005104045009707e-9, 2.9437885674125276e-9, 2.8900866425397173e
-9, 2.8391863763782666e-9, 2.7908855700040925e-9, 2.7449965240504724e-9, 2.
701344652098532e-9, 2.6597672410598127e-9, 2.620112340168444e-9, 2.58223776
27626872e-9, 2.5460101872034167e-9, 2.5113043451165663e-9, 2.47800228671260
23e-9, 2.4459927142729194e-9, 2.4151703760379443e-9, 2.3854355137145956e-9,
2.356693357667406e-9, 2.3288536645884813e-9, 2.3018302930745926e-9, 2.27554
```

```
08130893333e-9, 2.2499061457671714e-9, 2.224850230434318e-9, 2.200299716087
8976e-9, 2.176183674897004e-9, 2.1524333355733357e-9, 2.128981834710744e-9,
2.105763984416521e-9, -4.948958256977865e-43]
1.8000000059172825
```





3.3 Optimal control without cost concerns

In the problems formulated above, a maximum advertising time has been established and the goal is to get to the X_{eqb} using the minum amount of resources (control input). However, another problem could be formulated with the goal to reach X_{eqb} and fast as possible regarless of the cost.

In [82]:

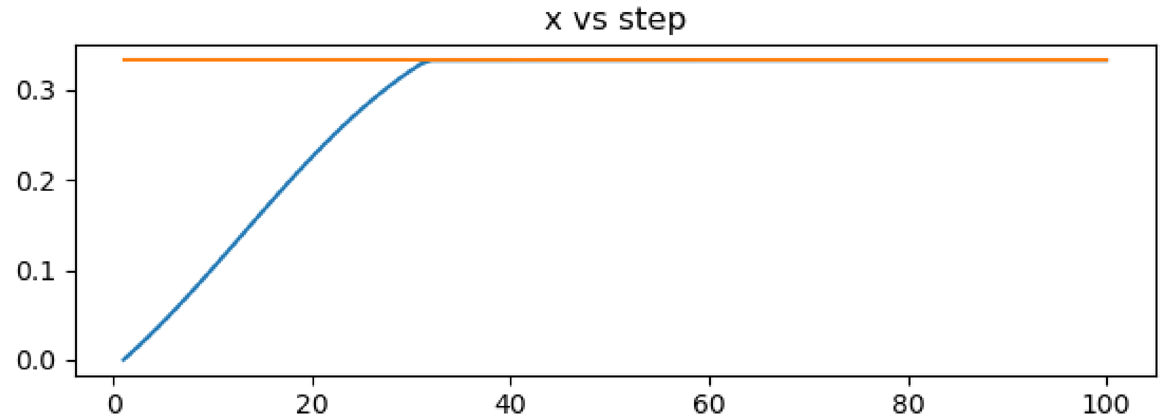
```
function fastestEquilibriumControl()
    m = Model(Ipopt.Optimizer);
    set_silent(m);
    numberOfIterations = 100;
    @variable(m, xFastestEquilibrium[1:numberOfIterations] >= 0);
    @variable(m, uFastestEquilibrium[1:numberOfIterations] >= 0);

    for t in 1:numberOfIterations-1
        @constraint(m, xFastestEquilibrium[t+1] == xFastestEquilibrium[t] +
    end

    @constraint(m, uFastestEquilibrium[numberOfIterations] == 0);
    @constraint(m, xFastestEquilibrium[numberOfIterations] == xEquilibrium);
    @constraint(m, sum(xFastestEquilibrium) >= 0);
    @constraint(m, sum(uFastestEquilibrium) >= 0);
```

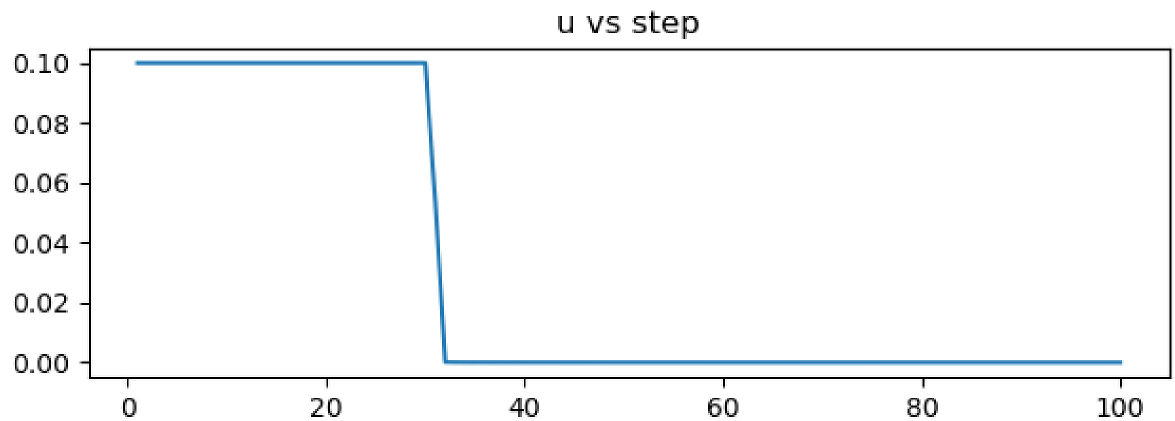
```
    @constraint(m, xFastestEquilibrium[1] == 0) #prevent explosion
    @constraint(m, uFastestEquilibrium .<= 0.1) #prevent explosion

    @objective(m, Min, sum(
        sum((xFastestEquilibrium .- xEquilibrium).^ 2)
    ));
    optimize!(m);

    return JuMP.objective_value.(m), JuMP.value.(xFastestEquilibrium), JuMP.
end;
```

In [83]:
```
####### exporting variables for further analysis ########
costFastestEquilibriumResult, xFastestEquilibriumResult, uFastestEquilibrium
println(costFastestEquilibriumResult);
println(xFastestEquilibriumResult);
println(uFastestEquilibriumResult);
println(controlAmountFastestEquilibriumResult);

##### plotting #######
clf(); #required for vscode on mac
fig = plt.figure();

ax1 = fig.add_subplot(2, 1, 1);
ax1.title.set_text("x vs step");
plt.plot(range(1, numberOfIterations), transpose(JuMP.value.(xFastestEquilib
plt.plot(range(1, numberOfIterations), xEquilibrium .* ones(numberOfIteratio

ax2 = fig.add_subplot(2, 1, 2);
ax2.title.set_text("u vs step");
plt.plot(range(1, numberOfIterations), transpose(JuMP.value.(uFastestEquilib

fig.tight_layout();
show()
gcf()
```

```
1.1287786900059669
[3.7489278434030316e-42, 0.010000000914057257, 0.020385001842951563, 0.0311
38070546259118, 0.04223815730502341, 0.0536600751359588, 0.0653745683945469
7, 0.07734847677589778, 0.08954499856168169, 0.10192405319891759, 0.1144427
3910638592, 0.12705587822168138, 0.1397166345067561, 0.15237718972580927,
0.1649894565952891, 0.17750580714480457, 0.18987979301832525, 0.20206683459
710909, 0.214024857246354, 0.2257148555779517, 0.23710137018720223, 0.24815
286559186028, 0.258842002760866, 0.2691458043372428, 0.2790457151226628, 0.
28852756433172444, 0.29758143933047604, 0.30620148286998594, 0.314385626886
29595, 0.3221352736915778, 0.32945489791439153, 0.33324242964152984, 0.3332
5435310645074, 0.3332622199922162, 0.33326846995020915, 0.3332737764543353,
0.33327843027129483, 0.333282587961871, 0.33328634607088775, 0.333289769894
4788, 0.333292906642185, 0.3332957921990959, 0.33329845491343996, 0.3333009
17865674, 0.3333032003034099, 0.3333053185902248, 0.33330728685703576, 0.33
33091174638717, 0.3333108213364755, 0.33331240821772934, 0.3333138868595585
4, 0.33331526517225374, 0.3333165503426856, 0.333317748929362, 0.3333188669
39951, 0.3333199098953184, 0.3333208828830478, 0.33332179060264894, 0.33332
263740411816, 0.33332342732112363, 0.33332416409979754, 0.3333248512239042,
0.33332549193699085, 0.33332608926200585, 0.33332664601877326, 0.3333271648
3963964, 0.33332764818355254, 0.33332809834878213, 0.33332851748446446, 0.3
333289076011122, 0.33332927058021816, 0.333329608183056, 0.3333299220587674
4, 0.33333021375181204, 0.33333048470884585, 0.3333307362850846, 0.33333096
97502015, 0.33333118629380265, 0.3333313870305173, 0.33333157300473604, 0.3
333317451950272, 0.3333319045182568, 0.3333320518334358, 0.333332187945314
6, 0.3333323136077449, 0.333332429526824, 0.33333253636383714, 0.3333326347
380116, 0.3333327252290936, 0.3333328083797587, 0.3333328846978658, 0.33333
295465856405, 0.3333330187062664, 0.3333330772565168, 0.33333313069781406,
0.3333331793935617, 0.33333322368458934, 0.3333332638934456, 0.333333300333
75397, 0.33333333333333326]
[0.10000000914057258, 0.10000000904115343, 0.10000000892809999, 0.100000008
79914342, 0.100000086515563, 0.10000000848204033, 0.10000000828658082, 0.1
0000000806025608, 0.10000000779698497, 0.10000000748918848, 0.1000000071273
3005, 0.10000000669928201, 0.10000000618943886, 0.10000000557745438, 0.1000
0000483640893, 0.10000000393009253, 0.10000000280888151, 0.1000000014033115
7, 0.0999999996137522, 0.09999999729322995, 0.0999999421767532, 0.09999999
003187027, 0.09999998414549331, 0.09999997551880245, 0.0999999621807184
7,
0.09999994001681634, 0.09999989922518243, 0.0999981144778697, 0.0999995637
303755, 0.0999982807567566, 0.05362601647758054, 0.00011067618723298284, 5.
877252214945719e-5, 4.041878234287372e-5, 3.095406201144896e-5, 2.514318237
7802584e-5, 2.1191212809619083e-5, 1.8315358483364304e-5, 1.611930311474912
4e-5, 1.4380718873742571e-5, 1.2965142830526683e-5, 1.178643190967346e-5,
1.0786826732166343e-5, 9.926141249922707e-6, 9.175547287207137e-6, 8.513824
627172805e-6, 7.925005490387722e-6, 7.396843285967514e-6, 6.919787377086979
e-6, 6.486278795757438e-6, 6.090255415695814e-6, 5.726797301629509e-6, 5.39
1867987347675e-6, 5.082122725440028e-6, 4.7947643394093445e-6, 4.5274334639
94498e-6, 4.2781239960185364e-6, 4.045117276804987e-6, 3.82693036363288e-6,
3.622275017722289e-6, 3.4300249277150242e-6, 3.2491893220327045e-6, 3.07889
15807609352e-6, 2.9183517912156876e-6, 2.766872437279839e-6, 2.623826595828
2146e-6, 2.4886481513880724e-6, 2.3608236448004022e-6, 2.2398854517340966e-
6, 2.1254060487102574e-6, 2.0169931723597837e-6, 1.9142857152979735e-6, 1.8
169502317181392e-6, 1.7246779494184122e-6, 1.6371822038536202e-6, 1.5541962
249809596e-6, 1.4754712199235918e-6, 1.400774704401899e-6, 1.32988904392956
25e-6, 1.2626101722868664e-6, 1.198746460034658e-6, 1.1381177100269382e-6,
1.0805542601827564e-6, 1.025896176320564e-6, 9.739925197494937e-7, 9.247006
756537763e-7, 8.778857292356062e-7, 8.334198774156694e-7, 7.911818655675755
e-7, 7.510564440749896e-7, 7.129338563870178e-7, 6.767094203843465e-7, 6.42
```

```
283405897246e-7, 6.095617942100757e-7, 5.784595250166943e-7, 5.489105402177
94e-7, 5.20896251502479e-7, 4.945246858616271e-7, 4.7024399229581973e-7, -
1.2641595362669033e-44]
3.0541570417764654
```





3.4 Model predective control

In [84]:

```
function MPC()
    t = collect(2:100) #eixo x
    loopCount = 100
    uMPC = Array{Float64}(undef, loopCount)
    xMPC = Array{Float64}(undef, loopCount)
    uMPC[1] = 0
    xMPC[1] = 0

    for i in t
        ################ optimization routine #####################
        m = Model(Ipopt.Optimizer);
        set_silent(m);
        movingTimeHorizon = 10; #prediction window
        @variable(m, xMPCLocal[1:movingTimeHorizon] >= xMPC[i-1]);
        @variable(m, uMPCLocal[1:movingTimeHorizon] >= 0);
        @constraint(m, xMPCLocal[1] == xMPC[i]) #initial point
        @constraint(m, xMPCLocal[10] == xEquilibrium) #initial point
        @constraint(m, uMPCLocal[1] == uMPC[i]) #initial point
        @constraint(m, uMPCLocal .<= 0.1)
        for t in 1:movingTimeHorizon-1
            @constraint(m, xMPCLocal[t+1] == xMPCLocal[t] + β1*uMPCLocal[t]*
        end
```

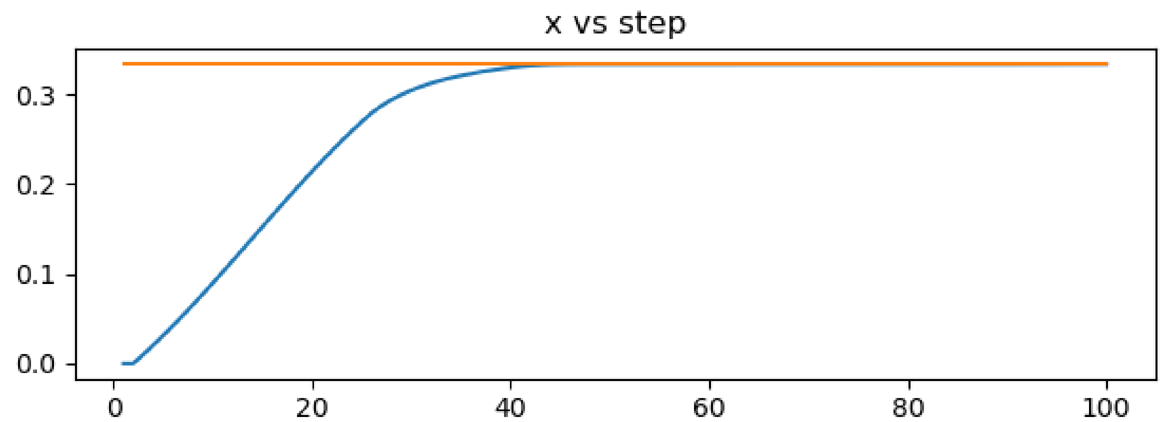
```
        @objective(m, Min, sum(
            uMPCLocal .* uMPCLocal .+ (xMPCLocal .- xEquilibrium).^2 .+ λ
        ));
        optimize!(m);
        ############################################################
        uMPC[i] = JuMP.value.(uMPCLocal)[2]
        xMPC[i] = xMPC[i - 1] + β1*uMPC[i - 1]*(1 - xMPC[i - 1]) + β2*(1 - x
    end

    return JuMP.objective_value.(m), xMPC, uMPC, sum(uMPCResult);
end;
```

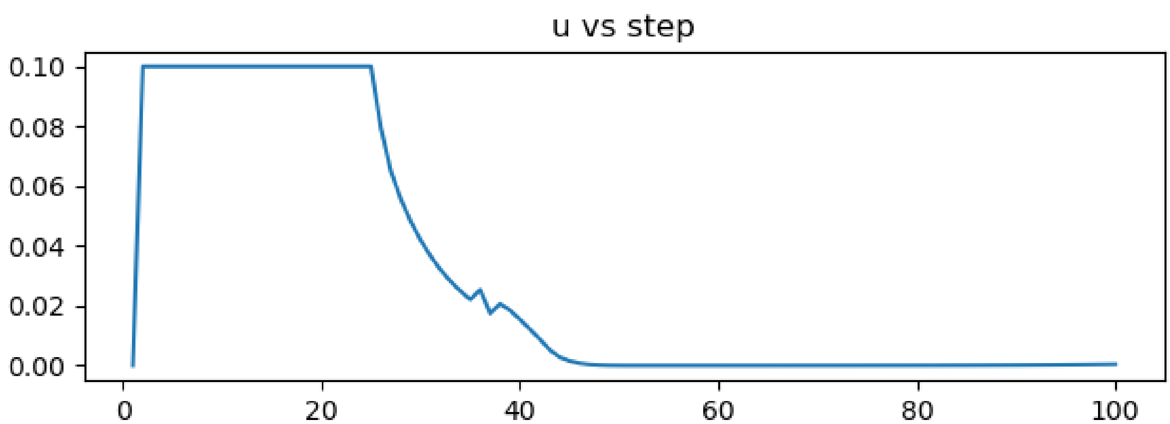
In [85]:

```
costMPCResult,xMPCResult,uMPCResult,controlAmountMPCResult = MPC();
println(costMPCResult);
println(xMPCResult);
println(uMPCResult);
println(controlAmountMPCResult);

clf(); #required for vscode on mac
fig = plt.figure();

ax1 = fig.add_subplot(2, 1, 1);
ax1.title.set_text("x vs step");
plt.plot(range(1, loopCount), transpose(xMPC));
plt.plot(range(1, loopCount), xEquilibrium .* ones(loopCount));

ax2 = fig.add_subplot(2, 1, 2);
ax2.title.set_text("u vs step");
plt.plot(range(1, loopCount), transpose(uMPC));

fig.tight_layout();
show()
gcf()
```

```
101.43442942075757
[0.0, 0.0, 0.01000000099868024, 0.020385002024324844, 0.031138070837779364,
0.04223815771891406, 0.05366007568828731, 0.06537456910241679, 0.0773484776
5687619, 0.08954499963484423, 0.10192405448446289, 0.11444274062747162, 0.1
2705588000295542, 0.13971663657856248, 0.15237719211235365, 0.1649894593420
5317, 0.17750581029561022, 0.18987979662555782, 0.20206683872462228, 0.2140
2486197359472, 0.2257148610059425, 0.2371013764474497, 0.24815287286005241,
0.25884201127876816, 0.26914581444939667, 0.2790457273416028, 0.28704237655
88757, 0.293668625925108, 0.2993453644162042, 0.3042463857975898, 0.3084889
8064792496, 0.31216827491577903, 0.3153616227065969, 0.318142995534 6933, 0.
3205669327291344, 0.32268164825446793, 0.3249151693672486, 0.32650265986862
52, 0.328228524658 1623, 0.3297219187201048, 0.330934360328611, 0.3318678947
805426, 0.33253275427947676, 0.33292330237156453, 0.33312827521826155, 0.33
323950288999116, 0.33329399519732783, 0.3333187023551317, 0.333328928769428
6, 0.3333326939935829, 0.3333337960869559, 0.33333393174665504, 0.333333991
7595427, 0.3333340752516587, 0.3333341682634692, 0.3333342729466131, 0.3333
343907412268, 0.3333345232865955, 0.33333467242682674, 0.3333348402397605,
0.3333350290690523, 0.33333524166873735, 0.33333548077146835, 0.33333574985
210196, 0.333336052648191, 0.33333639338786947, 0.33333677682487584, 0.3333
37208308656, 0.33333769385178597, 0.3333382402506079, 0.33333885512004147,
0.3333395470412498, 0.3333403256701751, 0.3333412021130066, 0.3333421884999
8676, 0.33334329823502246, 0.33334454369327404, 0.3333459477848078, 0.33334
752687503816, 0.33334930337392166, 0.3333513018764012, 0.33335355039295195,
0.33335608135425426, 0.33335893258734833, 0.33336214068847886, 0.3333657510
21602, 0.33336981392298337, 0.33337438617383736, 0.333379531661252, 0.33338
5322297521, 0.33339183902503355, 0.3333991729537022, 0.33340742664145606,
0.33341672862401717, 0.3334271944791847, 0.33343897290777563, 0.33345222809
54399, 0.3334671455011394, 0.3334839338525353, 0.3335025487426157]
[0.0, 0.10000000998680239, 0.10000000998680239, 0.10000000998975404, 0.1000
000099699274, 0.10000000997888855, 0.10000000998614632, 0.1000000099843392
8, 0.10000000998284196, 0.10000000997485345, 0.10000000997702911, 0.1000000
0996811832, 0.10000000998904311, 0.10000000989409542, 0.100000000998 80766,
0.10000000998756683, 0.10000000998704113, 0.10000000998650108, 0.1000000099
8594846, 0.10000000998538536, 0.10000000998481388, 0.10000000998423642, 0.1
0000000998365537, 0.10000000998307321, 0.10000000998249246, 0.0793994953553
1331, 0.06498463971829868, 0.05563247813957788, 0.0481678143381417, 0.04189
9244094217435, 0.03658167175712501, 0.0320178430900 8523, 0.0282080981469670
7, 0.024917720984724296, 0.022089632004667414, 0.025364078998641848, 0.0174
37988121477357, 0.020658282496373328, 0.018489360363111298, 0.0154238554720
00385, 0.01217293213289298, 0.008859171609827306, 0.005252921108355742, 0.0
027657466985860535, 0.0015142503355 30465, 0.0007469264602241208, 0.00034108
712030690437, 0.00014242033682523563, 5.3174631719658446e-5, 1.605188130830
852e-5, 2.381962839490817e-6, 1.349005322588036e-6, 1.7462040974569055e-6,
1.9516193122472187e-6, 2.196449080459465e-6, 2.471634634799612e-6, 2.781243
3776735537e-6, 3.12957718624131e-6, 3.5215252344226224e-6, 3.96263326362994
95e-6, 4.4608148817760285e-6, 5.017815075583173e-6, 5.646816670682363e-6,
6.35436658328814e-6, 7.150627125542681e-6, 8.04665400150099e-6, 9.054948810
1008e-6, 1.0189471452567304e-5, 1.146648894952125e-5, 1.2903378608410682e-
5, 1.452034702669946e-5, 1.6339953986809885e-5, 1.83911980092999e-5, 2.0697
77307182111e-5, 2.3287886283868658e-5, 2.6156164429525153e-5, 2.94699212771
9697e-5, 3.3148177309828016e-5, 3.7293886806430064e-5, 4.195664655320906e-
5, 4.7206154354406665e-5, 5.3129745080596103e-5, 5.9832718028758994e-5, 6.7
32501714848811e-5, 7.576565431119757e-5, 8.526329752029411e-5, 9.5952450025
7486e-5, 0.00010798238307859159, 0.00012152151325235132, 0.0001367593822148
2288, 0.00015390940723044758, 0.00017321189131776256, 0.0001951337587582776
2, 0.0002195774113509388, 0.0002471269039820237, 0.0002781266769409964, 0.0
```

```
0031301978761256815, 0.00035229539672581363, 0.00039231339567123567, 0.0004
409920078887757]
2.9671206150474885
```





3.5 Uncertanties

In [138…

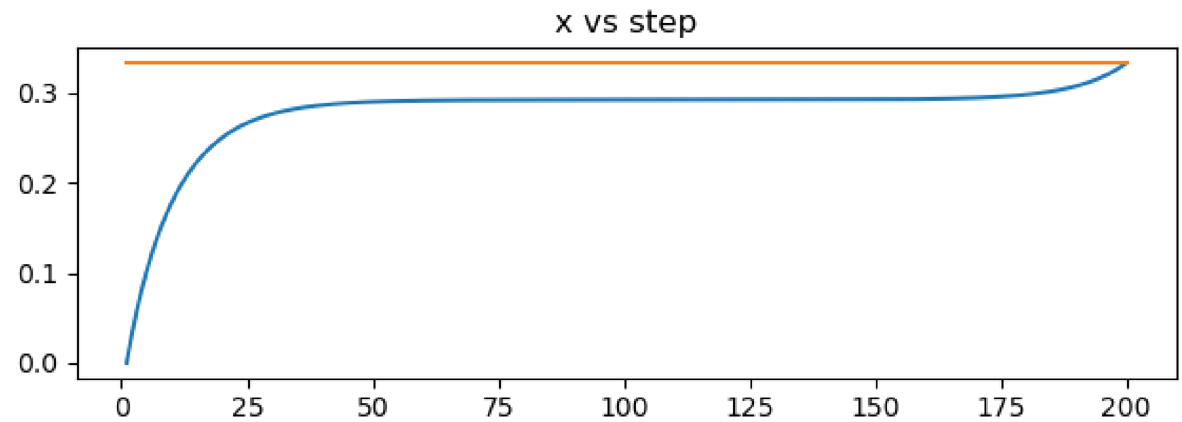
```
function optimalControlWithUncertanties()
    m = Model(Ipopt.Optimizer);
    set_silent(m);
    numberOfIterations = 200;
    @variable(m, xOptimalControl[1:numberOfIterations] >= 0);
    @variable(m, uOptimalControl[1:numberOfIterations] >= 0);
    @variable(m, t1[1:numberOfIterations] >= 0);
    @variable(m, t2[1:numberOfIterations] >= 0);
    @variable(m, t3[1:numberOfIterations] >= 0);

    for t in 1:numberOfIterations-1
        @constraint(
            m,
            xOptimalControl[t+1] == xOptimalControl[t]
            + β1*uOptimalControl[t]*(1 - xOptimalControl[t]) - ρβ1*(β1*uOpti
            + β2*(1 - xOptimalControl[t])*xOptimalControl[t] - ρβ2*(1 - xOpt
            - (δ*xOptimalControl[t] - ρδ*(δ*xOptimalControl[t]))
        );
    end

    @constraint(m, sum(xOptimalControl) >= 0);
    @constraint(m, sum(uOptimalControl) >= 0);
    @constraint(m, xOptimalControl[1] == 0) #initial point
```

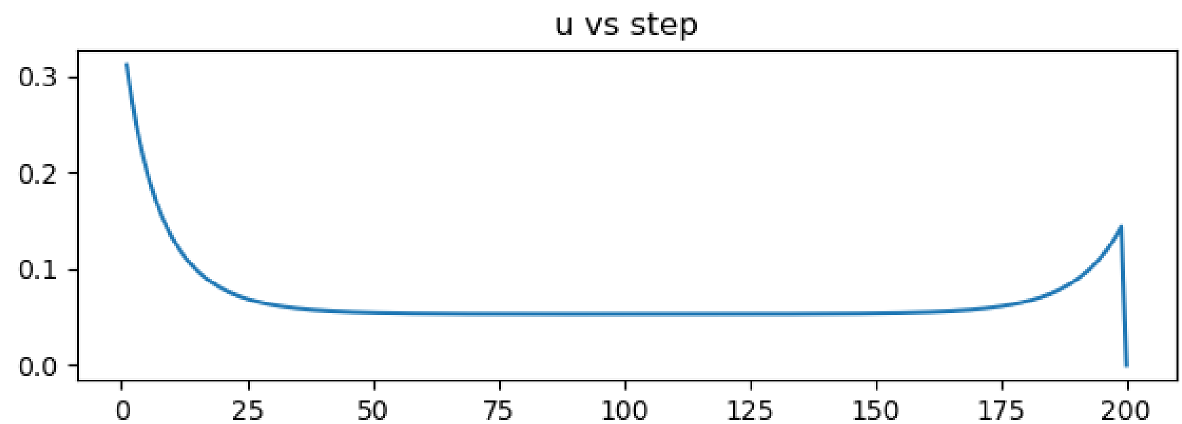
```
    @constraint(m, xOptimalControl[200] == xEquilibrium) #final point

    @objective(m, Min, sum(
        uOptimalControl .^2 .+ (xOptimalControl .- xEquilibrium).^2 .+ λ
    ));
    optimize!(m);
    return JuMP.objective_value.(m), JuMP.value.(xOptimalControl), JuMP.valu
end;
```

In [139…

```
####### exporting variables for further analysis ########
costOptimalControlWithUncertantiesResult, xOptimalControlWithUncertantiesRes
println(controlAmountOptimalControlWithUncertantiesResult)
println(xOptimalControlWithUncertantiesResult)

##### simple plot #######
clf(); #required for vscode on mac
fig = plt.figure();

ax1 = fig.add_subplot(2, 1, 1);
ax1.title.set_text("x vs step");
plt.plot(range(1, 200), transpose(JuMP.value.(xOptimalControlWithUncertantie
plt.plot(range(1, 200), xEquilibrium .* ones(200));

ax2 = fig.add_subplot(2, 1, 2);
ax2.title.set_text("u vs step");
plt.plot(range(1, 200), transpose(JuMP.value.(uOptimalControlWithUncertantie

fig.tight_layout();
show()
gcf()
```

```
13.736652085184138
[8.299344638756079e-41, 0.030558169989827313, 0.057434585427295896, 0.08119
50749639536, 0.10228992215834218, 0.12108346899053188, 0.13787491412453176,
0.15291321308924685, 0.16640794266587414, 0.17853735148884037, 0.1894544158
4846853, 0.19929145990209493, 0.20816372847159653, 0.21617218587022546, 0.2
2340573593217292, 0.2299430042227449, 0.23585378537846755, 0.24120023152281
406, 0.24603783832218515, 0.2504162712012395, 0.2543800639672196, 0.257969
21452243243, 0.2612196967197222, 0.2641639032064185, 0.26683103092855565,
0.2692474185575, 0.27143684325786077, 0.2734207827952834, 0.275218647879922
5, 0.27684798877834343, 0.27832467954566825, 0.2796630826880036, 0.28087619
66304094, 0.2819757880137002, 0.28297251055592165, 0.28387601197753737, 0.2
846950302925904, 0.28543748060323587, 0.28611053339580594, 0.28672068521809
085, 0.2872738225159972, 0.2877752793201876, 0.28822988939734256, 0.2886420
334144341, 0.2890156816063315, 0.28935443238595776, 0.289661547291061, 0.28
99399826216455, 0.2901924180865239, 0.29042128274574264, 0.2906287785073069
5, 0.2908169014112792, 0.29098746091160116, 0.29114209734558755, 0.29128229
776269987, 0.2914094102677088, 0.2915246570184885, 0.291629146005291, 0.291
7238817262601, 0.29180977486303966, 0.29188765105047537, 0.291958258825513
4, 0.2920222768323497, 0.29208032035361225, 0.2921329472307793, 0.292180663
23108227, 0.29222392691275845, 0.29226315403564107, 0.29229872155966247, 0.
292309712698492, 0.2923602130627706, 0.2923867279261242, 0.292410770640173
07, 0.29243257222706437, 0.2924523421716186, 0.29247027043497575, 0.2924865
2928048513, 0.2925012749294129, 0.2925146490624025, 0.29252678018113454, 0.
2925377848432874, 0.292547768782677, 0.29255682792535115, 0.292565049311409
3, 0.29257251193141326, 0.29257928748543083, 0.2925854410720108, 0.29259103
181371265, 0.2925961134252054, 0.2926007347293957, 0.29260494012654636, 0.2
926087700208962, 0.29261226120887845, 0.29261544723267263, 0.29261835870248
454, 0.29262102359065373, 0.2926234675004139, 0.2926257139118894, 0.2926277
8440769127, 0.2926296988802815, 0.292631475723097, 0.2926331320072707, 0.29
263468364564765, 0.2926361455456721, 0.29263753175261387, 0.292638855584511
8, 0.2926401297601292, 0.2926413665211517, 0.29264257774980146, 0.292643775
0829976, 0.2926449700241599, 0.29264617405372984, 0.29264739873947104, 0.29
264865584760913, 0.29264995745587796, 0.2926513160695601, 0.292652744741634
2, 0.2926542571981865, 0.29265586797028864, 0.29265759253361145, 0.29265944
745711475, 0.2926614505622428, 0.2926636210941545, 0.29266597990663473, 0.2
926685496624633, 0.2926713550511671, 0.29267442302624885, 0.292677783064172
1, 0.2926814674475915, 0.2926855115755522, 0.2926899543036386, 0.2926948383
173431, 0.29270021054224576, 0.29270612259494616, 0.2927126312790859, 0.292
7197991312304, 0.29272769502186047, 0.2927363948172538, 0.2927459821086248
5, 0.2927565490155364, 0.292768197071315, 0.2927810381989909, 0.29279519578
71564, 0.29281080587610164, 0.2928280184656489, 0.292846998957283, 0.292867
9297444733, 0.2928910119665118, 0.2929164674427787, 0.29294454080608706, 0.
2929755018556913, 0.2930096481526709, 0.29304730788275307, 0.29308884301423
777, 0.293134652781555, 0.29318517752815854, 0.29324090294596294, 0.2933023
647524045, 0.29337015385048876, 0.29344492202192457, 0.29352738820868124,
0.2936183454441058, 0.29371866850115164, 0.29382932233238035, 0.29395137138
42717, 0.29408598987710804, 0.29423447315138557, 0.29439825019245247, 0.294
5788974570236, 0.29477815413849745, 0.29499793902278976, 0.295240369102872,
0.2955077801385877, 0.29580274936887085, 0.2961281206064907, 0.296487031971
25403, 0.29688294654660846, 0.29731968627728517, 0.2978014694625851, 0.2983
3295224183065, 0.29891927451621614, 0.299566110805835, 0.30027972660328656,
0.30106704085756747, 0.3019356953058888, 0.30289413146913463, 0.30395167624
20648, 0.3051186371461442, 0.30640640847633155, 0.30782758977017366, 0.3093
961182671688, 0.3111274173205789, 0.3130385630886768, 0.31514847228932014,
0.31747811437990797, 0.3200507522640469, 0.32289221658154643, 0.32603121988
607797, 0.3294997186622537, 0.33333333333333326]
```





In [134…

```
function MPCWithUncertanties()
    t = collect(2:200) #eixo x
    loopCount = 200
    uMPC = Array{Float64}(undef, loopCount)
    xMPC = Array{Float64}(undef, loopCount)
    uMPC[1] = 0
    xMPC[1] = 0

    for i in t
        ################ optimization routine ####################
        m = Model(Ipopt.Optimizer);
        set_silent(m);
        movingTimeHorizon = 10; #prediction window
        @variable(m, xMPCLocal[1:movingTimeHorizon] >= xMPC[i-1]);
        @variable(m, uMPCLocal[1:movingTimeHorizon] >= 0);
        @constraint(m, xMPCLocal[1] == xMPC[i]) #initial point
        @constraint(m, xMPCLocal[10] == xEquilibrium) #initial point
        @constraint(m, uMPCLocal[1] == uMPC[i]) #initial point
        @constraint(m, uMPCLocal .<= 0.1)
        ####### have to force the setpoint constraint #########
        if (i >= 100)
            @constraint(m, xMPCLocal[2:10] .== xEquilibrium) #initial point
        end
        #######################################################
        for t in 1:movingTimeHorizon-1
            @constraint(
                m,
                xMPCLocal[t+1] == xMPCLocal[t]
                + β1*uMPCLocal[t]*(1 - xMPCLocal[t]) - ρβ1*(β1*uMPCLocal[t]*
```

```
                + β2*(1 - xMPCLocal[t])*xMPCLocal[t] - ρβ2*(1 - xMPCLocal[t]
                - (δ*xMPCLocal[t] - ρδ*(δ*xMPCLocal[t]))
            );
        end
        @objective(m, Min, sum(
            uMPCLocal .* uMPCLocal .+ (xMPCLocal .- xEquilibrium).^2 .+ λ
        ));
        optimize!(m);
        ##############################################################
        uMPC[i] = JuMP.value.(uMPCLocal)[2]
        xMPC[i] = xMPC[i - 1] + β1*uMPC[i - 1]*(1 - xMPC[i - 1]) + β2*(1 - x
    end

    return JuMP.objective_value.(m), xMPC, uMPC, sum(uMPCResult);
end;
MPCWithUncertanties()
```

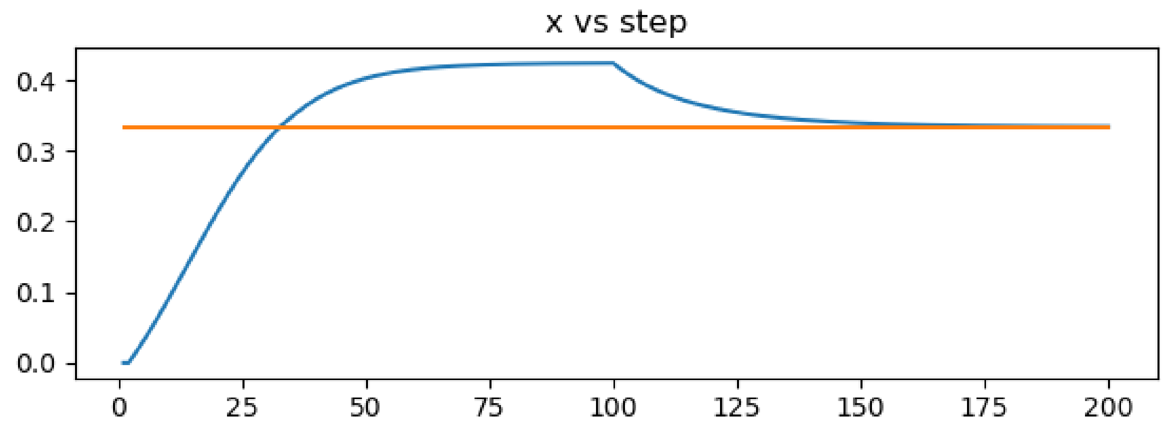
```
(101.43442942075757, [0.0, 0.0, 0.010000000996644748, 0.0203850020201989,
0.031138070832441817, 0.042238157707991, 0.053660075674897695, 0.0653745690
8742326, 0.07734847764089581, 0.08954499961880802  …  0.3339940982615921,
0.3339609945236356, 0.333929552370335, 0.3338996880969139, 0.33387132224507
71, 0.33384437938467953, 0.3338187879069022, 0.33379447982830235, 0.3337713
9060514044, 0.3337494589574241], [0.0, 0.10000000996644748, 0.1000000099664
4748, 0.10000000997881298, 0.10000000991396615, 0.10000000995625191, 0.1000
000099725796, 0.10000000997705102, 0.10000000998514587, 0.10000000997364052
  …  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], 2.9671206150474885)
```

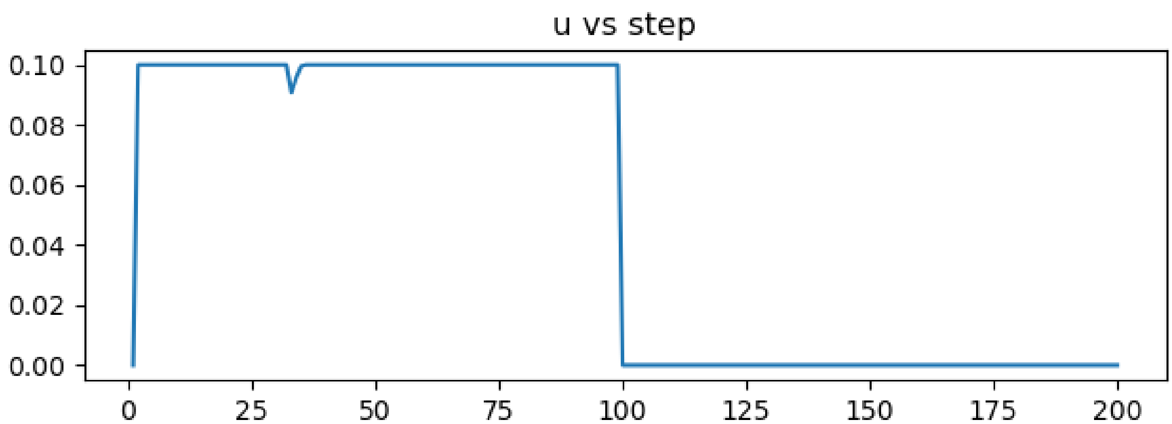
In [135…

```
loopCount = 200
costMPCWithUncertantiesResult,xMPCWithUncertantiesResult,uMPCWithUncertantie
println(costMPCWithUncertantiesResult);
println(xMPCWithUncertantiesResult);
println(uMPCWithUncertantiesResult);
println(controlAmountMPCWithUncertantiesResult);

clf(); #required for vscode on mac
fig = plt.figure();

ax1 = fig.add_subplot(2, 1, 1);
ax1.title.set_text("x vs step");
plt.plot(range(1, loopCount), transpose(xMPCWithUncertantiesResult));
plt.plot(range(1, loopCount), xEquilibrium .* ones(loopCount));

ax2 = fig.add_subplot(2, 1, 2);
ax2.title.set_text("u vs step");
plt.plot(range(1, loopCount), transpose(uMPCWithUncertantiesResult));

fig.tight_layout();
show()
gcf()
```

101.43442942075757
[0.0, 0.0, 0.010000000996644748, 0.0203850020201989, 0.031138070832441817,
0.042238157707991, 0.053660075674897695, 0.06537456908742326, 0.07734847764
089581, 0.08954499961880802, 0.10192405446810558, 0.1144427406112924, 0.127
05587998631573, 0.13971663655699157, 0.15237719209883974, 0.164989459328509
9, 0.1775058102820942, 0.18987979661212484, 0.20206683871132655, 0.21402486
19604878, 0.22571486099307203, 0.2371013764348594, 0.24815287284778165, 0.2
5884201126685163, 0.2691458144378642, 0.27904572733047917, 0.28852757945208
17, 0.29758145872458397, 0.3062015090880867, 0.31438566551881497, 0.3221353
40821187, 0.3294550784595456, 0.33635218495858155, 0.3422271354129996, 0.34
807282938817286, 0.3537988697638425, 0.3591747791627595, 0.364190792671190
1, 0.36886318499296333, 0.3732087056364321, 0.37724434379334343, 0.38098712
39283077, 0.3844539312136632, 0.3876613652929941, 0.39062562039501225, 0.39
336238952292235, 0.3958867902858825, 0.39821330989208503, 0.400355766862356
1, 0.40232728712658483, 0.40414029231340715, 0.40580649822020975, 0.4073369
216426306, 0.40874189394008575, 0.41003107990876886, 0.41121350072049645,
0.4122975598609534, 0.41329107116205716, 0.4142012881689511, 0.415034934212
41805, 0.4157982326723959, 0.41649693701848217, 0.4171363602998604, 0.41772
140383108003, 0.4182565848827965, 0.4187460632391574, 0.41919366652720563,
0.41960291425958757, 0.41997704056105734, 0.42031901557271373, 0.4206315655
464509, 0.4209171916565142, 0.4211781875659981, 0.42141665579419085, 0.4216
345229363624, 0.42183355379134757, 0.42201536445446075, 0.4221814344342049
7, 0.42233311785117683, 0.42247165377672774, 0.42259817576751807, 0.4227137
2065023305, 0.4228192366085501, 0.42291559062205164, 0.42300357530425126,
0.42308391518430244, 0.4231572724743494, 0.42322425236188593, 0.42328540786
395397, 0.42334124427755127, 0.42339222325824905, 0.4234387665567584, 0.423
481259441029, 0.42352005382942964, 0.4235554711586385, 0.42358780500806975,
0.4236173235009747, 0.4236442715007797, 0.423668872619755, 0.42369133105574
42, 0.4179487460067631, 0.4126440101638074, 0.40773494880338756, 0.40318452
79722018, 0.3989600898313868, 0.3950327213312159, 0.3913767297594248, 0.387
9692045578187, 0.38478964923292563, 0.3818196705710522, 0.3790427149743568,
0.37644384375680495, 0.3740095408192855, 0.37172754736667185, 0.36958671931
431925, 0.367576903816007, 0.365688831973963, 0.3639140252980889, 0.3622447
1389169466, 0.360673764674908, 0.35919461822993776, 0.35780123307663203, 0.
35648803637178983, 0.35524988017894715, 0.3540820025828211, 0.3529799930290
0296, 0.35193976135863986, 0.35095751008279535, 0.3500297095044078, 0.34915
307534926715, 0.34832454861285284, 0.34754127736854334, 0.3468006003157267,
0.3461000318746108, 0.3454372486588003, 0.34481007717759604, 0.344216482637
9929, 0.34365455873194184, 0.3431225183079521, 0.3426186848378511, 0.342141
4845997408, 0.34168943950711067, 0.34126116052186367, 0.3408553415958475,
0.3404707540914766, 0.3401062416373076, 0.3397607153790748, 0.3394331495907
9346, 0.3391225776141642, 0.3388280880977212, 0.3385488215100137, 0.3382839
6690364154, 0.338032758909214, 0.3377944749403085, 0.33756843259229413, 0.3
373539872194861, 0.3371505296765276, 0.3369574842111795, 0.3367743064968493
6, 0.33660048179422675, 0.33643552323232223, 0.3362789702000474, 0.33613038
68402291, 0.3359893606386362, 0.3358555011012141, 0.33572843851328266, 0.33
560782277496176, 0.33549332230754736, 0.3353846230259814, 0.335281427372938
5, 0.3351834534104001, 0.33509043396490185, 0.335002115822929, 0.3349182589
731996, 0.3348386358928137, 0.33476303087447035, 0.3346912393921546, 0.3346
2306750288384, 0.3345583312822721, 0.3344968562918289, 0.3344384770760529,
0.3343830366875131, 0.33433038623823436, 0.334280384475815, 0.3342328973828
109, 0.3341877977980152, 0.33414496505835295, 0.33410428466019343, 0.334065
64793895815, 0.3340289517659747, 0.3339940982615921, 0.3339609945236356, 0.
333929552370335, 0.3338996880969139, 0.3338713222450771, 0.3338443793846795
3, 0.3338187879069022, 0.33379447982830235, 0.33377139060514044, 0.33374945
89574241]

[0.0, 0.10000000996644748, 0.10000000996644748, 0.10000000997881298, 0.1000
0000991396615, 0.10000000995625191, 0.1000000099725796, 0.100000009977051O
2, 0.10000000998514587, 0.10000000997364052, 0.10000000998072853, 0.1000000
0996395378, 0.10000000993291285, 0.10000000998727085, 0.10000000998681925,
0.10000000998635311, 0.10000000998587373, 0.1000000099853826, 0.10000000998
487904, 0.10000000998435636, 0.1000000099838251, 0.10000000998328734, 0.100
00000998274523, 0.10000000998220095, 0.10000000998165683, 0.100000009981115
1, 0.10000000998057802, 0.10000000998004774, 0.10000000997952649, 0.1000000
0997901586, 0.10000000997851759, 0.10000000997803365, 0.09082015971188828,
0.09581193411540982, 0.09963694386150813, 0.10000000998588784, 0.1000000099
8576077, 0.1000000099856392, 0.1000000099855244, 0.10000000998541561, 0.100
000009985313, 0.10000000998521645, 0.10000000998512486, 0.100000009985039 8
1, 0.10000000998496031, 0.10000000998488609, 0.10000000998481691, 0.1000000
0998475257, 0.10000000998469279, 0.1000000099846373, 0.10000000998458593,
0.10000000998453838, 0.10000000998449442, 0.10000000998445384, 0.1000000099
8441638, 0.10000000998422967, 0.10000000998396552, 0.10000000998371998, 0.1
0000000998349215, 0.10000000998328099, 0.10000000998308554, 0.1000000099829
0486, 0.10000000998273803, 0.10000000998258411, 0.10000000998244224, 0.1000
0000998231158, 0.10000000998219136, 0.1000000099820808, 0.1000000099819792
1, 0.10000000998188588, 0.10000000998180021, 0.10000000998172161, 0.1000000
0998164953, 0.10000000998158344, 0.10000000998152289, 0.10000000998146741,
0.1000000099814166, 0.10000000998137008, 0.10000000998132749, 0.10000000998
128854, 0.1000000099812529, 0.10000000998122029, 0.10000000998119045, 0.100
00000998116318, 0.10000000998113825, 0.10000000998111548, 0.10000000998109 4
62, 0.10000000998107561, 0.1000000099810582, 0.1000000099810423, 0.10000000
998102779, 0.1000000099810145, 0.10000000998100238, 0.10000000998099132, 0.
10000000998098121, 0.10000000998097197, 0.10000000998096353, 0.10000000998O
95582, 0.1000000099809488, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0]
2.9671206150474885





4. Results and discussion

Results summary

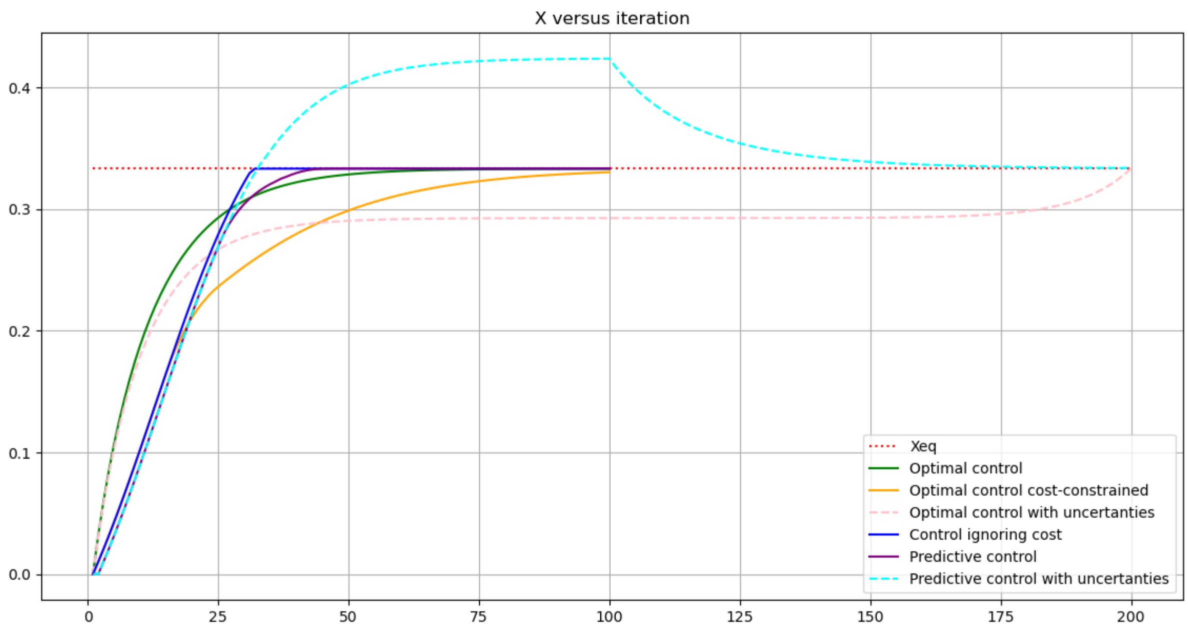
Control	Control amount	Control amount normalized*	Time to reach equilibrium
Optimal control	2.776115114183729	90.89627927478766	55
Optimal control with cost limit	1.8000000059172825	58.93606587008714	98
No cost concerns	3.0541570417764654	100.0	32
Predictive control	2.9671206150474885	97.15023079892606	41

normalization = original control amount (100 / highest control amount)

In [150…

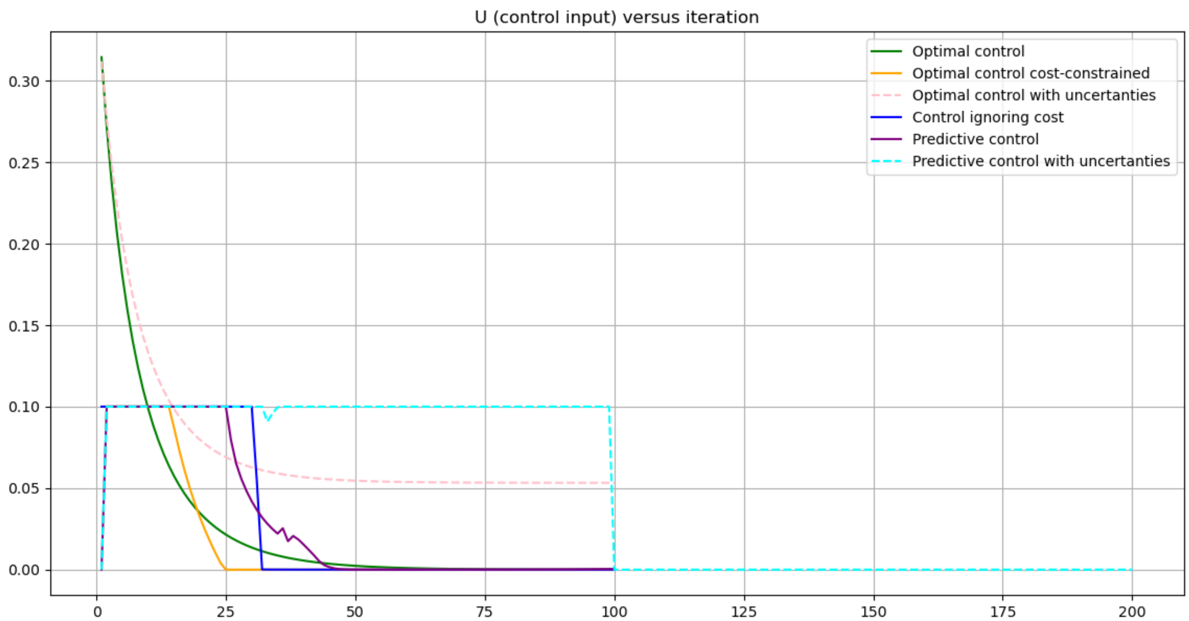
```
#X graph
#costOptimalControlResult, xOptimalControlResult, uOptimalControlResult, cor
#costOptimalControlLimitedCostsResult, xOptimalControlLimitedCostsResult, uC
#costFastestEquilibriumResult, xFastestEquilibriumResult, uFastestEquilibriu
#costMPCResult,xMPCResult,uMPCResult,controlAmountMPCResult = MPC();
#costOptimalControlWithUncertantiesResult, xOptimalControlWithUncertantiesRe

clf()
t1 = 1:100
t2 = 1:200
figure(figsize=(14,7))
```

```
plot(t2, xEquilibrium .* ones(200), color="red", linestyle="dotted", label="
plot(t1, xOptimalControlResult, color="green", label="Optimal control")
plot(t1, xOptimalControlLimitedCostsResult, color="orange", label="Optimal c
plot(t2, xOptimalControlWithUncertantiesResult, color="pink", label="Optimal
plot(t1, xFastestEquilibriumResult, color="blue", label="Control ignoring co
plot(t1, xMPCResult, color="purple", label="Predictive control")
plot(t2, xMPCWithUncertantiesResult, color="cyan", label="Predictive control
legend()
title("X versus iteration");
grid("on")
gcf()
```



In [151…

```
#cost graph
#costOptimalControlResult, xOptimalControlResult, uOptimalControlResult, con
#costOptimalControlLimitedCostsResult, xOptimalControlLimitedCostsResult, uO
#costFastestEquilibriumResult, xFastestEquilibriumResult, uFastestEquilibriu
#costMPCResult,xMPCResult,uMPCResult,controlAmountMPCResult = MPC();
#costOptimalControlWithUncertantiesResult, xOptimalControlWithUncertantiesRe

clf()
figure(figsize=(14,7))
t = 1:100
plot(t, uOptimalControlResult, color="green", label="Optimal control")
plot(t, uOptimalControlLimitedCostsResult, color="orange", label="Optimal co
plot(t, uOptimalControlWithUncertantiesResult[1:100], color="pink", label="O
plot(t, uFastestEquilibriumResult, color="blue", label="Control ignoring cos
plot(t, uMPCResult, color="purple", label="Predictive control")
plot(t2, uMPCWithUncertantiesResult, color="cyan", label="Predictive control
legend()
title("U (control input) versus iteration");
grid("on")
gcf()
```



5. Conclusion

Based on the results presented above some conclusions can be draw.

All control models were able to reach the equilibrium within 100 iterations when there were no uncertainties to the predefined constants.

However, some models accomplish this task faster and more efficient than others. From the table presented in the previous section, two control techniques appear to be most suitable for a marketing campaign:

- **Regular optimal control[1]**
  - **Control amount = 2.776115114183729**
  - **Time to reach equilibrium = 55 iterations**
- **Optimal control with no cost concerns [2]**
  - **Control amount = 3.0541570417764654**
  - **Time to reach equilibrium = 32 iterations**

Comparing these two, it is clear that even though [1] uses 10% less control input, it takes 70% more time to reach X_{eq} making [2] the overall best solution.

Another important fact to be highlighted is that all control techniques struggle to deal with uncertainties in the estimation of $\beta 1$, $\beta 2$ and δ. The analysis was made based on the worst case scenario which is when these constants are underestimated and slow down the information spread dynamics. What is happens in this case is that the model stabilizes around a different set-point other than X_{eq}. To solve this issue, it's necessary for "force" $x(t)$ to be equals X_{eq} after a certain number of iterations by using the following constraint:

```
@constraint(m, xOptimalControl[200] == xEquilibrium)
```

Without these uncertainties, all models naturally converge to X_{eq} in order to minimize the cost function while the control input goes to 0 as theoretically expected.

6. Bibliography

1. Information Spread in a Social Media Age Modeling and Control, Michael Muhlmeyer and Shaurya Agarwal
2. Barg, Michael C. (2016) "Find, Process, and Share: An Optimal Control in the Vidale-Wolfe Marketing Model," CODEE Journal: Vol. 11, Article 1