École de Technologie Supérieure

Devoir 2– 19 février 2015

Techniques de simulation numériques

SYS810

Milan Assuied

https://www.sharelatex.com/project/

1 Problème 1

1. Représentation discrète de la commande d'hélicoptère:

   Note: La substitution de type Halijak présentée dans cet exercice présente une différence avec la référence fournie dans le cours.
   Cette différence pourrait s'expliquer par une erreur dans les matrices de substitution, toutefois cette erreur est faible et les matrices ont été vérifiées.
   Les résultats étant cohérents sont donc présentés tels quels. La source de l'écart restant non identifiée.

   Dans cet exercice, on représente la commande d'un rotor d'hélicoptère à l'aide de quatre modèles de simulation différents:

   - Modèle continu,
   - Modèle discrétisé par substitution de type Tutsin,
   - Modèle discrétisé par substitution de type Tutsin décalé par un retard pur, afin de rendre le système explicite,
   - Modèle discrétisé par substitution de type Halijak.

   Les fonctions de transfert sont déterminées en utilisant les méthodes de la classe Discretizer. Le résultat suivant est obtenu et imprimé depuis la fenêtre de sortie MatLab.

```
Continuous transfer function
System:

ans =

        9.8 s^2 - 4.9 s + 61.4
  ----------------------------------
  s^4 + 0.44 s^3 - 0.007 s^2 + 0.11 s

Continuous-time transfer function.

Compensator:

ans =

    1.93 s^3 + 1.72 s^2 + 0.43 s + 0.11
  ---------------------------------------
  s^4 + 9.76 s^3 + 40.9 s^2 + 76.6 s + 136

Continuous-time transfer function.

Closed Loop:

ans =

              9.8 s^6 + 90.75 s^5 + 414.4 s^4 + 1150 s^3 + 3469 s^2 + 4037 s + 8350
↙
----------------------------------------------------------------------------------↙
---
  s^8 + 10.2 s^7 + 45.19 s^6 + 113.6 s^5 + 177.9 s^4 + 178.1 s^3 + 112.1 s^2 + 40.82 s +↙
6.754

Continuous-time transfer function.

Tutsin discretization
System:

ans =

  0.003857 z^4 - 3.876e-05 z^3 - 0.007713 z^2 + 0.0001167 z + 0.003934
  ---------------------------------------------------------------------
             z^4 - 3.983 z^3 + 5.948 z^2 - 3.948 z + 0.9826

Sample time: 0.04 seconds
Discrete-time transfer function.

Compensator:

ans =

  0.03241 z^4 - 0.06368 z^3 - 0.001135 z^2 + 0.06368 z - 0.03128
```

```
  ------------------------------------------------------------
         z^4 - 3.621 z^3 + 4.923 z^2 - 2.979 z + 0.6769

Sample time: 0.04 seconds
Discrete-time transfer function.

Closed Loop:

ans =

  0.003857 z^8 - 0.014 z^7 + 0.01141 z^6 + 0.01637 z^5 - 0.03173 z^4 + 0.009278 z^3 +↙
0.0138 z^2 - 0.01164 z + 0.002663
↙
----------------------------------------------------------------------------------------↙
----------------------------
               z^8 - 7.604 z^7 + 25.29 z^6 - 48.07 z^5 + 57.1 z^4 - 43.4 z^3 + 20.62 z^2↙
- 5.599 z + 0.665

Sample time: 0.04 seconds
Discrete-time transfer function.

Tutsin discretization with delay
System:

ans =

  0.003857 z^4 - 3.876e-05 z^3 - 0.007713 z^2 + 0.0001167 z + 0.003934
  ------------------------------------------------------------------
         z^5 + z^4 - 3.983 z^3 + 5.948 z^2 - 3.948 z + 0.9826

Sample time: 0.04 seconds
Discrete-time transfer function.

Compensator:

ans =

  0.03241 z^4 - 0.06368 z^3 - 0.001135 z^2 + 0.06368 z - 0.03128
  --------------------------------------------------------------
      z^5 + z^4 - 3.621 z^3 + 4.923 z^2 - 2.979 z + 0.6769

Sample time: 0.04 seconds
Discrete-time transfer function.

Closed Loop:

ans =


  0.003857 z^9 + 0.003818 z^8 - 0.02172 z^7 + 0.01153 z^6 + 0.0203 z^5 - 0.03173 z^4 +↙
0.009278 z^3 + 0.0138 z^2
```

```
↙
- 0.01164 z + 0.002663

↙
----------------------------------------------------------------------------------------↙
----------------------------

         z^9 - 2.982 z^8 - 1.656 z^7 + 21.34 z^6 - 47.08 z^5 + 57.1 z^4 - 43.4 z^3 +↙
20.62 z^2 - 5.599 z + 0.665


Sample time: 0.04 seconds
Discrete-time transfer function.

Halijak discretization
System:

ans =

       0.01556 z^3 - 0.03128 z^2 + 0.01588 z
  ------------------------------------------------
  1.018 z^4 - 4.053 z^3 + 6.053 z^2 - 4.018 z + 1

Sample time: 0.04 seconds
Discrete-time transfer function.

Compensator:

ans =

  0.0772 z^4 - 0.2288 z^3 + 0.2261 z^2 - 0.07446 z
  ------------------------------------------------
   1.39 z^4 - 5.103 z^3 + 7.04 z^2 - 4.327 z + 1

Sample time: 0.04 seconds
Discrete-time transfer function.

Closed Loop:

ans =

    0.02164 z^7 - 0.1229 z^6 + 0.2913 z^5 - 0.3686 z^4 + 0.2627 z^3 - 0.09998 z^2 +↙
0.01588 z
↙
----------------------------------------------------------------------------------------↙
-----
  1.415 z^8 - 10.83 z^7 + 36.26 z^6 - 69.4 z^5 + 83.05 z^4 - 63.63 z^3 + 30.48 z^2 -↙
8.345 z + 1

Sample time: 0.04 seconds
```

2. Modèles utilisés pour la simulation:

   Le modèle suivant est utilisé pour la simulation:

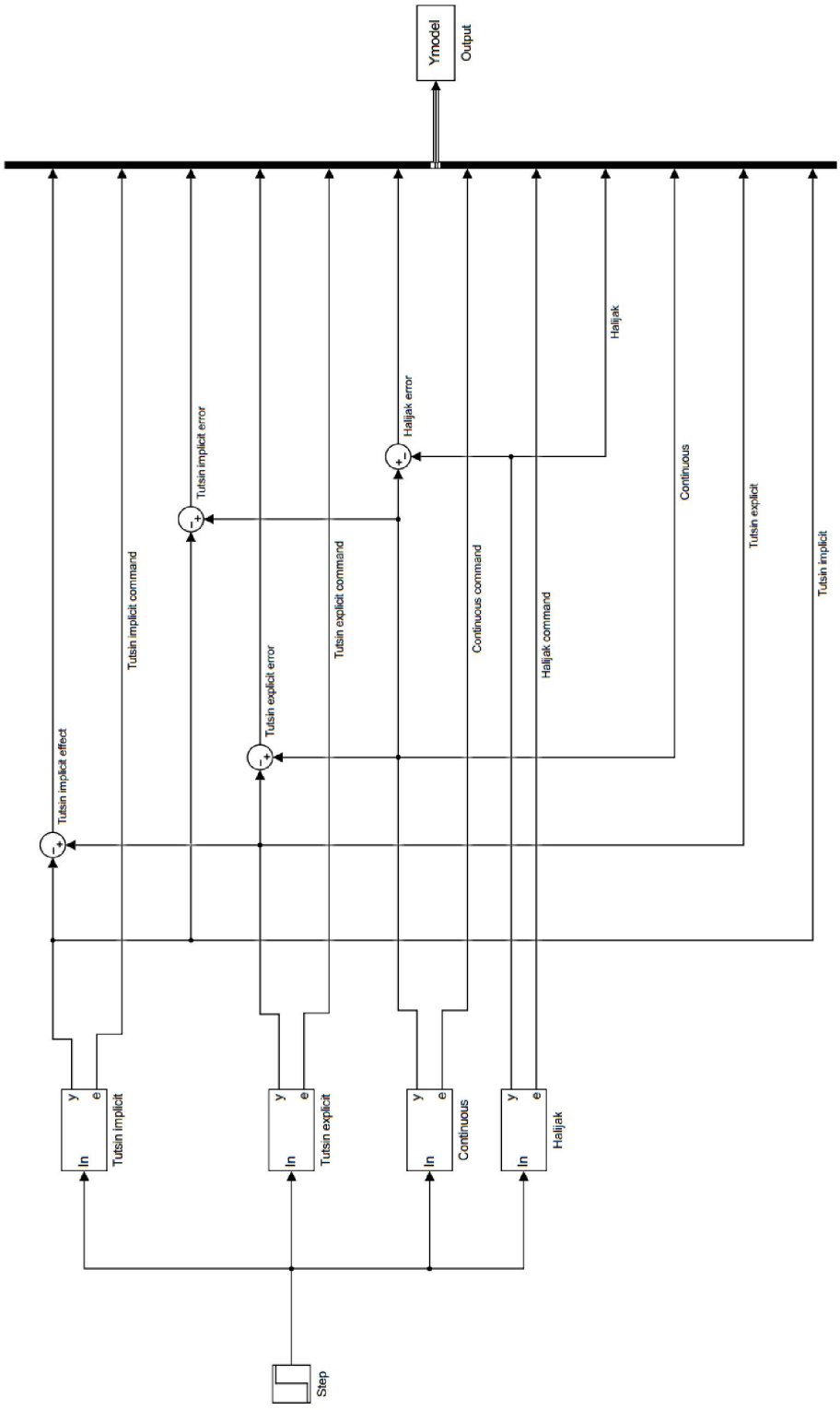

Figure 1: Modèle de simulation de commande d'Hélicoptère - Quatre représentations.

Le fait que Tustin soit une méthode implicite génère une boucle algébrique qui doit être brisée pour permettre le calcul. En effet une boucle algébrique revient à devoir connaître au même instant l'entrée et la sortie du système, ce qui n'est pas causal. Pour briser la boucle algébrique, il est rajouté un retard à la fonction de transfert discrète. Toutefois, cet ajout fausse la fonction qui représente le système et entraîne une erreur de précision.. Nous verrons dans les résultats que SimuLink peut - être en mesure de résoudre les boucles algébriques grâce des itérations successives dans la mesure ou celles-ci convergent.

Afin de comparer les effets d'une fonction de transfert implicite contre cette même fonction retardée, on utilise deux sous-systèmes différents afin de réaliser une simulation en parallèle. Les sous-systèmes contenant les fonctions de transfert continues et discrétisées selon Halijak sont similaires et non représentés:

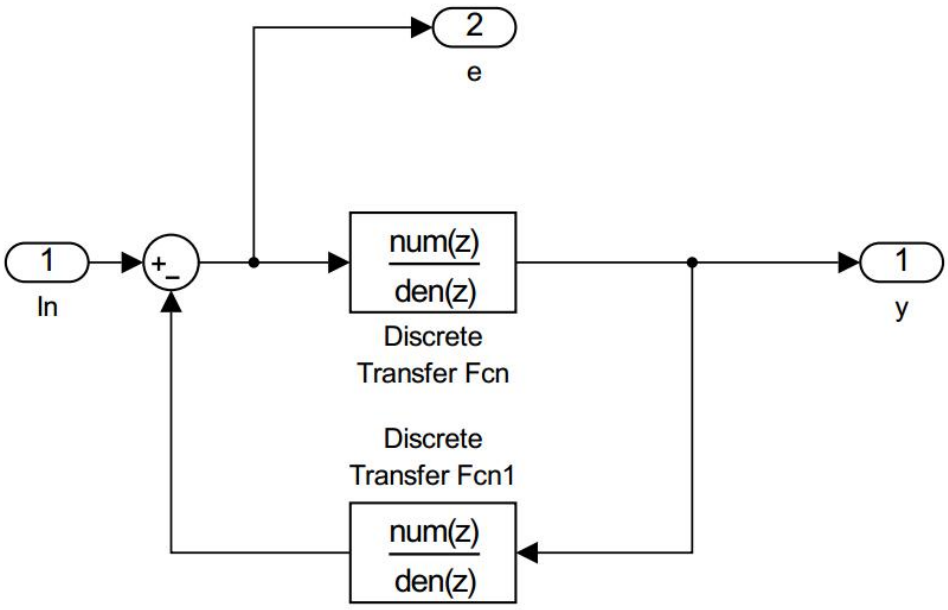

Figure 2: Sous-Système Tutsin

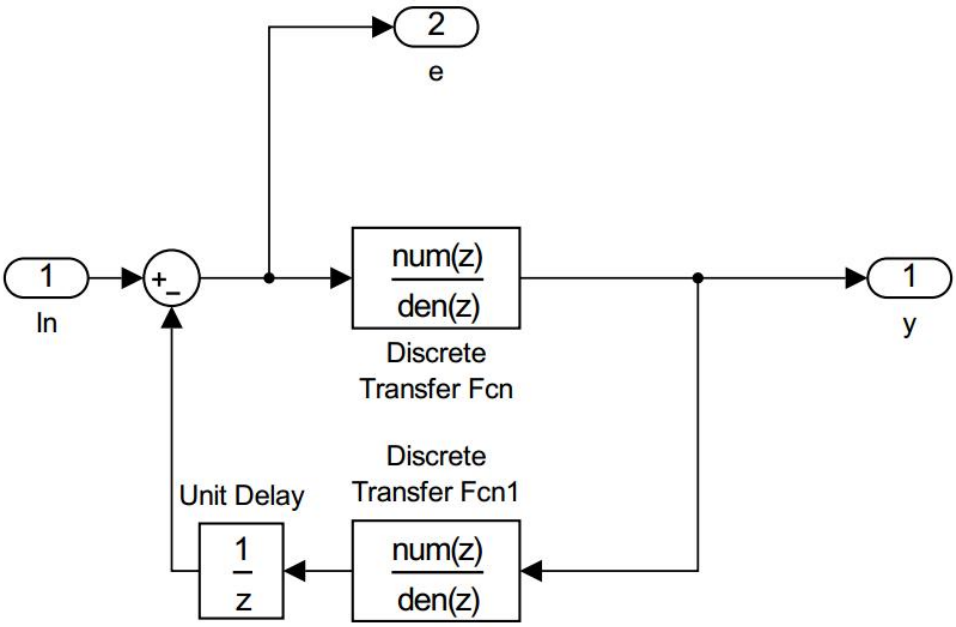

Figure 3: Sous-Système Tutsin

Note: Il aurait pu être utilisé la méthode de retard de la classe Discretizer pour ne pas avoir à modifier le sous-système en lui rajoutant un retard pur.

3. Résultats:

   On présente ici la réponse temporelle du système à une entrée de type échellon d'amplitude 0.1R.

   - Réponse temporelle

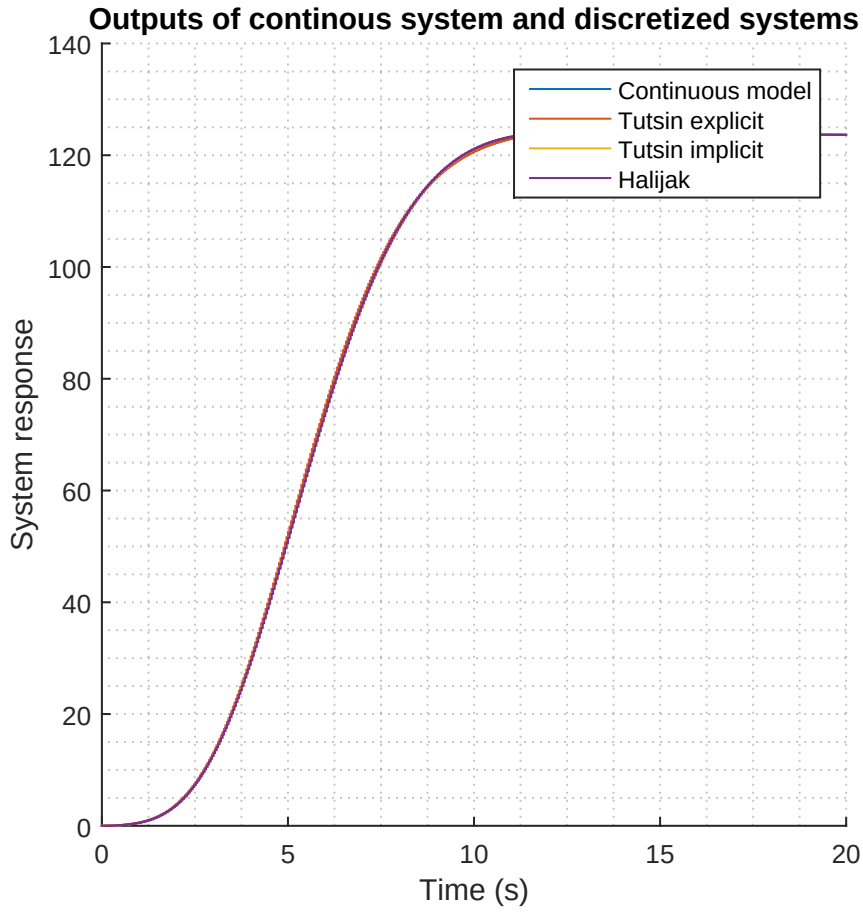

Figure 4: Réponse temporelle

On constate qu'il y a peu de différence sur la réponse du système. Ces écarts sont précisés sur la figure suivante:

- Écart par rapport au modèle continu:

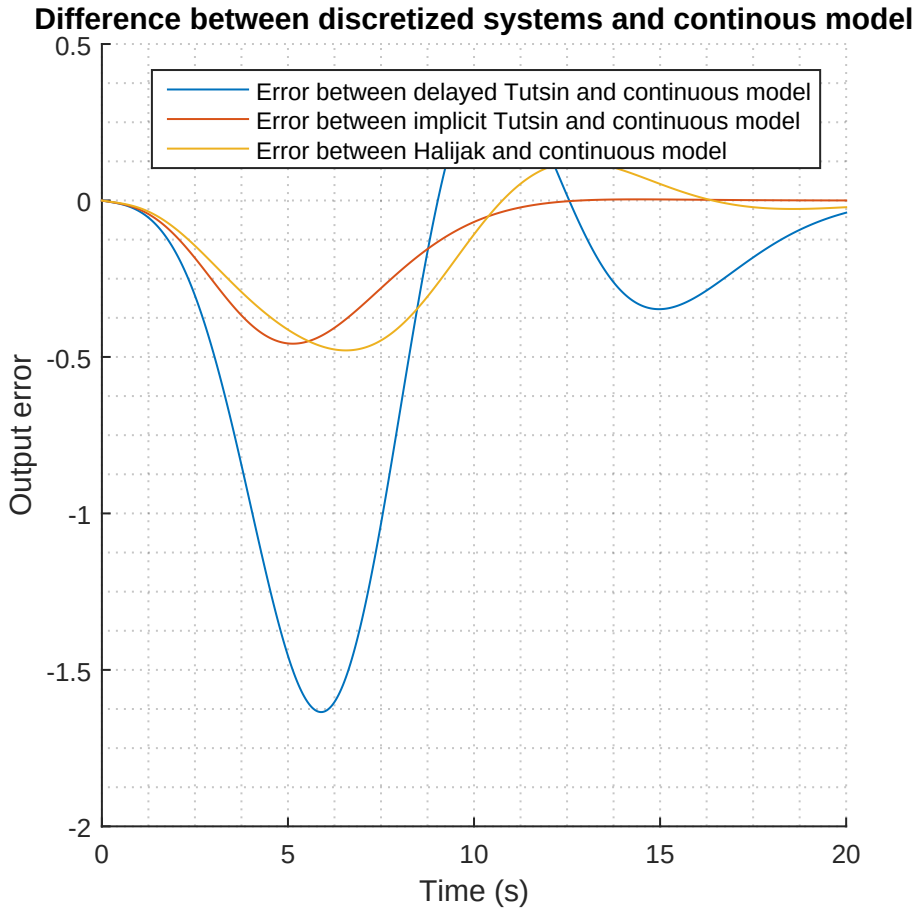

Figure 5: Écarts entre modèle continu et modèles discret

Il est ici intéressant de constater que la fonction de transfert implicite fournit le meilleur résultat.

Ceci n'est réalisable que parce que Simulink est en mesure de résoudre la boucle algébrique, au prix de performances de simulation dégradées. Il est important de noter que cette résolution n'est pas offerte par tous les logiciels, qu'elle n'est pas garantie par Simulink, et qu'elle doit être comparée aux résultats fournis par des systèmes causaux afin de garantir son exactitude.

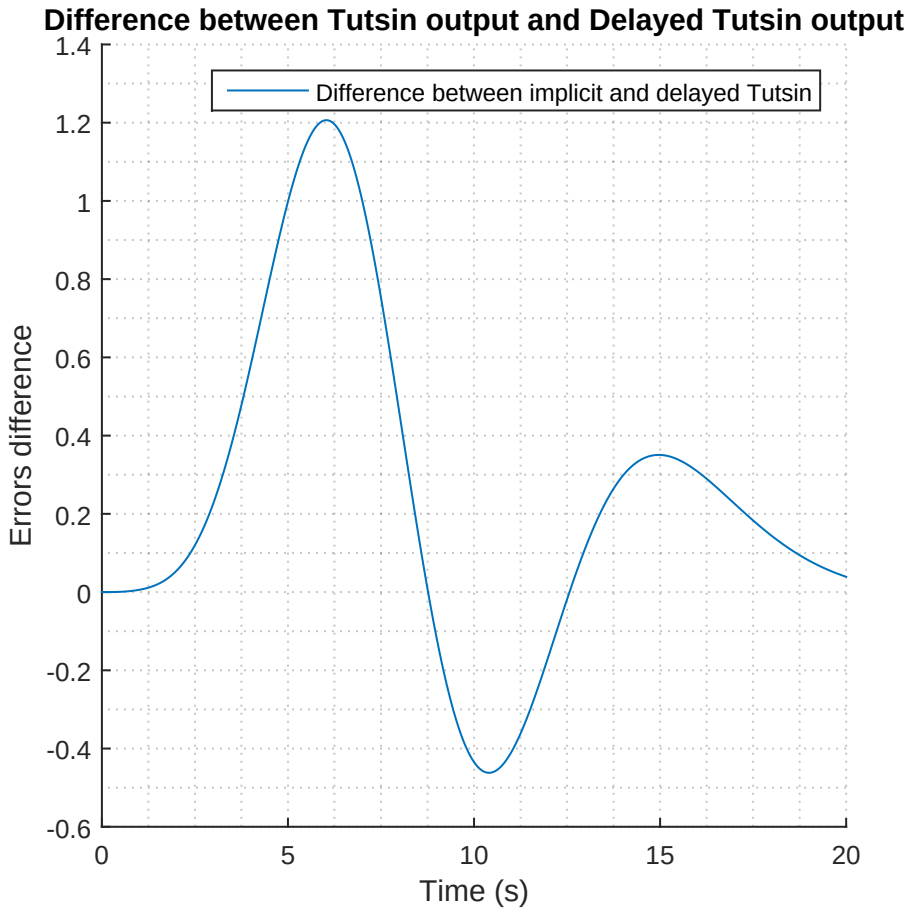

Figure 6: Écarts entre Tutsin et Tutsin retardé

Finalement, on présente la commande appliquée en entrée du système. Le système étant en boucle fermée, il s'agit de l'erreur entre l'entrée et sortie qui doit être annulée.

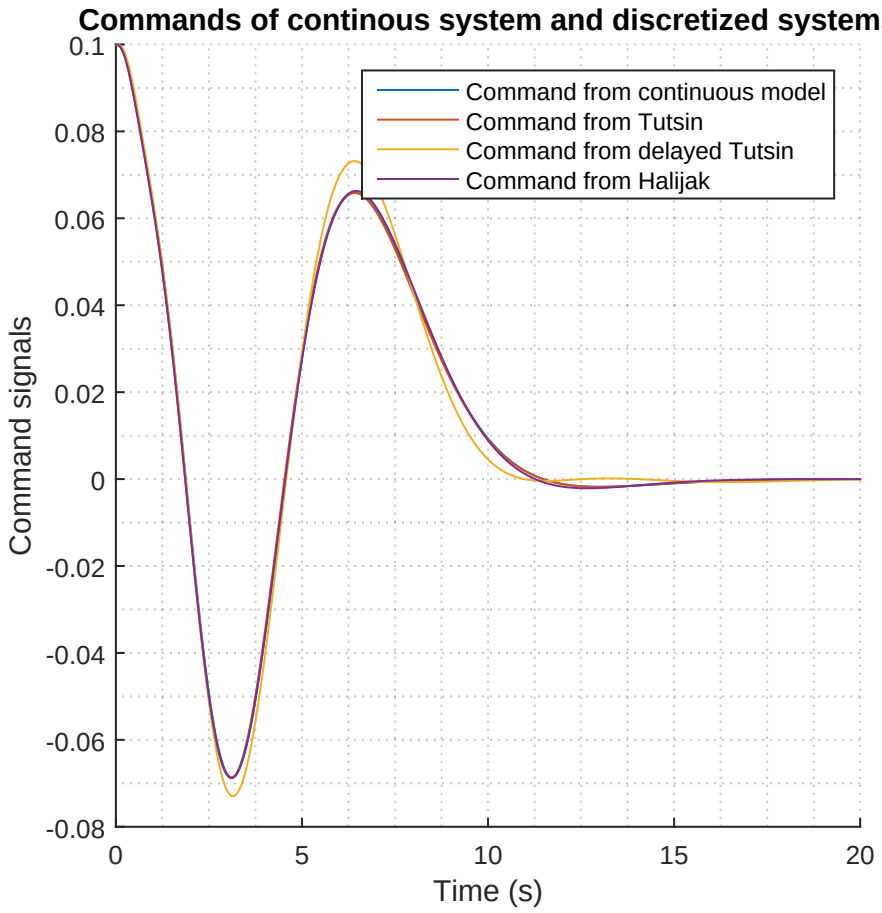

Figure 7: Signal de commande

2 Problème 2

Discrétistation suivant plusieurs méthodes

1. Représentation discrète d'une fonction de transfert:
   Pour cet exercice, nous utilisons de nouveau la classe Discretizer. Cette classe instanciée à partir d'une fonction de transfert continue permet d'accéder aux fonctions de transfert discrètes du type suivant:

   - Bloqueur d'ordre 0,
   - Tutsin
   - Halijak jusqu'à l'ordre 5
   - Boxer-Thalor jusqu'à l'ordre 5

   Les fonctions de transfert sont déterminées en utilisant les méthodes de la classe Discretizer. Le résultat suivant est obtenu et imprimé depuis la fenêtre de sortie MatLab.

```
Continuous transfer function

ans =

             100
  -------------------------
  s^3 + 11 s^2 + 30 s + 200

Continuous-time transfer function.

Tutsin discretization

ans =

  0.007576 z^3 + 0.02273 z^2 + 0.02273 z + 0.007576
  -------------------------------------------------
          z^3 - 2.061 z^2 + 1.485 z - 0.303

Sample time: 0.1 seconds
Discrete-time transfer function.

Boxer-Thalor discretization

ans =

              0.05 z^2 + 0.05 z
  --------------------------------------
  1.575 z^3 - 3.225 z^2 + 2.325 z - 0.475

Sample time: 0.1 seconds
Discrete-time transfer function.

Halijak discretization

ans =

         0.05 z^2 + 0.05 z
  ----------------------------
  2.1 z^3 - 4.8 z^2 + 3.9 z - 1

Sample time: 0.1 seconds
Discrete-time transfer function.

>>
```

De plus, Discretizer permet d'exécuter l'équation récurrente associée à l'un des types de discrétisation.

Pour valider nos équations, nous traçons les courbes suivantes associées à chaque technique:

- Courbe issue d l'équation récurente
- Courbe issue d'un modèle simulink éxécutant la fonction de transfert discrètisée
- Courbe issue d'un modèle simulink exécutant la fonction de transfert continue

2. Étude de stabilité:

   Avant de tracer les réponses impulsionnelles, nous étudions la stabilité du système en traçant son diagramme de Nyquist ainsi que le digramme Pôles-Zéros des fonctions de transfert dicrètisées:

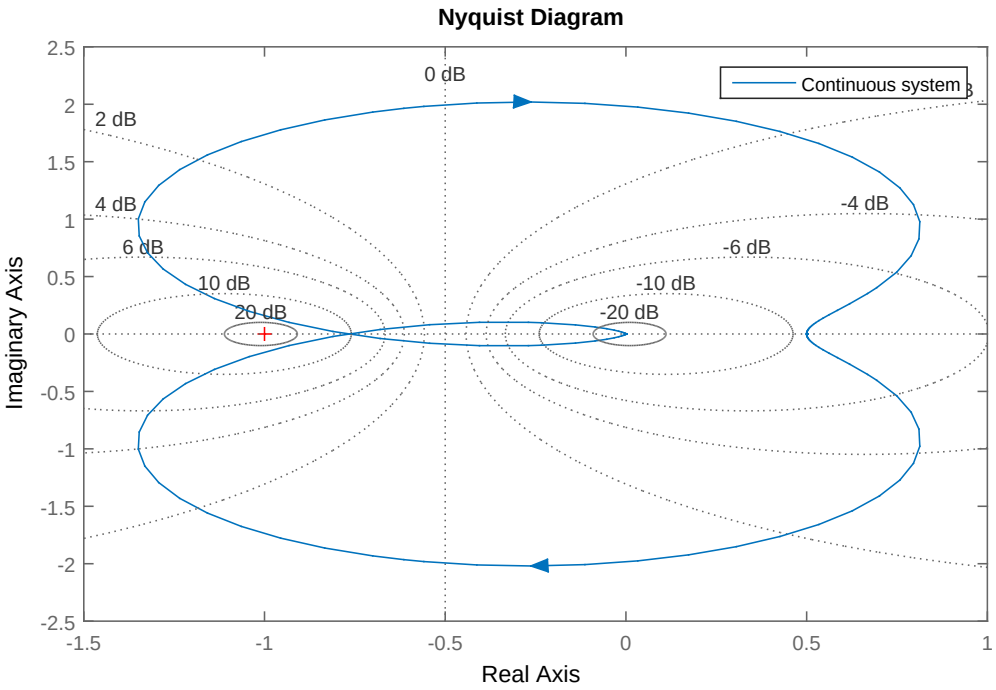

Figure 8: Diagramme de Nyquist

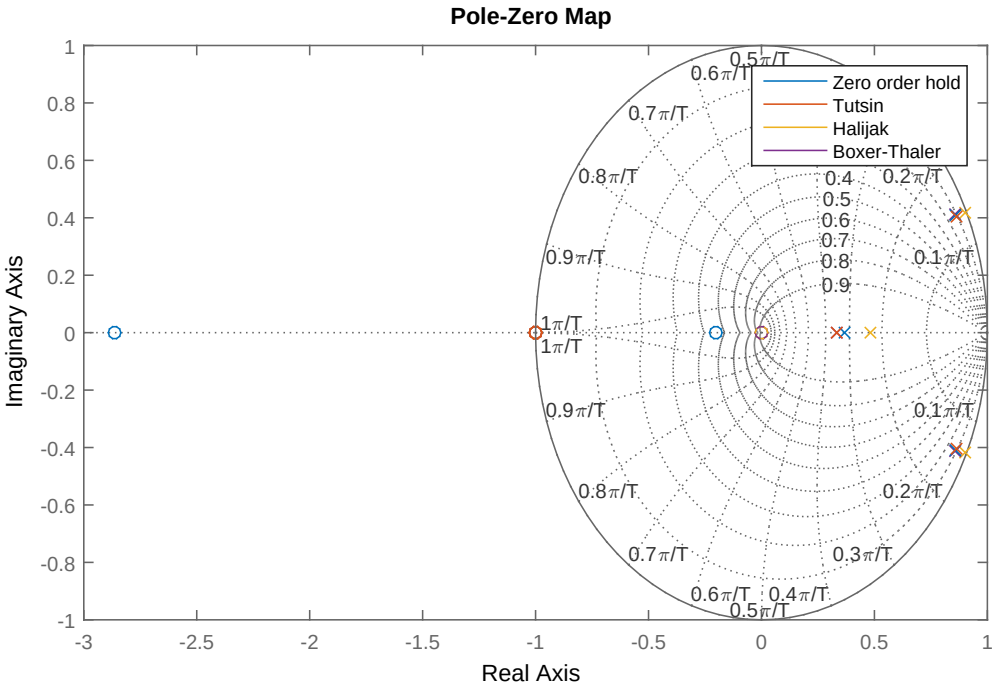

Figure 9: Pôles - Zéros discrets

Le point (-1) étant laissé à gauche du diagramme de Nyquist, on en déduit que le système est stable en boucle fermée.
De même, les pôles de chacune des représentations discrètes du système étant situés dans le cercle de rayon (1), on en déduit que chacune de ces représentation est stable.
On note toutefois que l'un des pôles de la fonction obtenue par Halijak présente un éloignement sensible des pôles obtenus via les autres représentations, ce qui devrait se traduire sur les résultats.

3. Résultats:

Bloqueur d'ordre 0

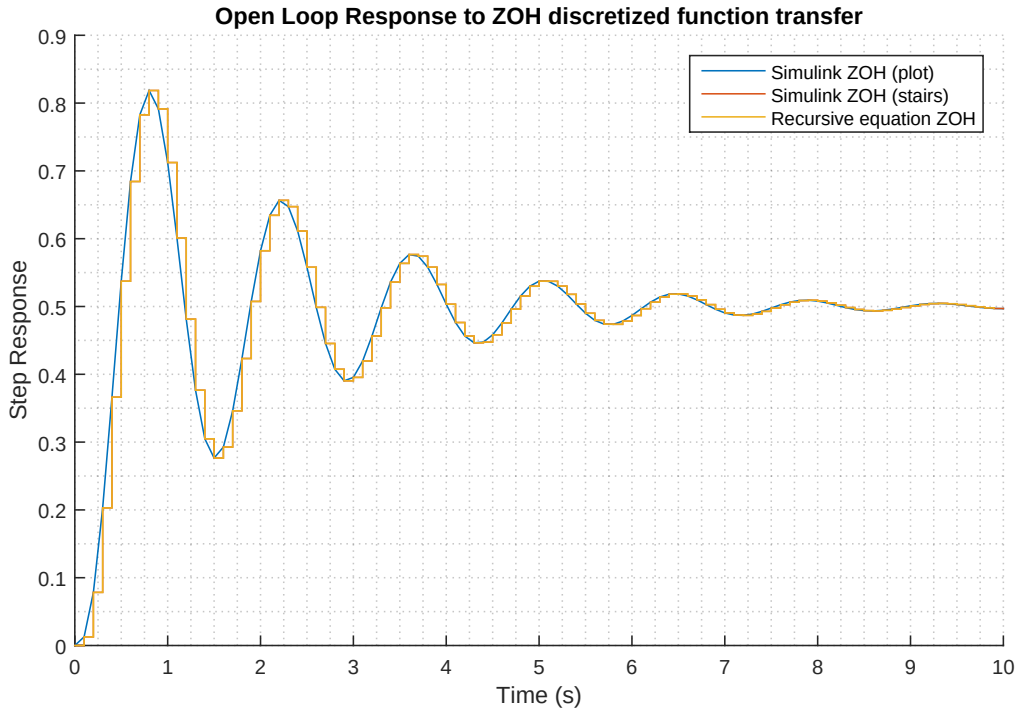

Figure 10: Courbes associées au Bloqueur d'Ordre Zero

Tutsin

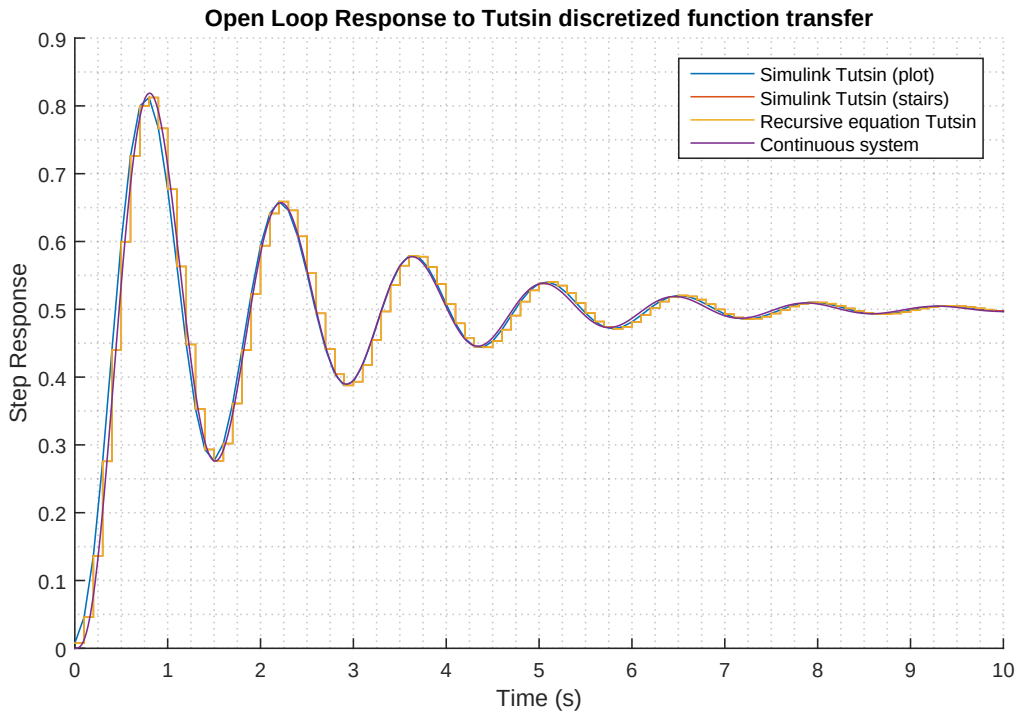

Figure 11: Courbes associées à Tutsin

Halijak

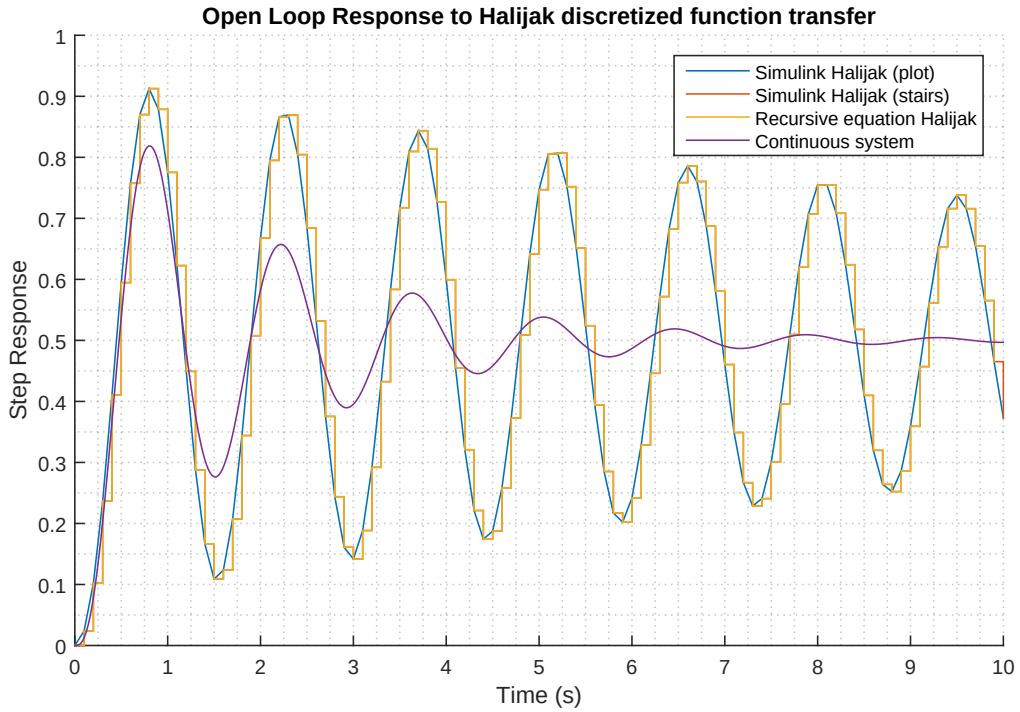

Figure 12: Courbes associées à Halijak

Note: Comme attendu, ce modèle est convergent mais sa précision est incorrecte. Contrairement à Tutsin, qui est la méthode implicite à 1-pas la plus précise, Halijak présente une erreur d'ordre 1.
Une solution est de diminuer la période d'échantillonnage pour obtenir une meilleure précision, tout en essayant de rester stable.

Nous divisons la période d'échantillonnage par 10 et traçons le diagramme Pôles-Zéros:

Ce diagramme confirme que la fonction de transfert est toujours stable, on peut alors simuler le système.

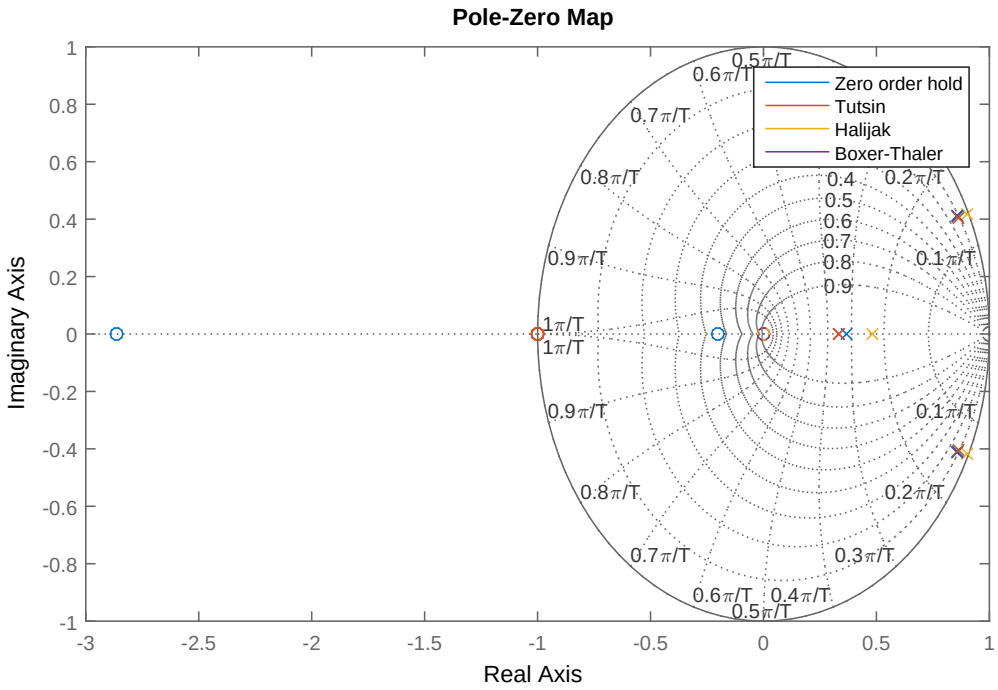

Figure 13: Diagramme Pôle-Zéro avec T = 0.01s

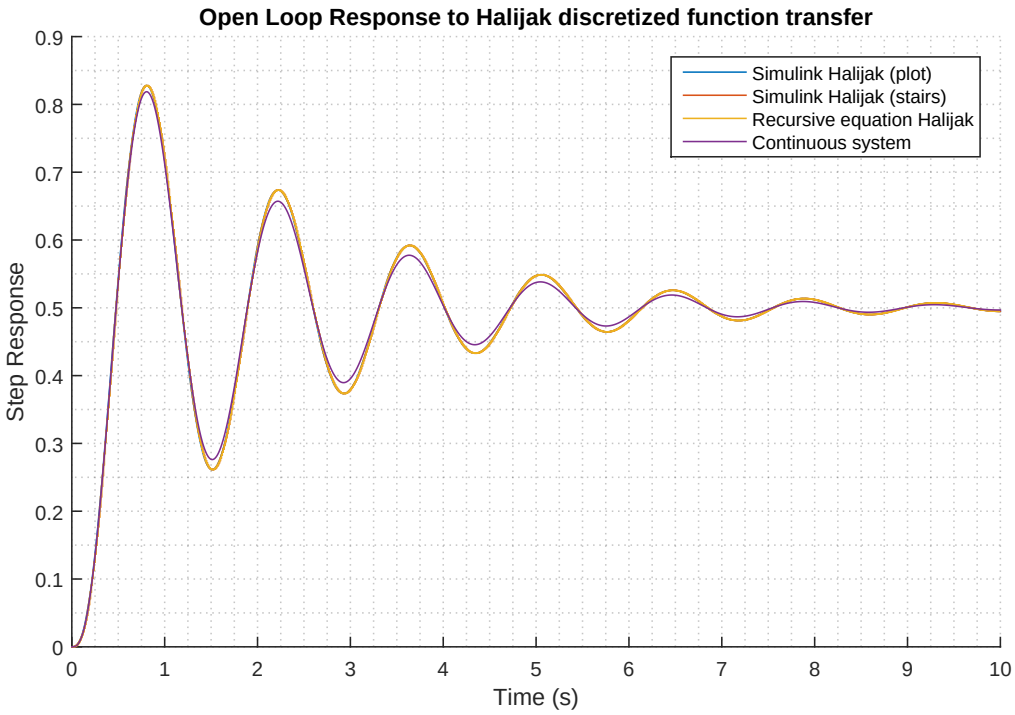

Figure 14: Courbes associées à Halijak avec T = 0.01s

4. Boxer-Thaler

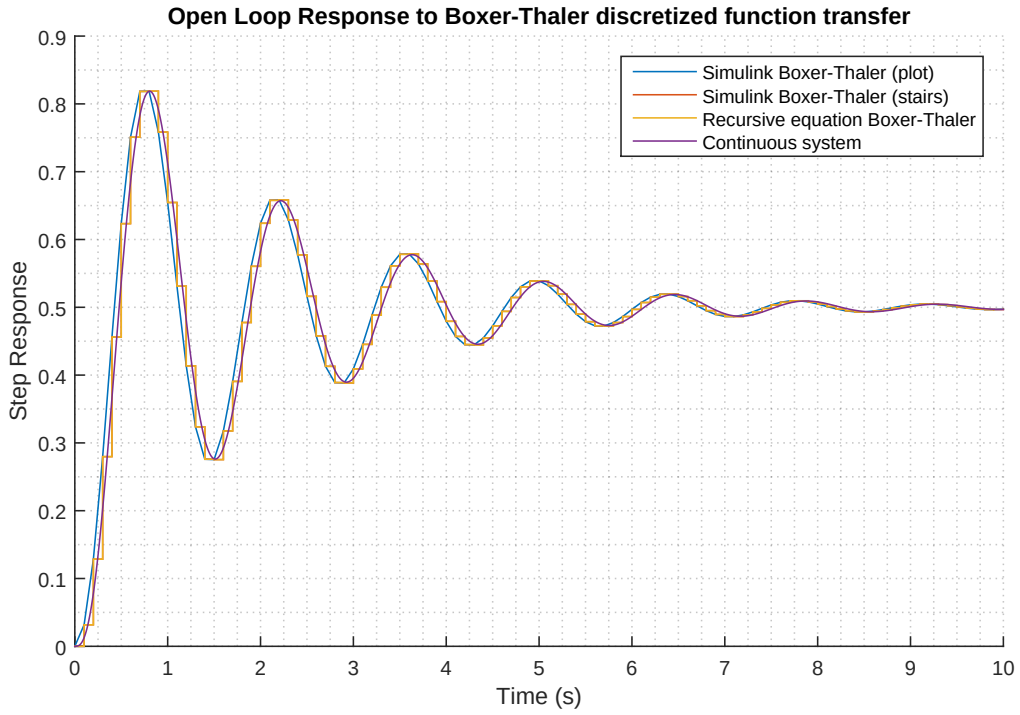

Figure 15: Courbes associées à Boxer-Thaler

ANNEXES: Code Matlab

1. Exercice1 : Code d'exécution

```
addpath(genpath('..'));

%Parametres
close all;
clear all; %#ok<CLSCR>
clc;

%Parametres
wSampleTime = 0.04;
wSimulationTime=20;
wMaxStep=wSampleTime/2;

%Fonctions de transfert
wSysNum  = [0,0,9.8,-4.9,61.4];
wSysDen  = [1,0.44,-0.007,0.11,0];
wCompNum = [0,1.93,1.72,0.43,0.11];
wCompDen = [1,9.76,40.9,76.6,136];

wSystem      = Discretizer(wSampleTime,wSysNum,wSysDen);
wCompensator = Discretizer(wSampleTime,wCompNum,wCompDen);

%Tutsin:
[wSys_Tutn,wSys_Tutd]   = wSystem.mGetDiscreteTf('tutsin');
[wComp_Tutn,wComp_Tutd] = wCompensator.mGetDiscreteTf('tutsin');

%Halijak
[wSys_Haln,wSys_Hald]   = wSystem.mGetDiscreteTf('halijak');
[wComp_Haln,wComp_Hald] = wCompensator.mGetDiscreteTf('halijak');


%Parametrage simulation
model='HelicoModel';
load_system(model)
tic

wSaveFileName     = 'Ymodel';

set_param(model,'StopFcn','save(wSaveFileName,wSaveFileName)');
set_param(strcat(model,'/Output'),'VariableName',wSaveFileName);

set_param(strcat(model,'/Tutsin implicit'),'S_num','wSys_Tutn');
set_param(strcat(model,'/Tutsin implicit'),'S_den','wSys_Tutd');
set_param(strcat(model,'/Tutsin implicit'),'C_num','wComp_Tutn');
set_param(strcat(model,'/Tutsin implicit'),'C_den','wComp_Tutd');
set_param(strcat(model,'/Tutsin implicit'),'T','wSampleTime');

set_param(strcat(model,'/Tutsin explicit'),'S_num','wSys_Tutn');
set_param(strcat(model,'/Tutsin explicit'),'S_den','wSys_Tutd');
set_param(strcat(model,'/Tutsin explicit'),'C_num','wComp_Tutn');
set_param(strcat(model,'/Tutsin explicit'),'C_den','wComp_Tutd');
set_param(strcat(model,'/Tutsin explicit'),'T','wSampleTime');
```

```
set_param(strcat(model,'/Halijak'),'S_num','wSys_Haln');
set_param(strcat(model,'/Halijak'),'S_den','wSys_Hald');
set_param(strcat(model,'/Halijak'),'C_num','wComp_Tutn');
set_param(strcat(model,'/Halijak'),'C_den','wComp_Tutd');
set_param(strcat(model,'/Halijak'),'T','wSampleTime');

set_param(strcat(model,'/Continuous'),'S_num','wSysNum');
set_param(strcat(model,'/Continuous'),'S_den','wSysDen');
set_param(strcat(model,'/Continuous'),'C_num','wCompNum');
set_param(strcat(model,'/Continuous'),'C_den','wCompDen');

set_param(model, 'StopTime', 'wSimulationTime');
set_param(model, 'MaxStep', 'wMaxStep');
set_param((strcat(model,'/Output')), 'SampleTime','wSampleTime');

myopts=simset('SrcWorkspace','current','DstWorkspace','current');

sim(model,wSimulationTime,myopts);
while (strcmp(get_param(model,'SimulationStatus'),'stopped')==0);
end

t_sim = toc;
fprintf('\nTemps de simulation => %3.3g s\n',t_sim)

%Post traitement
load(wSaveFileName);
wStruct = eval(wSaveFileName);

%Plots
h_zod=figure();
hold all
plot(Ymodel.Tutsin_explicit_error.Time,...
    Ymodel.Tutsin_explicit_error.Data);

plot(Ymodel.Tutsin_implicit_error.Time,...
    Ymodel.Tutsin_implicit_error.Data);

plot(Ymodel.Halijak_error.Time,...
    Ymodel.Halijak_error.Data);

legend('Error between delayed Tutsin and continuous model',...
    'Error between implicit Tutsin and continuous model',...
    'Error between Halijak and continuous model');
grid minor;
xlabel('Time (s)');
ylabel('Output error');
title('Difference between discretized systems and continous model');

h_zod2=figure();
hold all
plot(Ymodel.Continuous.Time,Ymodel.Continuous.Data);
stairs(Ymodel.Tutsin_explicit.Time,...
    Ymodel.Tutsin_explicit.Data);
```

```
stairs(Ymodel.Tutsin_implicit.Time,...
    Ymodel.Tutsin_implicit.Data);

stairs(Ymodel.Halijak.Time,...
    Ymodel.Halijak.Data);

legend('Continuous model',...
    'Tutsin explicit',...
    'Tutsin implicit',...
    'Halijak');
grid minor;
xlabel('Time (s)');
ylabel('System response');
title('Outputs of continous system and discretized systems');

h_zod3=figure();
hold all
plot(Ymodel.Tutsin_implicit_effect.Time,...
    Ymodel.Tutsin_implicit_effect.Data);

legend('Difference between implicit and delayed Tutsin');
grid minor;
xlabel('Time (s)');
ylabel('Errors difference');
title('Difference between Tutsin output and Delayed Tutsin output');

h_zod4=figure();
hold all
plot(Ymodel.Continuous_command.Time,Ymodel.Continuous_command.Data);
plot(Ymodel.Tutsin_implicit_command.Time,...
    Ymodel.Tutsin_implicit_command.Data);

plot(Ymodel.Tutsin_explicit_command.Time,...
    Ymodel.Tutsin_explicit_command.Data);

plot(Ymodel.Halijak_command.Time,...
    Ymodel.Halijak_command.Data);

legend('Command from continuous model',...
    'Command from Tutsin',...
    'Command from delayed Tutsin',...
    'Command from Halijak');
grid minor;
xlabel('Time (s)');
ylabel('Command signals');
title('Commands of continous system and discretized systems');

%Saving figures
wPaperPos = [0 0 5 5];
wPaperSize = [5 5];

set(h_zod, 'PaperPosition', wPaperPos);
set(h_zod, 'PaperSize', wPaperSize);
saveas(h_zod,'Helico-Errors','pdf')
```

```
set(h_zod2, 'PaperPosition', wPaperPos);
set(h_zod2, 'PaperSize', wPaperSize);
saveas(h_zod2,'Helico-Step Response','pdf')

set(h_zod3, 'PaperPosition', wPaperPos);
set(h_zod3, 'PaperSize', wPaperSize);
saveas(h_zod3,...
    'Helico-Difference between tutsin implicit and explicit',...
    'pdf')

set(h_zod4, 'PaperPosition', wPaperPos);
set(h_zod4, 'PaperSize', wPaperSize);
saveas(h_zod4,'Helico-Commands','pdf')

%Saving model
saveas(get_param(model,'Handle'),'Helico-Model.pdf');
saveas(get_param(strcat(model,'/Tutsin implicit'),'Handle'),...
    'Helico-Model Tutsin.pdf');
saveas(get_param(strcat(model,'/Tutsin explicit'),'Handle'),...
    'Helico-Model Tutsin delayed.pdf');
close_system(model,false);


%Print the transfer functions on command windows for saving:
clc;
disp('Continuous transfer function')
disp('System:')
wSystem.mGetTf
disp('Compensator:')
wCompensator.mGetTf
disp('Closed Loop:')
wSystem.mGetClosedLoop(wCompensator.mGetTf,'continuous')

disp('Tutsin discretization')
disp('System:')
wSystem.mGetDiscreteTf('tutsin')
disp('Compensator:')
wCompensator.mGetDiscreteTf('tutsin')
disp('Closed Loop:')
wSystem.mGetClosedLoop(wCompensator.mGetTf,'tutsin')

disp('Tutsin discretization with delay')
disp('System:')
wSystem.mGetRetardedDiscreteTf('tutsin',1)
disp('Compensator:')
wCompensator.mGetRetardedDiscreteTf('tutsin',1)
disp('Closed Loop:')
wSystem.mGetClosedLoop(...
    wCompensator.mGetRetardedDiscreteTf('tutsin',1),...
    'tutsin')

disp('Halijak discretization')
disp('System:')
wSystem.mGetDiscreteTf('halijak')
disp('Compensator:')
```

```
wCompensator.mGetDiscreteTf('halijak')
disp('Closed Loop:')
wSystem.mGetClosedLoop(wCompensator.mGetTf,'halijak')
```

2. Exercice2 : Code d'exécution

```
addpath(genpath('..'));

%Parametres
close all;
clear all; %#ok<CLSCR>
clc;

wSampleTime=0.1;
wSimulationTime=10;
wMaxStep=wSampleTime/1000;

wInputSignal=ones(1,wSimulationTime/wSampleTime);

wContinuousSystemNum = 100;
wContinuousSystemDen = [1,11,30,200];

wDiscretizer = Discretizer(wSampleTime,...
    wContinuousSystemNum,...
    wContinuousSystemDen);

% Getting recursive equation and Tf for Zero Order Hold:
wYzod = wDiscretizer.mComputeRecursion(wInputSignal,'zoh');
[wYzodN,wYzodD] = wDiscretizer.mGetDiscreteTf('zoh');

% Getting recursive equation and Tf for Tutsin:
wYtut = wDiscretizer.mComputeRecursion(wInputSignal,'tutsin');
[wYtutN,wYtutD] = wDiscretizer.mGetDiscreteTf('tutsin');

% Getting recursive equation and Tf for Halijak:
wYhal = wDiscretizer.mComputeRecursion(wInputSignal,'halijak');
[wYhalN,wYhalD] = wDiscretizer.mGetDiscreteTf('halijak');

% Getting recursive equation and Tf for Boxer Thaler:
wYbox = wDiscretizer.mComputeRecursion(wInputSignal,'boxerThaler');
[wYboxN,wYboxD] = wDiscretizer.mGetDiscreteTf('boxerThaler');

%Parametrage simulation
model='zodModel';
load_system(model)
tic

wSaveFileName     = 'Ymodel';

set_param(model,'StopFcn','save(wSaveFileName,wSaveFileName)');
set_param(strcat(model,'/Output'),'VariableName',wSaveFileName);

set_param(strcat(model,'/ZOD'),'Numerator','wYzodN');
set_param(strcat(model,'/ZOD'),'Denominator','wYzodD');
set_param(strcat(model,'/ZOD'),'SampleTime','wSampleTime');

set_param(strcat(model,'/Tutsin'),'Numerator','wYtutN');
set_param(strcat(model,'/Tutsin'),'Denominator','wYtutD');
```

```
set_param(strcat(model,'/Tutsin'),'SampleTime','wSampleTime');

set_param(strcat(model,'/Halikaj'),'Numerator','wYhalN');
set_param(strcat(model,'/Halikaj'),'Denominator','wYhalD');
set_param(strcat(model,'/Halikaj'),'SampleTime','wSampleTime');

set_param(strcat(model,'/Boxer Thalor'),'Numerator','wYboxN');
set_param(strcat(model,'/Boxer Thalor'),'Denominator','wYboxD');
set_param(strcat(model,'/Boxer Thalor'),'SampleTime','wSampleTime');

set_param(strcat(model,'/Continuous'),...
    'Numerator','wContinuousSystemNum');
set_param(strcat(model,'/Continuous'),...
    'Denominator','wContinuousSystemDen');

set_param(model, 'StopTime', 'wSimulationTime');

set_param(model, 'MaxStep', 'wMaxStep');

myopts=simset('SrcWorkspace','current','DstWorkspace','current');

sim(model,wSimulationTime,myopts);
while (strcmp(get_param(model,'SimulationStatus'),'stopped')==0);
end

t_sim = toc;
fprintf('\nTemps de simulation => %3.3g s\n',t_sim)

%Post traitement
load(wSaveFileName);
wStruct = eval('wSaveFileName');

%Plots
h_zod=figure();
hold all
plot(Ymodel.Yzod.Time,Ymodel.Yzod.Data);
stairs(Ymodel.Yzod.Time,Ymodel.Yzod.Data);
stairs(0:wSampleTime:(size(wInputSignal,2)-1)*wSampleTime,wYzod);
legend('Simulink ZOH (plot)',...
    'Simulink ZOH (stairs)',...
    'Recursive equation ZOH');
grid minor;
xlabel('Time (s)');
ylabel('Step Response');
title('Open Loop Response to ZOH discretized function transfer');

h_tut=figure();
hold all
plot(Ymodel.Ytut.Time,Ymodel.Ytut.Data);
stairs(Ymodel.Ytut.Time,Ymodel.Ytut.Data);
stairs(0:wSampleTime:(size(wInputSignal,2)-1)*wSampleTime,wYtut);
plot(Ymodel.Ys.Time,Ymodel.Ys.Data);
legend('Simulink Tutsin (plot)',...
    'Simulink Tutsin (stairs)',...
    'Recursive equation Tutsin',...
```

```
    'Continuous system');
grid minor;
xlabel('Time (s)');
ylabel('Step Response');
title('Open Loop Response to Tutsin discretized function transfer');

h_hal=figure();
hold all
plot(Ymodel.Yhal.Time,Ymodel.Yhal.Data);
stairs(Ymodel.Yhal.Time,Ymodel.Yhal.Data);
stairs(0:wSampleTime:(size(wInputSignal,2)-1)*wSampleTime,wYhal);
plot(Ymodel.Ys.Time,Ymodel.Ys.Data);
legend('Simulink Halijak (plot)',...
    'Simulink Halijak (stairs)',...
    'Recursive equation Halijak','Continuous system');
grid minor;
xlabel('Time (s)');
ylabel('Step Response');
title('Open Loop Response to Halijak discretized function transfer');

h_box=figure();
hold all
plot(Ymodel.Ybox.Time,Ymodel.Ybox.Data);
stairs(Ymodel.Ybox.Time,Ymodel.Ybox.Data);
stairs(0:wSampleTime:(size(wInputSignal,2)-1)*wSampleTime,wYbox);
plot(Ymodel.Ys.Time,Ymodel.Ys.Data);
legend('Simulink Boxer-Thaler (plot)',...
    'Simulink Boxer-Thaler (stairs)',...
    'Recursive equation Boxer-Thaler','Continuous system');
grid minor;
xlabel('Time (s)');
ylabel('Step Response');
title('Open Loop Response to Boxer-Thaler discretized function transfer');

%Poles - zeros
h_spz = figure();
pzmap(wDiscretizer.mGetTf());
legend('Continuous system');
sgrid;
grid minor;

h_zpz  = figure();
pzmap(wDiscretizer.mGetDiscreteTf('zoh'),...
    wDiscretizer.mGetDiscreteTf('tutsin'),...
    wDiscretizer.mGetDiscreteTf('halijak'),...
    wDiscretizer.mGetDiscreteTf('boxerThaler'));
legend('Zero order hold','Tutsin','Halijak','Boxer-Thaler');
zgrid;
grid minor;

h_snyq = figure();
nyquist(wDiscretizer.mGetTf());
legend('Continuous system');
grid minor;
```

```
%Saving figures
wPaperPos = [0 0 8 5];
wPaperSize = [8 5];

set(h_zod, 'PaperPosition', wPaperPos);
set(h_zod, 'PaperSize', wPaperSize);
saveas(h_zod, 'ZTransform ZoH', 'pdf')

set(h_tut, 'PaperPosition', wPaperPos);
set(h_tut, 'PaperSize', wPaperSize);
saveas(h_tut, 'ZTransform Tutsin', 'pdf')

set(h_hal, 'PaperPosition', wPaperPos);
set(h_hal, 'PaperSize', wPaperSize);
saveas(h_hal, 'ZTransform Halijak', 'pdf')

set(h_box, 'PaperPosition', wPaperPos);
set(h_box, 'PaperSize', wPaperSize);
saveas(h_box, 'ZTransform Boxer-Thalor', 'pdf')

set(h_spz, 'PaperPosition', wPaperPos);
set(h_spz, 'PaperSize', wPaperSize);
saveas(h_spz, 'ZTransform Poles-Zeros', 'pdf')

set(h_zpz, 'PaperPosition', wPaperPos);
set(h_zpz, 'PaperSize', wPaperSize);
saveas(h_zpz, 'ZTransform Poles-Zeros discretes', 'pdf')

set(h_snyq, 'PaperPosition', wPaperPos);
set(h_snyq, 'PaperSize', wPaperSize);
saveas(h_snyq, 'ZTransform Nyquist', 'pdf')

%Saving model
saveas(get_param(model,'Handle'),'ZTransform-Model.pdf');
close_system(model,false)

%Print the transfer functions on command windows for saving:
clc;
disp('Continuous transfer function')
wDiscretizer.mGetTf

disp('Tutsin discretization')
wDiscretizer.mGetDiscreteTf('tutsin')

disp('Boxer-Thalor discretization')
wDiscretizer.mGetDiscreteTf('boxerThaler')

disp('Halijak discretization')
wDiscretizer.mGetDiscreteTf('halijak')
```

3. Général : Discretizer

```
classdef Discretizer < handle

    properties (SetAccess = private, GetAccess = private)

        mT;
        mContinuousTf;

        %Matrices de transfer:
        mQHalijak;
        mQBoxer;

    end

    methods %Public

        %Constructors
        function oInstance = Discretizer(iT, varargin)
            if nargin == 2
                oInstance.mContinuousTf = varargin{1};
            elseif nargin == 3
                oInstance.mContinuousTf = tf(varargin{1},varargin{2});
            else
                oInstance.mContinuousTf = 0;
                h = errordlg('Invalid constructor');
                waitfor(h);
            end

            oInstance.mSetSampleTime(iT);

        end

        %Public methods

        %Accessors

        function mSetSampleTime(iThis,iSampleTime)
            iThis.mT = iSampleTime;
            mUpdateMatrix(iThis);
        end

        function oSampleTime = mGetSampleTime(iThis)
            oSampleTime = iThis.mT;
        end

        function oMatrix = mGetHalijakMatrix(iThis,iRank)
            oMatrix = iThis.mQHalijak{iRank};
        end

        function oMatrix = mGetBoxerThalerMatrix(iThis,iRank)
            oMatrix = iThis.mQBoxer{iRank};
        end
```

```
function oTf = mGetTf(iThis)
    oTf = iThis.mContinuousTf;
end

%Compute discrete TF
function varargout = mGetDiscreteTf(iThis,iType)

    wDiscreteTf = mProcessTf(iThis,iThis.mContinuousTf,iType);

    if (nargout == 0) || (nargout == 1)
        varargout{1} = wDiscreteTf;
    elseif nargout == 2
        [wHnum,wHden] = tfdata(wDiscreteTf,'v');
        varargout{1} = wHnum;
        varargout{2} = wHden;
    else
        h = errordlg('Invalid number of output arguments, valids outputs ar
        waitfor(h)
    end

end

%Compute closed loop TF.
function varargout = mGetClosedLoop(iThis,iFeedBackTf,iType)

    wDiscreteTf = mProcessTf(iThis,iThis.mContinuousTf,iType);
    wDiscreteFeedBackTf = mProcessTf(iThis,iFeedBackTf,iType);

    wCLTF = feedback(wDiscreteTf,wDiscreteFeedBackTf);

    if (nargout == 0) || (nargout == 1)
        varargout{1} = wCLTF;
    elseif nargout == 2
        [wHnum,wHden] = tfdata(wCLTF,'v');
        varargout{1} = wHnum;
        varargout{2} = wHden;
    else
        h = errordlg('Invalid number of output arguments, valids outputs ar
        waitfor(h)
    end

end

%Compute discrete TF and apply retard
function varargout = mGetRetardedDiscreteTf(iThis,iType,iRetard)

    wHnum    = 0; %#ok<NASGU>
    wHden    = 0; %#ok<NASGU>
    wRetard  = ones(1,iRetard);

    [wHnum,wHden] = mGetDiscreteTf(iThis,iType);
    wHden         = [wRetard,wHden];

    if (nargout == 0) || (nargout == 1)
        varargout{1} = tf(wHnum,wHden,iThis.mT);
```

```
            elseif nargout == 2
                varargout{1} = wHnum;
                varargout{2} = wHden;
            else
                h = errordlg('Invalid number of output arguments, valids outputs ar
                waitfor(h)
            end
        end

        %Execute recursion equation with the specified input, on the
        %specified typed discrete function.
        function Y = mComputeRecursion(iThis,U,iType)

            [wNum,wDen] = iThis.mGetDiscreteTf(iType);
            Y = iThis.mProcessRecursion(wNum,wDen,U);

        end

    end %Public methods

    %Private methods
    methods (Access = private)

        %Updates all matrixes when requested
        function mUpdateMatrix(iThis)
            mUpdateBoxerThalerMatrix(iThis);
            mUpdateHalijakMatrix(iThis);
        end

        %Halijak substitution matrix
        function mUpdateHalijakMatrix(iThis)

            mQ1=...
                [iThis.mT,  0;...
                1, -1];

            mQ2=...
                [0, iThis.mT^2, 0;...
                iThis.mT, -iThis.mT,  0;...
                1,  -2,  1];

            mQ3=...
                [0,  iThis.mT^3/2, iThis.mT^3/2, 0;...
                0,   iThis.mT^2,  -iThis.mT^2,   0;...
                iThis.mT, -2*iThis.mT,   iThis.mT,     0;...
                1,  -3,      3,     -1];

            mQ4 =...
                [0,  iThis.mT^4/4,  2*iThis.mT^4/4,  iThis.mT^4/4 , 0;...
                0,   iThis.mT^3/2,  0       , -iThis.mT^3/2 , 0;...
                0,   iThis.mT^2  , -2*iThis.mT^2  ,  iThis.mT^2   , 0;...
                iThis.mT, -3*iThis.mT  ,  3*iThis.mT    , -iThis.mT
, 0;...
                1,  -4    ,   6       , -4      , 1];
```

```
            iThis.mQHalijak = {mQ1, mQ2, mQ3, mQ4};

        end

        %Boxer Thalor substitution matrix
        function mUpdateBoxerThalerMatrix(iThis)

            mQ1=...
                [iThis.mT/2,  iThis.mT/2;...
                1          , -1];

            mQ2=...
                [iThis.mT^2/12, 10*iThis.mT^2/12, iThis.mT^2/12;...
                iThis.mT/2    , 0               , -iThis.mT/2;...
                1,             -2               ,  1];

            mQ3=...
                [0            ,    iThis.mT^3/2 ,    iThis.mT^3/2 ,  0;...
                iThis.mT^2/12,  9*iThis.mT^2/12, -9*iThis.mT^2/12, -iThis.mT^2/12;.
                iThis.mT/2    , -1*iThis.mT/2   , -1*iThis.mT/2   ,  iThis.mT/2;...
                1             , -3              ,   3             , -1];

            mQ4 =...
                [-iThis.mT^4/720,  124*iThis.mT^4/720,  474*iThis.mT^4/720,
124*iThis.mT^4/720, -iThis.mT^4/720;...
                0               ,   iThis.mT^3/2     ,  0
, -iThis.mT^3/2       , 0;...
                iThis.mT^2/12   ,   8*iThis.mT^2/12  , -18*iThis.mT^2/12
,  8*iThis.mT^2/12    , iThis.mT^2/12;...
                iThis.mT/2      , -iThis.mT          , 0
,  iThis.mT           , -iThis.mT/2;...
                1               ,  -4                ,   6
, -4                   , 1];

            iThis.mQBoxer = {mQ1, mQ2, mQ3, mQ4};

        end

        %Execute recursion equation with the specified input.
        function Y = mProcessRecursion(iThis,num,den,U)

            iThis; %#ok<VUNUS>

            wTrimedDen = den(find(den,1):size(den,2));
            wTrimedNum = num(find(num,1):size(num,2));

            %How many iterations of Y are not computable.
            %If the system is implicit, then wUdelta = 0
            wUdelta = size(wTrimedDen,2)-size(wTrimedNum,2);
            if (wUdelta < 0)
                h = errordlg('Error, non-causal system');
                waitfor(h);
                return;
            end
```

```
        Y = zeros(1,wUdelta);

        for i = size(Y,2)+1:size(U,2)

            wY = 0;
            wU = 0;

            %Building input sum according to matlab 1-based indexing
            for k = 1:size(wTrimedNum,2)
                if(i-k+1-wUdelta > 0)
                    wU = wU + wTrimedNum(k)*U(i-k+1-wUdelta);
                else
                    wU = wU + 0;
                end
            end

            %Building output sum according to matlab 1-based indexing
            for k = 1:size(wTrimedDen,2)-1
                if((i-k > 0) && (i-k <= size(Y,2)))
                    wY = wY + wTrimedDen(k+1)*Y(i-k);
                else
                    wY = wY + 0;
                end
            end

            %Sum must be pondered by the highest numerator coefficient
            Y(i) = 1/wTrimedDen(1) * (-wY + wU);

        end
    end

    %Process TF conversions.
    function oTf = mProcessTf(iThis,iTf,iType)

        wTf      = iTf;
        wHnum    = 0; %#ok<NASGU>
        wHden    = 0; %#ok<NASGU>

        if (strcmp(get(iTf,'Variable'),'s'))
            switch (iType)
                case 'zoh'

                    wTf = c2d(iTf,iThis.mT,'zoh');

                case 'tutsin'

                    wTf = c2d(iTf,iThis.mT,'tutsin');

                case {'halijak','boxerThaler'}

                    [wHnum,wHden] = tfdata(iTf,'v');

                    if(strcmp(iType,'halijak'))
                        wH = [fliplr(wHnum);fliplr(wHden)]*iThis.mGetHalijakMat
                    else
```

```
                        wH = [fliplr(wHnum);fliplr(wHden)]*iThis.mGetBoxerThale
                    end

                    wTf = tf(wH(1,:),wH(2,:),iThis.mT);

                case 'continuous'
                    %Return input

                otherwise
                    warning('mProcessTf returning input (continuous) TF')
            end

        else
            switch (iType)
                case 'zoh'

                    wTf = d2d(iTf,iThis.mT,'zoh');

                case 'tutsin'

                    wTf = d2d(iTf,iThis.mT,'tutsin');

                otherwise

                    warning('mProcessTf returning input (discrete) TF')
            end

        end

        oTf = wTf;

    end
end %Private methods

end %Class
```

——— Fin du devoir ———