ADERSA

MELISSA

Memorandum of Understanding
ECT/FG/MMM/97.012

ESTEC/Contract N° 12924/98/NL/MV
Contract change notice No 04 of August 2002

TECHNICAL NOTE : 73.3

Nitrite control of the Nitrifying compartment

AUTOMATIC CONVERSION OF THE SOFTWARE INTO C LANGAGE

Version : 1
Issue : 0

J.-J. LECLERCQ

June 2003

10, rue de la Croix Martre
91873 PALAISEAU Cedex
Téléphone : (33) 1 60 13 53 53
Télécopie : (33) 1 69 20 05 63
Courriel : adersa@adersa.com

adersa

Document Change Log

Version	Issue	Date	Observation
0	0	May 2003	Draft
1	0	June 2003	Original version

TABLE OF CONTENTS

Abbreviations or notations:
PC : Personal Computer
PLC : Programmable Logical Computer
CST : Completely Stirred Tank
DLL : Dynamic Link Library

1. INTRODUCTION

The state system of the internal model of the control is identical to the one of the simulated process, except the fact that the fixed bed of the model is considered as only one CST (Completely Stirred Tank) instead of a series of several CST's.
So, at the beginning of the study related to the control (elaboration, tuning of performance and robustness), it seemed natural to re-use the algorithm of the process itself and to apply it to the control. All the study was done with a controller written with Matlab® code as it was easier that way to modify the code for comparing several approaches.
It was foreseen that, at the end and before its implementation in the computer of the pilot plant at UAB, the Matlab® code will be translated automatically into C language, using the tool '*Matlab Compiler*'.
According to the Matlab® documentation itself: "The Matlab Compiler automatically converts Matlab M-files to C and C++ code. The Matlab Compiler includes the Matlab C/C++ Math and Graphics Libraries, which allows the conversion of Matlab applications to C and C++ code for stand-alone applications".
This tool has been tested, before the nitrite controller was ready, on another application of the MELISSA project : the global simulator as it is achieved in the TN's 54.4 and 64.1 .

2. DLL OR STAND-ALONE MATLAB APPLICATION

2.1. Introduction

The DLL (mex-file) or stand-alone mode allows to use the Matlab functionalities (scientific computation, graphics …) and to furnish to users an application independent of Matlab.
A mex-file and a stand-alone application differ in these respects :

- a mex-file runs in the same process space as the Matlab interpreter. When a mex-file is invoked, the Matlab interpreter dynamically links in the mex-file.
- a stand-alone C or C++ application runs independently of Matlab.

In the present study, the test is done with a mex-file considering that the controller is connected with a process simulated with Matlab code.

The different steps of development of a stand-alone or DLL application follow (figure 1) :

- Elaboration of the application with Matlab language : writing of the Matlab functions in m-file only (Matlab scripts are forbidden);
- C code generation of each of the functions built in the previous step : this work is realized by the Matlab compiler;
- Compiling of the C functions (ANSI compiler)
- Linking with the Matlab libraries.

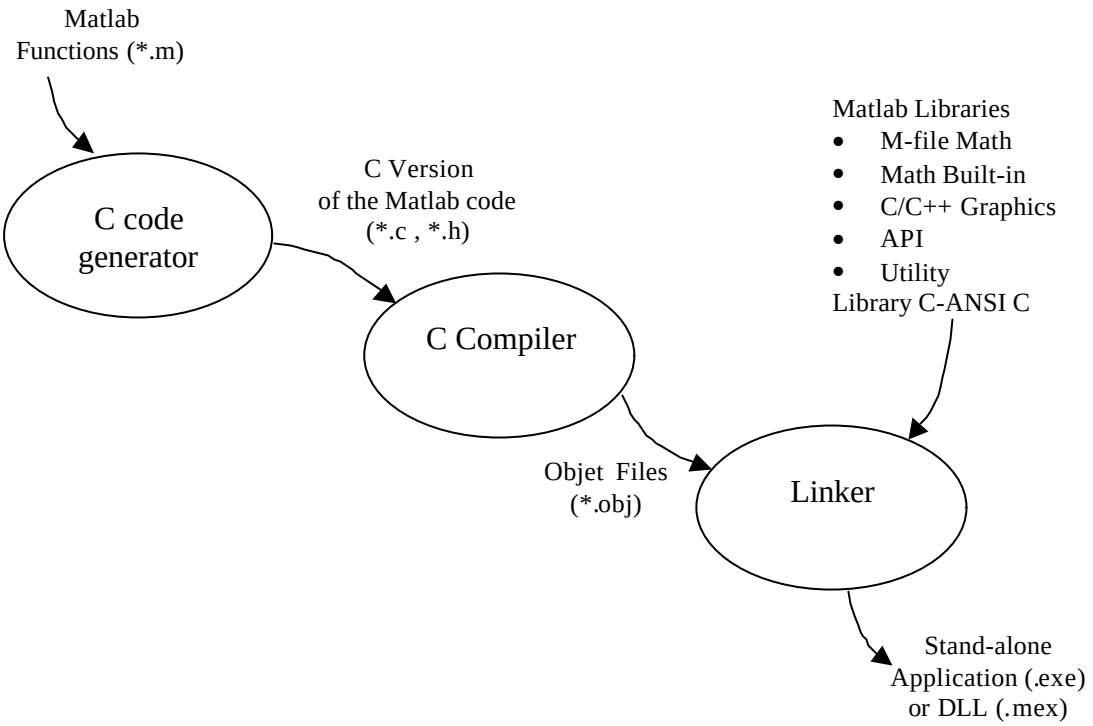

Figure 1 : Development of a stand-alone or DLL application

2.2. Limitations

2.2.1. Compiler

The version of the Matlab Compiler supports nearly all the functionalities of Matlab. Nevertheless there are some limitations. The following Matlab cannot be compiled :

- Matlab scripts;
- Matlab files using objects;
- Matlab files using functions *input* or *eval*. However, when these functions do not use variables of the Work Space, they can be compiled.

2.2.1. Stand-alone application

The previous limitations are still true for stand-alone applications. Moreover the following functions of Table 1 are not supported.

add_block	add_line	applescript	assignin
callstats	close_system	cputime	dbclear
dbcont	dbdown	dbquit	dbstack
dbstatus	dbstep	dbstop	dbtype
dbup	delete_block	delete_line	diary
echo	edt	errorstat	errortrap
evalin	fields	fschange	functionscalled
get_param	hcreate	help	home
hregister	inferiorto	inmem	isglobal
isjava	isruntime	java	javaArray
javaMethod	javaObject	keyboard	linmod
lookfor	macprint	mactools	methods
mislocked	mlock	more	munlock

new_system	open_system	pack	pfile
rehash	runtime	set_param	sim
simget	simset	sldebug	str2func
superiorto	system_dependent	trmginput	type
vms	what	which	who
whos			

Table 1 : Functions non supported in stand-alone mode (from Matlab documentation)

2.2.3. Loading and saving files

In the stand-alone mode, the functions *load* and *save* do not work when the variables of the file are defined. Particularly the specific loading does not work.

2.3. Compiling

When an application involves several functions, only the main function has to be declared to the compiler : the called functions will be compiled and linked automatically.
The following instructions are for C language. Analogue instructions exist for C++.

Building a mex-file :
To build the C code of the Matlab function *TheApplication.m* and to create the associated mex-file (that has to be used with Matlab) :

mcc –x TheApplication

Building a stand-alone application :
To build the C code of the Matlab function *TheApplication.m* and to create the associated stand-alone application (that can run without Matlab) :

mcc –m TheApplication

Building a stand-alone application with graphic:
To build the C code of the Matlab function *TheApplication.m* which uses graphic functions and to create the associated stand-alone application (that can run without Matlab) :

mcc –B sgl TheApplication

3. EXAMPLE OF APPLICAT ION

3.1. Description of the application

As said in section 1, the Matlab Compiler is tested on the global simulator as it is achieved in the TN's 54.4 and 64.1 .
The programme of this global simulator is composed of 3 parts :

- a part for initializing of the parameters and arrays;
- another part composed of a loop calling the simulation of the process composed of the 5 compartments of the MELISSA project;
- and a third part of various statements (data process for plotting …).

The programme can be represented by the following statements of a Matlab script file (fig. 2).

```
InitializingStatements % Initialization of the programme
for i=1:n
  [OutputArguments]=simpro(InputArguments); % Process
  FurtherStatements1 % Complementary instructions group 1
end
FurtherStatements2 % Complementary instructions group 2
```

Fig. 2 : Condensed form of the Matlab programme of the simulator

As said in section 2, the script file of the fig.2 cannot be converted with the Matlab Compiler. But the function *simpro* can be.
The function *simpro* calls other various functions (Table 2) from Matlab library or not. Data are common to *simpro* and the Matlab Workspace by means of *global* statements.

Name	Function
simpro	: main function (to be translated in C language)
| -stesta_2	: steady state of the Rhodo compartment
| | -rx_rhodo	: growth rate of Rhodo biomass
| | | -cosh,sinh	: mathematical Matlab functions
| | -dich2_rh	: dichotomy algorithm
| | -pr_rhodo	: Rhodo process
| | -sqrt	: mathematical Matlab function (square root)
| -stesta_3	: steady state of the Nitri compartment
| | -pr_nitri	: Nitri process
| | | -irate	: instantaneous rate
| | | -reshape	: Matlab function
| -stesta_4	: steady state of the Spiru compartment
| | -rx_spiru	: growth rate of Spiru biomass
| -stesta_5	: steady state of the Higher Plant compartment

Table 2 : Sub-functions of *simpro*

3.2. Results

The Matlab Compiler converts each functions of the table 2 into corresponding '*.h' and '*.c' files : an example of conversion is given in annex. The conversion transforms the programme in such a way that it is no more understandable by a programmer. That could be considered redhibitory in the present study because if a problem occurs during execution on the target machine, it becomes nearly impossible to discover its origin, particularly if it comes from a Matlab function whose programme is unknown and inaccessible.
Then these files are compiled and linked to produce the mex-file of the function 'simpro'. No particular problems is encountered during that step.
The resulting application gives the same results as the original Matlab one.
As the resulting application is executable (and no more interpretable), a gain of running time was expected. That is confirmed but the gain is low (only 17 %) as reported in table 3.

Run number	Running time (s)		Gain (%)
	Compiled programme (mex)	Interpreted programme (Matlab)	
1	0.471	0.601	22
2	0.440	0.541	19
3	0.441	0.510	15
4	0.441	0.530	17
5	0.440	0.541	19
6	0.441	0.531	17
7	0.410	0.531	23
8	0.491	0.531	8
9	0.441	0.531	17
10	0.440	0.531	17
Mean	**0.4437**	**0.534**	**17**

Table 3 : Comparison of running times of the 2 modes (mex and Matlab)

4. CONCLUSION

The Matlab Compiler has been tested on an application considered as representative of the controller (as that one was not ready at the moment of the test). Although the application is compiled instead of interpreted, the gain of running time is low.

The most interesting result is that the conversion transforms the original programme in such a way that it is no more understandable by a programmer. Moreover the Matlab functions (integration algorithms particularly) are inaccessible. Then if a problem occurs during the running on the target PLC, it would be impossible to analyze it. It is why the automatic conversion by means of the Matlab Compiler has been declared redhibitory in the case of the present study.

The conversion of the Matlab code of the nitrite controller will be done manually (step by step) in the TN 73.1 .

5. REFERENCE

Matlab Compiler User's Guide. The MathWorks, Inc. September 2000. Revised for Version 2.1 (Release 12).

6. ANNEX : DETAILS OF THE APPLICATION EXAMPLE

6.1. Example of an original function written with Matlab langage

Illustration of the function *simpro* (written with Matlab language) to be converted by the Matlab Compiler :

```
%**************************************************************************
%       Connected compartments 0,1,2,3,4A and 4B (numbered 5 here)      *
%       Version 3.2     September 2002                                 *
%                                                                      *
%       simpro.m   Simulation of the MELISSA loop (HPC included)       *
%                  Open loop and Steady state                          *
%                                                                      *
%**************************************************************************
function [CG0_0, Y0_0, Y0_1, CG0_2, CL0_2, FR0_2, X0_2, Y0_2, ...
      CG0_3, CL0_3, X0_3, Y0_3, dX0_3, CG0_4, CL0_4, FR0_4, X0_4, Y0_4, ...
      CG0_5, CL0_5, Y0_5, ri_5] = ...
  simpro(EE, fm_O2_0, fm_CO2_0, rO2_0, rCO2_0, initNH3, initNO2, initAcOH, initBuOH, ...
         initLiqueL, addCO2, addO2, Yx_AcOH, YO2_5, YCO2_5, spNO3_5, share2, cover)

global VM_0  Gin_0
global Gin_1  Fin_1  NG1_1  NG_1  NL_1  Kdis_1  rKdis_1  Yp_1p
global Gin_2  Fin_2  NG_2  NL_2  NO_2  NS_2  NI_2  NX_2  vNB_2  vNM_2  vNS_2  ...
   VL_2  Yx1_2  Kdis_2  KSSO4_2  A2_2  B2_2  GG1_2  GG2_2  ...
   FRmin_2  FRmax_2  fI_2  RT_2  Ea_2  Es_2  muM_2  KJ_2  EpsJ_2  q_rhod_2  zmin_I_2
global Gin_3  Fin_3  NL_3  NG_3  NS_3  NB_3  NX_3  NO_3  NI_3  NV_3 ...
   Ae_3  Be_3  Ce_3  De_3  E_3  WX_3  WYG_3  WYL_3 ...
   iO2_3  iCO2_3  iNH3_3  iNO2_3  iSub_3  iXNs_3  iXNb_3  iXag_3  ...
   KlNs_3  KlNb_3  KmNs_3  KmNb_3  mumax_3  maint_3  Yx_3  Yx1_3  Ym1_3  ...
   RL_3  indG_3  indL_3  Kdis_3
global Fin_4  NG_4  NL_4  NO_4  vNB_4  vNM_4  VL_4  Yx1_4  Kdis_4  ...
   A2_4  B2_4  GG1_4  GG2_4  ...
   FRmin_4  FRmax_4  FRmaxc_4  fI_4  zPC_4  zCH_4  ...
   RT_4  Ea_4  Es_4  muM_4  muEPS_4  Kj_4  KjEPS_4  Fmin_4  zmin_I_4  ...
   KSNO3_4  KSSO4_4  KSPO4_4  KSPC_4
global Gin_5  Fin_5  NG_5  NL_5  NS_5  VG_5  VL_5  Yx1e_5  Yx1w_5  Kdis_5  ...
   Diete_5  Dietw_5  A2_5  B2_5  ind_3u5
global idisp

%>>> CONSU
% Concentrations in the incoming Gas of Consu
% ===========================================
  CG0_0 = [fm_O2_0; fm_CO2_0]/VM_0;  % mol/l of O2 CO2 going into Consu
% Initial output vector of Consu (static state)
% =============================================
  Y0_0 = CG0_0 + [rO2_0; rCO2_0]/Gin_0;

%>>> LIQUE
% Concentrations in the incoming Gas and Liquid flows of Lique
% ============================================================
  %1. Gas phase : inert gas
  %2. Liquid phase :
  %cFaec_0 = qFaec_0/Fin_1; % mol/l
  %cUrea_0 = qUrea_0/Fin_1; % mol/l
  %cSO4_0  = qSO4_0/Fin_1;  % mol/l
  %cPO4_0  = qPO4_0/Fin_1;  % mol/l
% Output vector of Lique (static state)
% =====================================
  Y_1 = EE(2)*Yp_1p; % (mol/h): production rate of H2 CO2 NH3 AcOH BuOH
  NY1= NG1_1+2*NG_1;
  NY = NY1+NL_1;
  % Gas concentration in mol/l (H2 and CO2 are totaly under Gas phase)
  Y0_1 = zeros(NY,1);
  Y0_1(1:NG1_1) = Y_1(1:NG1_1)/Gin_1;  % output conc. of H2 in Gas (mol/l)
```

```
  % Liquid concentration in mol/l (CO2 NH3 AcOH & BuOH are totaly in Liquid phase)
  Y0_1(NY1+[1:4]) = Y_1(NY1+[1:4])/Fin_1./(1+Kdis_1(1:4)); % conc. in mol/l of CO2 NH3 molec.
form
  Y0_1(NY1+5:NY) = initLiqueL;   % conc. SO4 PO4 in mol/l

%>>> RHODO
% Initial concentrations in the incoming Gas and Liquid flows of Rhodo
% ====================================================================
  %1. Gas phase (loop of inert gas) :
  fm_CO2_2 = 0;   % no CO2  (molar fraction)
  fm_NH3_2 = 0;   % no NH3  (molar fraction)
  fm_Ac_2  = 0;   % no AcOH (molar fraction)
  fm_Bu_2  = 0;   % no BuOH (molar fraction)
  CG0_2 = [fm_CO2_2; fm_NH3_2; fm_Ac_2; fm_Bu_2] / VM_0;  % mol/l
  %2. Liquid phase :
  %CL0_2 = [cCO2_0; cNH3_0; cAc_0; cBu_0; cSO4_0; cPO4_0] % mol/l
  ind = [NY1+1:NY];
  CL0_2  = Y0_1(ind).*rKdis_1; % mol/l of the molecular form CO2 NH3 AcOH BuOH
  CL0_2(1:NG_2) = CL0_2(1:NG_2) + ...
       [YCO2_5;    % addition of CO2 coming from HPC output
        initNH3;   % NH3 GL coming from HPC output and going into L phase of Rhodo input
        initAcOH;  % AcOH coming from L Rhodo output
        initBuOH]; % BuOH coming from L Rhodo output
% Initial concentration of biomass in Rhodo (to consume 'Yx_AcOH' part of AcOH load)
% ==================================================================================
  aa = 1 + Yx_AcOH*((1+Kdis_2(3))/(1+Kdis_2(NG_2))*Yx1_2(4,2)/Yx1_2(3,1) - 1);
  cX0_2 = -Yx_AcOH * (1+Kdis_2(3)) / Yx1_2(3,1)* (aa*CL0_2(3)+CL0_2(NG_2)) / aa;
% Initial light flux and state vector of Rhodo (steady state)
% ===========================================================
  %[FR0_2, X0_2, Y0_2] = stesta_2(...
  % NG_2, NL_2, NO_2, vNB_2, vNM_2, VL_2, Fin_2, Yx1_2, KSSO4_2, ...
  % A2_2, B2_2, GG1_2, GG2_2, CG0_2, CL0_2, cX0_2, ...
  % FRmin_2, FRmax_2, fI_2, RT_2, Ea_2, Es_2, muM_2, KJ_2, EpsJ_2, q_rhod_2, zmin_I_2);
  [FR0_2, X0_2, Y0_2, dX0_2] = stesta_2(...
   NG_2, NL_2, NO_2, NS_2, NI_2, NX_2, vNB_2, vNM_2, vNS_2, VL_2, Fin_2, Yx1_2, KSSO4_2, ...
   A2_2, B2_2, GG1_2, GG2_2, CG0_2, CL0_2, cX0_2, ...
   FRmin_2, FRmax_2, fI_2, RT_2, Ea_2, Es_2, muM_2, KJ_2, EpsJ_2, q_rhod_2, zmin_I_2);
  %1. CO2 NH3 in G phase of Rhodo going into L phase of Rhodo :
  X0_2(1:2) = X0_2(1:2) + (Y0_2(1:2)*Gin_2/Fin_2)./(1+Kdis_2(1:2)); % modif of state vector
  %2. Modification of output vector
  Y0_2(1:2) = zeros(2,1);
  Y0_2(NG_2+[1:2]) = X0_2(1:2);                                % modif of output vector

%>>> NITRI
% Concentrations in the incoming Gas and Liquid flows of Nitri
% ============================================================
  %1. Gas phase (O2 CO2 from Consu, O2 used for H2 transformation) :
  CG0_3 = [Y0_0; 0];  % mol/l of O2 CO2 NH3 (NH3:null) from Consu
  CG0_3(1) = CG0_3(1) - Y_1(1)/2/Gin_3; %O2 used for H2 transformation
  %2. Liquid phase (from Rhodo):
  %CL0_3 = [C_O2_0; C_CO2_0; C_NH3_0; C_NO2_0; C_NO3_0; C_PO4_0; C_SO4_0];
  CL0_3 = link2_3(X0_2)';
  CL0_3(1) = YO2_5;        % function of expected set point of O2_L at Spiru output
  CL0_3(NG_3+2) = spNO3_5*Fin_5/Fin_3;  % set point of NO3 at HPC output
  CL0_3(NG_3+1) = CL0_3(NG_3+1) + initNO2;  % NO2 from previous step
% Initial state and output vectors of Nitri (steady state)
% ========================================================
  if idisp, CG0_3,CL0_3,end
  [X0_3, Y0_3, dX0_3] = stesta_3( ...
   NL_3, NG_3, NS_3, NB_3, NX_3, NO_3, NI_3, NV_3, ...
   Ae_3, Be_3, Ce_3, De_3, E_3, WX_3, WYG_3, WYL_3, ...
   iO2_3, iCO2_3, iNH3_3, iNO2_3, iSub_3, iXNs_3, iXNb_3, iXag_3, ...
   KlNs_3, KlNb_3, KmNs_3, KmNb_3, mumax_3, maint_3, Yx_3, Yx1_3, Ym1_3, ...
   CG0_3, CL0_3, RL_3, indG_3, indL_3);
  if isempty(X0_3), break,  end
  if idisp,X0_3p = [X0_3(indL_3);X0_3(indL_3(3)+1);X0_3(indL_3(3)+2)],end
%>>> SPIRU
% Initial concentrations in the incoming Gas and Liquid flows of Spiru
% ====================================================================
  CGL0 = link3_4(Y0_3); % G and L from Nitri
  %1. Gas phase :
  CG0_4 = CGL0(1:NG_4);   % mol/l
```

```
  %2. Liquid phase [O2 CO2 NO3 SO4 PO4]
  CL0_4 = CGL0(NG_4+1:2*NG_4+NL_4);    % mol/l
% Initial light flux and state vector of Spiru (steady state)
% ============================================================
  if idisp, CG0_4,CL0_4,end
  [FR0_4, X0_4, Y0_4] = stesta_4( ...
   NG_4, NL_4, NO_4, vNB_4, vNM_4, Fin_4, VL_4, Yx1_4, ...
   A2_4, B2_4, GG1_4, GG2_4, CG0_4, CL0_4, EE(1), ...
   FRmin_4, FRmaxc_4, fI_4, zPC_4, zCH_4, ...
   RT_4, Ea_4, Es_4, muM_4, muEPS_4, Kj_4, KjEPS_4, Fmin_4, zmin_I_4, ...
   KSNO3_4, KSSO4_4, KSPO4_4, KSPC_4);
  if (isempty(FR0_4) | isempty(X0_4)), break,  end
  if idisp, X0_4, FR0_4, end

%>>> HPC
% Concentrations in the incoming Gas and Liquid flows of HPC
% ============================================================
  %1. Gas phase [O2 CO2 NH3]:
  indN1 = (NB_3+2)*NG_3; % index of NH3 G in output vector of Nitri
  %CG0_5 = [Y0_4(1)+addO2/Gin_5;Y0_4(NG_4)+addCO2/Gin_5;Y0_3(indN1)]; % O2 CO2 NH3
  CG0_5 = [Y0_4(1:NG_4)+[addO2;addCO2]/Gin_5;Y0_3(indN1)]; % O2 CO2 NH3
  %2. Liquid phase [O2 CO2 NH3 NO3 SO4 PO4]
  CL0_5 = link245(X0_2,Y0_3,X0_4,share2,NG_2,Kdis_2,...
                  NG_3,ind_3u5,Kdis_3,NG_4,NL_4,Kdis_4,NG_5,Kdis_5);
% Initial state and variation rates vectors of HPC (steady state)
% ================================================================
  if idisp, CG0_5,CL0_5,end
  [X0_5, ri_5] = stesta_5(...
     NG_5, NL_5, NS_5, VG_5, VL_5, Fin_5, Yx1e_5, Yx1w_5, ...
     Diete_5, Dietw_5, cover, A2_5, B2_5, CG0_5, CL0_5);
  Y0_5=X0_5;
  if idisp, Y0_5, end
```

6.2. Converted files of the example function

The conversion of the function *simpro* (written with Matlab language) produces 2 files : *simpro.h* and *simpro.c* far bigger than the original :

simpro.h :

```
/*
 * MATLAB Compiler: 3.0
 * Date: Fri Sep 27 14:55:29 2002
 * Arguments: "-B" "macro_default" "-O" "all" "-O" "fold_scalar_mxarrays:on"
 * "-O" "fold_non_scalar_mxarrays:on" "-O" "optimize_integer_for_loops:on" "-O"
 * "array_indexing:on" "-O" "optimize_conditionals:on" "-O" "all" "-O"
 * "fold_scalar_mxarrays:on" "-O" "fold_non_scalar_mxarrays:on" "-O"
 * "optimize_integer_for_loops:on" "-O" "array_indexing:on" "-O"
 * "optimize_conditionals:on" "-x" "-W" "mex" "-L" "C" "-t" "-T"
 * "link:mexlibrary" "libmatlbmx.mlib" "-h" "-v" "-d" "compil" "simpro.m"
 */

#ifndef MLF_V2
#define MLF_V2 1
#endif

#ifndef __simpro_h
#define __simpro_h 1

#ifdef __cplusplus
extern "C" {
#endif

#include "libmatlb.h"

extern void InitializeModule_simpro(void);
extern void TerminateModule_simpro(void);
extern _mexLocalFunctionTable _local_function_table_simpro;

extern mxArray * mlfSimpro(mxArray * * Y0_0,
                           mxArray * * Y0_1,
```

```
                mxArray * * CG0_2,
                mxArray * * CL0_2,
                mxArray * * FR0_2,
                mxArray * * X0_2,
                mxArray * * Y0_2,
                mxArray * * CG0_3,
                mxArray * * CL0_3,
                mxArray * * X0_3,
                mxArray * * Y0_3,
                mxArray * * dX0_3,
                mxArray * * CG0_4,
                mxArray * * CL0_4,
                mxArray * * FR0_4,
                mxArray * * X0_4,
                mxArray * * Y0_4,
                mxArray * * CG0_5,
                mxArray * * CL0_5,
                mxArray * * Y0_5,
                mxArray * * ri_5,
                mxArray * EE,
                mxArray * fm_O2_0,
                mxArray * fm_CO2_0,
                mxArray * rO2_0,
                mxArray * rCO2_0,
                mxArray * initNH3,
                mxArray * initNO2,
                mxArray * initAcOH,
                mxArray * initBuOH,
                mxArray * initLiqueL,
                mxArray * addCO2,
                mxArray * addO2,
                mxArray * Yx_AcOH,
                mxArray * YO2_5,
                mxArray * YCO2_5,
                mxArray * spNO3_5,
                mxArray * share2,
                mxArray * cover);
extern void mlxSimpro(int nlhs, mxArray * plhs[], int nrhs, mxArray * prhs[]);

#ifdef __cplusplus
}
#endif

#endif
```

simpro.c :

```
/*
 * MATLAB Compiler: 3.0
 * Date: Fri Sep 27 14:55:29 2002
 * Arguments: "-B" "macro_default" "-O" "all" "-O" "fold_scalar_mxarrays:on"
 * "-O" "fold_non_scalar_mxarrays:on" "-O" "optimize_integer_for_loops:on" "-O"
 * "array_indexing:on" "-O" "optimize_conditionals:on" "-O" "all" "-O"
 * "fold_scalar_mxarrays:on" "-O" "fold_non_scalar_mxarrays:on" "-O"
 * "optimize_integer_for_loops:on" "-O" "array_indexing:on" "-O"
 * "optimize_conditionals:on" "-x" "-W" "mex" "-L" "C" "-t" "-T"
 * "link:mexlibrary" "libmatlbmx.mlib" "-h" "-v" "-d" "compil" "simpro.m"
 */
#include "simpro.h"
#include "libmatlbm.h"
#include "link245.h"
#include "link2_3.h"
#include "link3_4.h"
#include "stesta_2.h"
#include "stesta_3.h"
#include "stesta_4.h"
#include "stesta_5.h"

extern mxArray * A2_2;
extern mxArray * A2_4;
extern mxArray * A2_5;
extern mxArray * Ae_3;
```

```
extern mxArray * B2_2;
extern mxArray * B2_4;
extern mxArray * B2_5;
extern mxArray * Be_3;
extern mxArray * Ce_3;
extern mxArray * De_3;
extern mxArray * Diete_5;
extern mxArray * Dietw_5;
extern mxArray * E_3;
extern mxArray * Ea_2;
extern mxArray * Ea_4;
extern mxArray * EpsJ_2;
extern mxArray * Es_2;
extern mxArray * Es_4;
extern mxArray * FRmax_2;
extern mxArray * FRmaxc_4;
extern mxArray * FRmin_2;
extern mxArray * FRmin_4;
extern mxArray * Fin_1;
extern mxArray * Fin_2;
extern mxArray * Fin_3;
extern mxArray * Fin_4;
extern mxArray * Fin_5;
extern mxArray * Fmin_4;
extern mxArray * GG1_2;
extern mxArray * GG1_4;
extern mxArray * GG2_2;
extern mxArray * GG2_4;
extern mxArray * Gin_0;
extern mxArray * Gin_1;
extern mxArray * Gin_2;
extern mxArray * Gin_3;
extern mxArray * Gin_5;
extern mxArray * KJ_2;
extern mxArray * KSNO3_4;
extern mxArray * KSPC_4;
extern mxArray * KSPO4_4;
extern mxArray * KSSO4_2;
extern mxArray * KSSO4_4;
extern mxArray * Kdis_1;
extern mxArray * Kdis_2;
extern mxArray * Kdis_3;
extern mxArray * Kdis_4;
extern mxArray * Kdis_5;
extern mxArray * KjEPS_4;
extern mxArray * Kj_4;
extern mxArray * KlNb_3;
extern mxArray * KlNs_3;
extern mxArray * KmNb_3;
extern mxArray * KmNs_3;
extern mxArray * NB_3;
extern mxArray * NG1_1;
extern mxArray * NG_1;
extern mxArray * NG_2;
extern mxArray * NG_3;
extern mxArray * NG_4;
extern mxArray * NG_5;
extern mxArray * NI_2;
extern mxArray * NI_3;
extern mxArray * NL_1;
extern mxArray * NL_2;
extern mxArray * NL_3;
extern mxArray * NL_4;
extern mxArray * NL_5;
extern mxArray * NO_2;
extern mxArray * NO_3;
extern mxArray * NO_4;
extern mxArray * NS_2;
extern mxArray * NS_3;
extern mxArray * NS_5;
extern mxArray * NV_3;
```

```
extern mxArray * NX_2;
extern mxArray * NX_3;
extern mxArray * RL_3;
extern mxArray * RT_2;
extern mxArray * RT_4;
extern mxArray * VG_5;
extern mxArray * VL_2;
extern mxArray * VL_4;
extern mxArray * VL_5;
extern mxArray * VM_0;
extern mxArray * WX_3;
extern mxArray * WYG_3;
extern mxArray * WYL_3;
extern mxArray * Ym1_3;
extern mxArray * Yp_1p;
extern mxArray * Yx1_2;
extern mxArray * Yx1_3;
extern mxArray * Yx1_4;
extern mxArray * Yx1e_5;
extern mxArray * Yx1w_5;
extern mxArray * Yx_3;
extern mxArray * fI_2;
extern mxArray * fI_4;
extern mxArray * iCO2_3;
extern mxArray * iNH3_3;
extern mxArray * iNO2_3;
extern mxArray * iO2_3;
extern mxArray * iSub_3;
extern mxArray * iXNb_3;
extern mxArray * iXNs_3;
extern mxArray * iXag_3;
extern mxArray * idisp;
extern mxArray * indG_3;
extern mxArray * indL_3;
extern mxArray * ind_3u5;
extern mxArray * maint_3;
extern mxArray * muEPS_4;
extern mxArray * muM_2;
extern mxArray * muM_4;
extern mxArray * mumax_3;
extern mxArray * q_rhod_2;
extern mxArray * rKdis_1;
extern mxArray * vNB_2;
extern mxArray * vNB_4;
extern mxArray * vNM_2;
extern mxArray * vNM_4;
extern mxArray * vNS_2;
extern mxArray * zCH_4;
extern mxArray * zPC_4;
extern mxArray * zmin_I_2;
extern mxArray * zmin_I_4;
static mxArray * _mxarray0_;
static mxArray * _mxarray1_;

static double _array3_[4] = { 1.0, 2.0, 3.0, 4.0 };
static mxArray * _mxarray2_;
static mxArray * _mxarray4_;
static mxArray * _mxarray5_;
static mxArray * _mxarray6_;

static double _array8_[2] = { 1.0, 2.0 };
static mxArray * _mxarray7_;

void InitializeModule_simpro(void) {
   _mxarray0_ = mclInitializeDouble(2.0);
   _mxarray1_ = mclInitializeDouble(1.0);
   _mxarray2_ = mclInitializeDoubleVector(1, 4, _array3_);
   _mxarray4_ = mclInitializeDouble(4.0);
   _mxarray5_ = mclInitializeDouble(5.0);
   _mxarray6_ = mclInitializeDouble(0.0);
   _mxarray7_ = mclInitializeDoubleVector(1, 2, _array8_);
```

```
}

void TerminateModule_simpro(void) {
   mxDestroyArray(_mxarray7_);
   mxDestroyArray(_mxarray6_);
   mxDestroyArray(_mxarray5_);
   mxDestroyArray(_mxarray4_);
   mxDestroyArray(_mxarray2_);
   mxDestroyArray(_mxarray1_);
   mxDestroyArray(_mxarray0_);
}

static mxArray * Msimpro(mxArray * * Y0_0,
                         mxArray * * Y0_1,
                         mxArray * * CG0_2,
                         mxArray * * CL0_2,
                         mxArray * * FR0_2,
                         mxArray * * X0_2,
                         mxArray * * Y0_2,
                         mxArray * * CG0_3,
                         mxArray * * CL0_3,
                         mxArray * * X0_3,
                         mxArray * * Y0_3,
                         mxArray * * dX0_3,
                         mxArray * * CG0_4,
                         mxArray * * CL0_4,
                         mxArray * * FR0_4,
                         mxArray * * X0_4,
                         mxArray * * Y0_4,
                         mxArray * * CG0_5,
                         mxArray * * CL0_5,
                         mxArray * * Y0_5,
                         mxArray * * ri_5,
                         int nargout_,
                         mxArray * EE,
                         mxArray * fm_O2_0,
                         mxArray * fm_CO2_0,
                         mxArray * rO2_0,
                         mxArray * rCO2_0,
                         mxArray * initNH3,
                         mxArray * initNO2,
                         mxArray * initAcOH,
                         mxArray * initBuOH,
                         mxArray * initLiqueL,
                         mxArray * addCO2,
                         mxArray * addO2,
                         mxArray * Yx_AcOH,
                         mxArray * YO2_5,
                         mxArray * YCO2_5,
                         mxArray * spNO3_5,
                         mxArray * share2,
                         mxArray * cover);

_mexLocalFunctionTable _local_function_table_simpro
  = { 0, (mexFunctionTableEntry *)NULL };

/*
 * The function "mlfSimpro" contains the normal interface for the "simpro"
 * M-function from file "j:\partage\c0-5_mb\simpro.m" (lines 1-163). This
 * function processes any input arguments and passes them to the implementation
 * version of the function, appearing above.
 */
mxArray * mlfSimpro(mxArray * * Y0_0,
                    mxArray * * Y0_1,
                    mxArray * * CG0_2,
                    mxArray * * CL0_2,
                    mxArray * * FR0_2,
                    mxArray * * X0_2,
                    mxArray * * Y0_2,
                    mxArray * * CG0_3,
                    mxArray * * CL0_3,
```

```
                mxArray * * X0_3,
                mxArray * * Y0_3,
                mxArray * * dX0_3,
                mxArray * * CG0_4,
                mxArray * * CL0_4,
                mxArray * * FR0_4,
                mxArray * * X0_4,
                mxArray * * Y0_4,
                mxArray * * CG0_5,
                mxArray * * CL0_5,
                mxArray * * Y0_5,
                mxArray * * ri_5,
                mxArray * EE,
                mxArray * fm_O2_0,
                mxArray * fm_CO2_0,
                mxArray * rO2_0,
                mxArray * rCO2_0,
                mxArray * initNH3,
                mxArray * initNO2,
                mxArray * initAcOH,
                mxArray * initBuOH,
                mxArray * initLiqueL,
                mxArray * addCO2,
                mxArray * addO2,
                mxArray * Yx_AcOH,
                mxArray * YO2_5,
                mxArray * YCO2_5,
                mxArray * spNO3_5,
                mxArray * share2,
                mxArray * cover) {
  int nargout = 1;
  mxArray * CG0_0 = NULL;
  mxArray * Y0_0__ = NULL;
  mxArray * Y0_1__ = NULL;
  mxArray * CG0_2__ = NULL;
  mxArray * CL0_2__ = NULL;
  mxArray * FR0_2__ = NULL;
  mxArray * X0_2__ = NULL;
  mxArray * Y0_2__ = NULL;
  mxArray * CG0_3__ = NULL;
  mxArray * CL0_3__ = NULL;
  mxArray * X0_3__ = NULL;
  mxArray * Y0_3__ = NULL;
  mxArray * dX0_3__ = NULL;
  mxArray * CG0_4__ = NULL;
  mxArray * CL0_4__ = NULL;
  mxArray * FR0_4__ = NULL;
  mxArray * X0_4__ = NULL;
  mxArray * Y0_4__ = NULL;
  mxArray * CG0_5__ = NULL;
  mxArray * CL0_5__ = NULL;
  mxArray * Y0_5__ = NULL;
  mxArray * ri_5__ = NULL;
  mlfEnterNewContext(
    21,
    18,
    Y0_0,
    Y0_1,
    CG0_2,
    CL0_2,
    FR0_2,
    X0_2,
    Y0_2,
    CG0_3,
    CL0_3,
    X0_3,
    Y0_3,
    dX0_3,
    CG0_4,
    CL0_4,
    FR0_4,
```

```
  X0_4,
  Y0_4,
  CG0_5,
  CL0_5,
  Y0_5,
  ri_5,
  EE,
  fm_O2_0,
  fm_CO2_0,
  rO2_0,
  rCO2_0,
  initNH3,
  initNO2,
  initAcOH,
  initBuOH,
  initLiqueL,
  addCO2,
  addO2,
  Yx_AcOH,
  YO2_5,
  YCO2_5,
  spNO3_5,
  share2,
  cover);
if (Y0_0 != NULL) {
   ++nargout;
}
if (Y0_1 != NULL) {
   ++nargout;
}
if (CG0_2 != NULL) {
   ++nargout;
}
if (CL0_2 != NULL) {
   ++nargout;
}
if (FR0_2 != NULL) {
   ++nargout;
}
if (X0_2 != NULL) {
   ++nargout;
}
if (Y0_2 != NULL) {
   ++nargout;
}
if (CG0_3 != NULL) {
   ++nargout;
}
if (CL0_3 != NULL) {
   ++nargout;
}
if (X0_3 != NULL) {
   ++nargout;
}
if (Y0_3 != NULL) {
   ++nargout;
}
if (dX0_3 != NULL) {
   ++nargout;
}
if (CG0_4 != NULL) {
   ++nargout;
}
if (CL0_4 != NULL) {
   ++nargout;
}
if (FR0_4 != NULL) {
   ++nargout;
}
if (X0_4 != NULL) {
   ++nargout;
```

```
 }
 if (Y0_4 != NULL) {
    ++nargout;
 }
 if (CG0_5 != NULL) {
    ++nargout;
 }
 if (CL0_5 != NULL) {
    ++nargout;
 }
 if (Y0_5 != NULL) {
    ++nargout;
 }
 if (ri_5 != NULL) {
    ++nargout;
 }
 CG0_0
  = Msimpro(
      &Y0_0__,
      &Y0_1__,
      &CG0_2__,
      &CL0_2__,
      &FR0_2__,
      &X0_2__,
      &Y0_2__,
      &CG0_3__,
      &CL0_3__,
      &X0_3__,
      &Y0_3__,
      &dX0_3__,
      &CG0_4__,
      &CL0_4__,
      &FR0_4__,
      &X0_4__,
      &Y0_4__,
      &CG0_5__,
      &CL0_5__,
      &Y0_5__,
      &ri_5__,
      nargout,
      EE,
      fm_O2_0,
      fm_CO2_0,
      rO2_0,
      rCO2_0,
      initNH3,
      initNO2,
      initAcOH,
      initBuOH,
      initLiqueL,
      addCO2,
      addO2,
      Yx_AcOH,
      YO2_5,
      YCO2_5,
      spNO3_5,
      share2,
      cover);
 mlfRestorePreviousContext(
  21,
  18,
  Y0_0,
  Y0_1,
  CG0_2,
  CL0_2,
  FR0_2,
  X0_2,
  Y0_2,
  CG0_3,
  CL0_3,
  X0_3,
```

```
    Y0_3,
    dX0_3,
    CG0_4,
    CL0_4,
    FR0_4,
    X0_4,
    Y0_4,
    CG0_5,
    CL0_5,
    Y0_5,
    ri_5,
    EE,
    fm_O2_0,
    fm_CO2_0,
    rO2_0,
    rCO2_0,
    initNH3,
    initNO2,
    initAcOH,
    initBuOH,
    initLiqueL,
    addCO2,
    addO2,
    Yx_AcOH,
    YO2_5,
    YCO2_5,
    spNO3_5,
    share2,
    cover);
  if (Y0_0 != NULL) {
     mclCopyOutputArg(Y0_0, Y0_0__);
  } else {
     mxDestroyArray(Y0_0__);
  }
  if (Y0_1 != NULL) {
     mclCopyOutputArg(Y0_1, Y0_1__);
  } else {
     mxDestroyArray(Y0_1__);
  }
  if (CG0_2 != NULL) {
     mclCopyOutputArg(CG0_2, CG0_2__);
  } else {
     mxDestroyArray(CG0_2__);
  }
  if (CL0_2 != NULL) {
     mclCopyOutputArg(CL0_2, CL0_2__);
  } else {
     mxDestroyArray(CL0_2__);
  }
  if (FR0_2 != NULL) {
     mclCopyOutputArg(FR0_2, FR0_2__);
  } else {
     mxDestroyArray(FR0_2__);
  }
  if (X0_2 != NULL) {
     mclCopyOutputArg(X0_2, X0_2__);
  } else {
     mxDestroyArray(X0_2__);
  }
  if (Y0_2 != NULL) {
     mclCopyOutputArg(Y0_2, Y0_2__);
  } else {
     mxDestroyArray(Y0_2__);
  }
  if (CG0_3 != NULL) {
     mclCopyOutputArg(CG0_3, CG0_3__);
  } else {
     mxDestroyArray(CG0_3__);
  }
  if (CL0_3 != NULL) {
     mclCopyOutputArg(CL0_3, CL0_3__);
```

```
    } else {
      mxDestroyArray(CL0_3__);
    }
    if (X0_3 != NULL) {
      mclCopyOutputArg(X0_3, X0_3__);
    } else {
      mxDestroyArray(X0_3__);
    }
    if (Y0_3 != NULL) {
      mclCopyOutputArg(Y0_3, Y0_3__);
    } else {
      mxDestroyArray(Y0_3__);
    }
    if (dX0_3 != NULL) {
      mclCopyOutputArg(dX0_3, dX0_3__);
    } else {
      mxDestroyArray(dX0_3__);
    }
    if (CG0_4 != NULL) {
      mclCopyOutputArg(CG0_4, CG0_4__);
    } else {
      mxDestroyArray(CG0_4__);
    }
    if (CL0_4 != NULL) {
      mclCopyOutputArg(CL0_4, CL0_4__);
    } else {
      mxDestroyArray(CL0_4__);
    }
    if (FR0_4 != NULL) {
      mclCopyOutputArg(FR0_4, FR0_4__);
    } else {
      mxDestroyArray(FR0_4__);
    }
    if (X0_4 != NULL) {
      mclCopyOutputArg(X0_4, X0_4__);
    } else {
      mxDestroyArray(X0_4__);
    }
    if (Y0_4 != NULL) {
      mclCopyOutputArg(Y0_4, Y0_4__);
    } else {
      mxDestroyArray(Y0_4__);
    }
    if (CG0_5 != NULL) {
      mclCopyOutputArg(CG0_5, CG0_5__);
    } else {
      mxDestroyArray(CG0_5__);
    }
    if (CL0_5 != NULL) {
      mclCopyOutputArg(CL0_5, CL0_5__);
    } else {
      mxDestroyArray(CL0_5__);
    }
    if (Y0_5 != NULL) {
      mclCopyOutputArg(Y0_5, Y0_5__);
    } else {
      mxDestroyArray(Y0_5__);
    }
    if (ri_5 != NULL) {
      mclCopyOutputArg(ri_5, ri_5__);
    } else {
      mxDestroyArray(ri_5__);
    }
    return mlfReturnValue(CG0_0);
}

/*
 * The function "mlxSimpro" contains the feval interface for the "simpro"
 * M-function from file "j:\partage\c0-5_mb\simpro.m" (lines 1-163). The feval
 * function calls the implementation version of simpro through this function.
 * This function processes any input arguments and passes them to the
```

```
 * implementation version of the function, appearing above.
 */
void mlxSimpro(int nlhs, mxArray * plhs[], int nrhs, mxArray * prhs[]) {
  mxArray * mprhs[18];
  mxArray * mplhs[22];
   int i;
  if (nlhs > 22) {
     mlfError(
      mxCreateString(
        "Run-time Error: File: simpro Lin e: 9 Column: "
        "1 The function \"simpro\" was called with mor"
        "e than the declared number of outputs (22)."),
       NULL);
   }
  if (nrhs > 18) {
     mlfError(
      mxCreateString(
        "Run-time Error: File: simpro Line: 9 Column: "
        "1 The function \"simpro\" was called with mor"
        "e than the declared number of inputs (18)."),
       NULL);
   }
  for (i = 0; i < 22; ++i) {
     mplhs[i] = NULL;
   }
  for (i = 0; i < 18 && i < nrhs; ++i) {
     mprhs[i] = prhs[i];
   }
  for (; i < 18; ++i) {
     mprhs[i] = NULL;
   }
  mlfEnterNewContext(
    0,
    18,
    mprhs[0],
    mprhs[1],
    mprhs[2],
    mprhs[3],
    mprhs[4],
    mprhs[5],
    mprhs[6],
    mprhs[7],
    mprhs[8],
    mprhs[9],
    mprhs[10],
    mprhs[11],
    mprhs[12],
    mprhs[13],
    mprhs[14],
    mprhs[15],
    mprhs[16],
    mprhs[17]);
  mplhs[0]
    = Msimpro(
       &mplhs[1],
       &mplhs[2],
       &mplhs[3],
       &mplhs[4],
       &mplhs[5],
       &mplhs[6],
       &mplhs[7],
       &mplhs[8],
       &mplhs[9],
       &mplhs[10],
       &mplhs[11],
       &mplhs[12],
       &mplhs[13],
       &mplhs[14],
       &mplhs[15],
       &mplhs[16],
       &mplhs[17],
```

```
        &mplhs[18],
        &mplhs[19],
        &mplhs[20],
        &mplhs[21],
        nlhs,
        mprhs[0],
        mprhs[1],
        mprhs[2],
        mprhs[3],
        mprhs[4],
        mprhs[5],
        mprhs[6],
        mprhs[7],
        mprhs[8],
        mprhs[9],
        mprhs[10],
        mprhs[11],
        mprhs[12],
        mprhs[13],
        mprhs[14],
        mprhs[15],
        mprhs[16],
        mprhs[17]);
    mlfRestorePreviousContext(
      0,
      18,
      mprhs[0],
      mprhs[1],
      mprhs[2],
      mprhs[3],
      mprhs[4],
      mprhs[5],
      mprhs[6],
      mprhs[7],
      mprhs[8],
      mprhs[9],
      mprhs[10],
      mprhs[11],
      mprhs[12],
      mprhs[13],
      mprhs[14],
      mprhs[15],
      mprhs[16],
      mprhs[17]);
    plhs[0] = mplhs[0];
    for (i = 1; i < 22 && i < nlhs; ++i) {
      plhs[i] = mplhs[i];
    }
    for (; i < 22; ++i) {
      mxDestroyArray(mplhs[i]);
    }
}

/*
 * The function "Msimpro" is the implementation version of the "simpro"
 * M-function from file "j:\partage\c0-5_mb\simpro.m" (lines 1-163). It
 * contains the actual compiled code for that M-function. It is a static
 * function and must only be called from one of the interface functions,
 * appearing below.
 */
/*
 * %********************************************************************
 * %      Connected compartments 0,1,2,3,4A and 4B (numbered 5 here)    *
 * %      Version 3.2     September 2002                                *
 * %                                                                    *
 * %      simpro.m   Simulation of the MELISSA loop (HPC included)      *
 * %               Open loop and Steady state                           *
 * %                                                                    *
 * %********************************************************************
 * function [CG0_0, Y0_0, Y0_1, CG0_2, CL0_2, FR0_2, X0_2, Y0_2, ...
 */
```

```
static mxArray * Msimpro(mxArray * * Y0_0,
                         mxArray * * Y0_1,
                         mxArray * * CG0_2,
                         mxArray * * CL0_2,
                         mxArray * * FR0_2,
                         mxArray * * X0_2,
                         mxArray * * Y0_2,
                         mxArray * * CG0_3,
                         mxArray * * CL0_3,
                         mxArray * * X0_3,
                         mxArray * * Y0_3,
                         mxArray * * dX0_3,
                         mxArray * * CG0_4,
                         mxArray * * CL0_4,
                         mxArray * * FR0_4,
                         mxArray * * X0_4,
                         mxArray * * Y0_4,
                         mxArray * * CG0_5,
                         mxArray * * CL0_5,
                         mxArray * * Y0_5,
                         mxArray * * ri_5,
                         int nargout_,
                         mxArray * EE,
                         mxArray * fm_O2_0,
                         mxArray * fm_CO2_0,
                         mxArray * rO2_0,
                         mxArray * rCO2_0,
                         mxArray * initNH3,
                         mxArray * initNO2,
                         mxArray * initAcOH,
                         mxArray * initBuOH,
                         mxArray * initLiqueL,
                         mxArray * addCO2,
                         mxArray * addO2,
                         mxArray * Yx_AcOH,
                         mxArray * YO2_5,
                         mxArray * YCO2_5,
                         mxArray * spNO3_5,
                         mxArray * share2,
                         mxArray * cover) {
  mexLocalFunctionTable save_local_function_table_
    = mclSetCurrentLocalFunctionTable(&_local_function_table_simpro);
  mxArray * CG0_0 = NULL;
  mxArray * X0_5 = NULL;
  mxArray * indN1 = NULL;
  mxArray * CGL0 = NULL;
  mxArray * X0_3p = NULL;
  mxArray * dX0_2 = NULL;
  mxArray * cX0_2 = NULL;
  mxArray * aa = NULL;
  mxArray * ind = NULL;
  mxArray * fm_Bu_2 = NULL;
  mxArray * fm_Ac_2 = NULL;
  mxArray * fm_NH3_2 = NULL;
  mxArray * fm_CO2_2 = NULL;
  mxArray * NY = NULL;
  mxArray * NY1 = NULL;
  mxArray * Y_1 = NULL;
  mxArray * ans = NULL;
  mclCopyArray(&EE);
  mclCopyArray(&fm_O2_0);
  mclCopyArray(&fm_CO2_0);
  mclCopyArray(&rO2_0);
  mclCopyArray(&rCO2_0);
  mclCopyArray(&initNH3);
  mclCopyArray(&initNO2);
  mclCopyArray(&initAcOH);
  mclCopyArray(&initBuOH);
  mclCopyArray(&initLiqueL);
  mclCopyArray(&addCO2);
  mclCopyArray(&addO2);
```

```
mclCopyArray(&Yx_AcOH);
mclCopyArray(&YO2_5);
mclCopyArray(&YCO2_5);
mclCopyArray(&spNO3_5);
mclCopyArray(&share2);
mclCopyArray(&cover);
/*
 * CG0_3, CL0_3, X0_3, Y0_3, dX0_3, CG0_4, CL0_4, FR0_4, X0_4, Y0_4, ...
 * CG0_5, CL0_5, Y0_5, ri_5] = ...
 * simpro(EE, fm_O2_0, fm_CO2_0, rO2_0, rCO2_0, initNH3, initNO2, initAcOH, initBuOH, ...
 * initLiqueL, addCO2, addO2, Yx_AcOH, YO2_5, YCO2_5, spNO3_5, share2, cover)
 *
 * global VM_0  Gin_0
 * global Gin_1  Fin_1  NG1_1  NG_1  NL_1  Kdis_1  rKdis_1  Yp_1p
 * global Gin_2  Fin_2  NG_2  NL_2  NO_2  NS_2  NI_2  NX_2  vNB_2  vNM_2  vNS_2  ...
 * VL_2  Yx1_2  Kdis_2  KSSO4_2  A2_2  B2_2  GG1_2  GG2_2  ...
 * FRmin_2  FRmax_2  fI_2  RT_2  Ea_2  Es_2  muM_2  KJ_2  EpsJ_2  q_rhod_2  zmin_I_2
 * global Gin_3  Fin_3  NL_3  NG_3  NS_3  NB_3  NX_3  NO_3  NI_3  NV_3 ...
 * Ae_3  Be_3  Ce_3  De_3  E_3  WX_3  WYG_3  WYL_3 ...
 * iO2_3  iCO2_3  iNH3_3  iNO2_3  iSub_3  iXNs_3  iXNb_3  iXag_3  ...
 * KlNs_3  KlNb_3  KmNs_3  KmNb_3  mumax_3  maint_3  Yx_3  Yx1_3  Ym1_3  ...
 * RL_3  indG_3  indL_3  Kdis_3
 * global Fin_4  NG_4  NL_4  NO_4  vNB_4  vNM_4  VL_4  Yx1_4  Kdis_4  ...
 * A2_4  B2_4  GG1_4  GG2_4  ...
 * FRmin_4  FRmax_4  FRmaxc_4  fI_4  zPC_4  zCH_4  ...
 * RT_4  Ea_4  Es_4  muM_4  muEPS_4  Kj_4  KjEPS_4  Fmin_4  zmin_I_4  ...
 * KSNO3_4  KSSO4_4  KSPO4_4  KSPC_4
 * global Gin_5  Fin_5  NG_5  NL_5  NS_5  VG_5  VL_5  Yx1e_5  Yx1w_5  Kdis_5  ...
 * Diete_5  Dietw_5  A2_5  B2_5  ind_3u5
 * global idisp
 *
 * %>>> CONSU
 * % Concentrations in the incoming Gas of Consu
 * % ========================================
 * CG0_0 = [fm_O2_0; fm_CO2_0]/VM_0;  % mol/l of O2 CO2 going into Consu
 */
mlfAssign(
  &CG0_0,
  mclMrdivide(
    mlfVertcat(
      mclVa(fm_O2_0, "fm_O2_0"), mclVa(fm_CO2_0, "fm_CO2_0"), NULL),
    mclVg(&VM_0, "VM_0")));
/*
 * % Initial output vector of Consu (static state)
 * % ==============================================
 * Y0_0 = CG0_0 + [rO2_0; rCO2_0]/Gin_0;
 */
mlfAssign(
  Y0_0,
  mclPlus(
    mclVv(CG0_0, "CG0_0"),
    mclMrdivide(
      mlfVertcat(mclVa(rO2_0, "rO2_0"), mclVa(rCO2_0, "rCO2_0"), NULL),
      mclVg(&Gin_0, "Gin_0"))));
/*
 *
 * %>>> LIQUE
 * % Concentrations in the incoming Gas and Liquid flows of Lique
 * % ============================================================
 * %1. Gas phase : inert gas
 * %2. Liquid phase :
 * %cFaec_0 = qFaec_0/Fin_1; % mol/l
 * %cUrea_0 = qUrea_0/Fin_1; % mol/l
 * %cSO4_0  = qSO4_0/Fin_1;  % mol/l
 * %cPO4_0  = qPO4_0/Fin_1;  % mol/l
 * % Output vector of Lique (static state)
 * % ====================================
 * Y_1 = EE(2)*Yp_1p; % (mol/h): production rate of H2 CO2 NH3 AcOH BuOH
 */
mlfAssign(
  &Y_1,
```

```
 mclMtimes(mclIntArrayRef1(mclVa(EE, "EE"), 2), mclVg(&Yp_1p, "Yp_1p")));
/*
 * NY1= NG1_1+2*NG_1;
 */
mlfAssign(
 &NY1,
 mclPlus(
  mclVg(&NG1_1, "NG1_1"), mclMtimes(_mxarray0_, mclVg(&NG_1, "NG_1"))));
/*
 * NY = NY1+NL_1;
 */
mlfAssign(&NY, mclPlus(mclVv(NY1, "NY1"), mclVg(&NL_1, "NL_1")));
/*
 * % Gas concentration in mol/l (H2 and CO2 are totaly under Gas phase)
 * Y0_1 = zeros(NY,1);
 */
mlfAssign(Y0_1, mlfZeros(mclVv(NY, "NY"), _mxarray1_, NULL));
/*
 * Y0_1(1:NG1_1) = Y_1(1:NG1_1)/Gin_1;  % output conc. of H2 in Gas (mol/l)
 */
mclArrayAssign1(
 Y0_1,
 mclMrdivide(
  mclArrayRef1(
   mclVv(Y_1, "Y_1"),
   mlfColon(_mxarray1_, mclVg(&NG1_1, "NG1_1"), NULL)),
  mclVg(&Gin_1, "Gin_1")),
 mlfColon(_mxarray1_, mclVg(&NG1_1, "NG1_1"), NULL));
/*
 * % Liquid concentration in mol/l (CO2 NH3 AcOH & BuOH are totaly in Liquid phase)
 * Y0_1(NY1+[1:4]) = Y_1(NY1+[1:4])/Fin_1./(1+Kdis_1(1:4)); % conc. in mol/l of CO2 NH3 molec. form
 */
mclArrayAssign1(
 Y0_1,
 mclRdivide(
  mclMrdivide(
   mclArrayRef1(
    mclVv(Y_1, "Y_1"), mclPlus(mclVv(NY1, "NY1"), _mxarray2_)),
   mclVg(&Fin_1, "Fin_1")),
  mclPlus(
   _mxarray1_,
   mclArrayRef1(
    mclVg(&Kdis_1, "Kdis_1"), mlfColon(_mxarray1_, _mxarray4_, NULL)))),
 mclPlus(mclVv(NY1, "NY1"), _mxarray2_));
/*
 * Y0_1(NY1+5:NY) = initLiqueL;   % conc. SO4 PO4 in mol/l
 */
mclArrayAssign1(
 Y0_1,
 mclVa(initLiqueL, "initLiqueL"),
 mlfColon(mclPlus(mclVv(NY1, "NY1"), _mxarray5_), mclVv(NY, "NY"), NULL));
/*
 *
 * %>>> RHODO
 * % Initial concentrations in the incoming Gas and Liquid flows of Rhodo
 * % ====================================================================
 * %1. Gas phase (loop of inert gas) :
 * fm_CO2_2 = 0;   % no CO2  (molar fraction)
 */
mlfAssign(&fm_CO2_2, _mxarray6_);
/*
 * fm_NH3_2 = 0;   % no NH3  (molar fraction)
 */
mlfAssign(&fm_NH3_2, _mxarray6_);
/*
 * fm_Ac_2  = 0;   % no AcOH (molar fraction)
 */
mlfAssign(&fm_Ac_2, _mxarray6_);
/*
 * fm_Bu_2  = 0;   % no BuOH (molar fraction)
 */
```

```
mlfAssign(&fm_Bu_2, _mxarray6_);
/*
 * CG0_2 = [fm_CO2_2; fm_NH3_2; fm_Ac_2; fm_Bu_2] / VM_0;  % mol/l
 */
mlfAssign(
  CG0_2,
  mclMrdivide(
    mlfVertcat(
      mclVv(fm_CO2_2, "fm_CO2_2"),
      mclVv(fm_NH3_2, "fm_NH3_2"),
      mclVv(fm_Ac_2, "fm_Ac_2"),
      mclVv(fm_Bu_2, "fm_Bu_2"),
      NULL),
    mclVg(&VM_0, "VM_0")));
/*
 * %2. Liquid phase :
 * %CL0_2 = [cCO2_0; cNH3_0; cAc_0; cBu_0; cSO4_0; cPO4_0] % mol/l
 * ind = [NY1+1:NY];
 */
mlfAssign(
  &ind,
  mlfColon(mclPlus(mclVv(NY1, "NY1"), _mxarray1_), mclVv(NY, "NY"), NULL));
/*
 * CL0_2  = Y0_1(ind).*rKdis_1; % mol/l of the molecular form CO2 NH3 AcOH BuOH
 */
mlfAssign(
  CL0_2,
  mclTimes(
    mclArrayRef1(mclVv(*Y0_1, "Y0_1"), mclVv(ind, "ind")),
    mclVg(&rKdis_1, "rKdis_1")));
/*
 * CL0_2(1:NG_2) = CL0_2(1:NG_2) + ...
 */
mclArrayAssign1(
  CL0_2,
  mclPlus(
    mclArrayRef1(
      mclVv(*CL0_2, "CL0_2"),
      mlfColon(_mxarray1_, mclVg(&NG_2, "NG_2"), NULL)),
    mlfVertcat(
      mclVa(YCO2_5, "YCO2_5"),
      mclVa(initNH3, "initNH3"),
      mclVa(initAcOH, "initAcOH"),
      mclVa(initBuOH, "init BuOH"),
      NULL)),
  mlfColon(_mxarray1_, mclVg(&NG_2, "NG_2"), NULL));
/*
 * [YCO2_5;    % addition of CO2 coming from HPC output
 * initNH3;   % NH3 GL coming from HPC output and going into L phase of Rhodo input
 * initAcOH;  % AcOH coming from L Rhodo output
 * initBuOH]; % BuOH coming from L Rhodo output
 * % Initial concentration of biomass in Rhodo (to consume 'Yx_AcOH' part of AcOH load)
 * % ========================================================================
 * aa = 1 + Yx_AcOH*((1+Kdis_2(3))/(1+Kdis_2(NG_2))*Yx1_2(4,2)/Yx1_2(3,1) - 1);
 */
mlfAssign(
  &aa,
  mclPlus(
    _mxarray1_,
    mclMtimes(
      mclVa(Yx_AcOH, "Yx_AcOH"),
      mclMinus(
        mclMrdivide(
          mclMtimes(
            mclMrdivide(
              mclPlus(
                _mxarray1_, mclIntArrayRef1(mclVg(&Kdis_2, "Kdis_2"), 3)),
              mclPlus(
                _mxarray1_,
                mclArrayRef1(
                  mclVg(&Kdis_2, "Kdis_2"), mclVg(&NG_2, "NG_2")))),
```

```
          mclIntArrayRef2(mclVg(&Yx1_2, "Yx1_2"), 4, 2)),
        mclIntArrayRef2(mclVg(&Yx1_2, "Yx1_2"), 3, 1)),
       _mxarray1_))));
/*
 * cX0_2 = -Yx_AcOH * (1+Kdis_2(3)) / Yx1_2(3,1)* (aa*CL0_2(3)+CL0_2(NG_2)) / aa;
 */
mlfAssign(
 &cX0_2,
 mclMrdivide(
  mclMtimes(
   mclMrdivide(
    mclMtimes(
     mclUminus(mclVa(Yx_AcOH, "Yx_AcOH")),
     mclPlus(
      _mxarray1_, mclIntArrayRef1(mclVg(&Kdis_2, "Kdis_2"), 3))),
    mclIntArrayRef2(mclVg(&Yx1_2, "Yx1_2"), 3, 1)),
   mclPlus(
    mclMtimes(
     mclVv(aa, "aa"), mclIntArrayRef1(mclVv(*CL0_2, "CL0_2"), 3)),
    mclArrayRef1(mclVv(*CL0_2, "CL0_2"), mclVg(&NG_2, "NG_2")))),
  mclVv(aa, "aa")));
/*
 * % Initial light flux and state vector of Rhodo (steady state)
 * % ============================================================
 * %[FR0_2, X0_2, Y0_2] = stesta_2(...
 * % NG_2, NL_2, NO_2, vNB_2, vNM_2, VL_2, Fin_2, Yx1_2, KSSO4_2, ...
 * % A2_2, B2_2, GG1_2, GG2_2, CG0_2, CL0_2, cX0_2, ...
 * % FRmin_2, FRmax_2, fI_2, RT_2, Ea_2, Es_2, muM_2, KJ_2, EpsJ_2, q_rhod_2, zmin_I_2);
 * [FR0_2, X0_2, Y0_2, dX0_2] = stesta_2(...
 */
mlfAssign(
 FR0_2,
 mlfStesta_2(
  X0_2,
  Y0_2,
  &dX0_2,
  mclVg(&NG_2, "NG_2"),
  mclVg(&NL_2, "NL_2"),
  mclVg(&NO_2, "NO_2"),
  mclVg(&NS_2, "NS_2"),
  mclVg(&NI_2, "NI_2"),
  mclVg(&NX_2, "NX_2"),
  mclVg(&vNB_2, "vNB_2"),
  mclVg(&vNM_2, "vNM_2"),
  mclVg(&vNS_2, "vNS_2"),
  mclVg(&VL_2, "VL_2"),
  mclVg(&Fin_2, "Fin_2"),
  mclVg(&Yx1_2, "Yx1_2"),
  mclVg(&KSSO4_2, "KSSO4_2"),
  mclVg(&A2_2, "A2_2"),
  mclVg(&B2_2, "B2_2"),
  mclVg(&GG1_2, "GG1_2"),
  mclVg(&GG2_2, "GG2_2"),
  mclVv(*CG0_2, "CG0_2"),
  mclVv(*CL0_2, "CL0_2"),
  mclVv(cX0_2, "cX0_2"),
  mclVg(&FRmin_2, "FRmin_2"),
  mclVg(&FRmax_2, "FRmax_2"),
  mclVg(&fI_2, "fI_2"),
  mclVg(&RT_2, "RT_2"),
  mclVg(&Ea_2, "Ea_2"),
  mclVg(&Es_2, "Es_2"),
  mclVg(&muM_2, "muM_2"),
  mclVg(&KJ_2, "KJ_2"),
  mclVg(&EpsJ_2, "EpsJ_2"),
  mclVg(&q_rhod_2, "q_rhod_2"),
  mclVg(&zmin_I_2, "zmin_I_2")));
/*
 * NG_2, NL_2, NO_2, NS_2, NI_2, NX_2, vNB_2, vNM_2, vNS_2, VL_2, Fin_2, Yx1_2, KSSO4_2, ...
 * A2_2, B2_2, GG1_2, GG2_2, CG0_2, CL0_2, cX0_2, ...
 * FRmin_2, FRmax_2, fI_2, RT_2, Ea_2, Es_2, muM_2, KJ_2, EpsJ_2, q_rhod_2, zmin_I_2);
```

```
 * %1. CO2 NH3 in G phase of Rhodo going into L phase of Rhodo :
 * X0_2(1:2) = X0_2(1:2) + (Y0_2(1:2)*Gin_2/Fin_2)./(1+Kdis_2(1:2)); % modif of state vector
 */
mclArrayAssign1(
  X0_2,
  mclPlus(
    mclArrayRef1(
      mclVv(*X0_2, "X0_2"), mlfColon(_mxarray1_, _mxarray0_, NULL)),
    mclRdivide(
      mclMrdivide(
        mclMtimes(
          mclArrayRef1(
            mclVv(*Y0_2, "Y0_2"), mlfColon(_mxarray1_, _mxarray0_, NULL)),
          mclVg(&Gin_2, "Gin_2")),
        mclVg(&Fin_2, "Fin_2")),
      mclPlus(
        _mxarray1_,
        mclArrayRef1(
          mclVg(&Kdis_2, "Kdis_2"),
          mlfColon(_mxarray1_, _mxarray0_, NULL))))),
  mlfColon(_mxarray1_, _mxarray0_, NULL));
/*
 * %2. Modification of output vector
 * Y0_2(1:2) = zeros(2,1);
 */
mclArrayAssign1(
  Y0_2,
  mlfZeros(_mxarray0_, _mxarray1_, NULL),
  mlfColon(_mxarray1_, _mxarray0_, NULL));
/*
 * Y0_2(NG_2+[1:2]) = X0_2(1:2);                          % modif of output vector
 */
mclArrayAssign1(
  Y0_2,
  mclArrayRef1(
    mclVv(*X0_2, "X0_2"), mlfColon(_mxarray1_, _mxarray0_, NULL)),
  mclPlus(mclVg(&NG_2, "NG_2"), _mxarray7_));
/*
 *
 * %>>> NITRI
 * % Concentrations in the incoming Gas and Liquid flows of Nitri
 * % ========================================================
 * %1. Gas phase (O2 CO2 from Consu, O2 used for H2 transformation) :
 * CG0_3 = [Y0_0; 0];  % mol/l of O2 CO2 NH3 (NH3:null) from Consu
 */
mlfAssign(CG0_3, mlfVertcat(mclVv(*Y0_0, "Y0_0"), _mxarray6_, NULL));
/*
 * CG0_3(1) = CG0_3(1) - Y_1(1)/2/Gin_3; %O2 used for H2 transformation
 */
mclIntArrayAssign1(
  CG0_3,
  mclMinus(
    mclIntArrayRef1(mclVv(*CG0_3, "CG0_3"), 1),
    mclMrdivide(
      mclMrdivide(mclIntArrayRef1(mclVv(Y_1, "Y_1"), 1), _mxarray0_),
      mclVg(&Gin_3, "Gin_3"))),
  1);
/*
 * %2. Liquid phase (from Rhodo):
 * %CL0_3 = [C_O2_0; C_CO2_0; C_NH3_0; C_NO2_0; C_NO3_0; C_PO4_0; C_SO4_0];
 * CL0_3 = link2_3(X0_2)';
 */
mlfAssign(CL0_3, mlfCtranspose(mlfLink2_3(mclVv(*X0_2, "X0_2"))));
/*
 * CL0_3(1) = YO2_5;       % function of expected set point of O2_L at Spiru output
 */
mclIntArrayAssign1(CL0_3, mclVa(YO2_5, "YO2_5"), 1);
/*
 * CL0_3(NG_3+2) = spNO3_5*Fin_5/Fin_3;  % set point of NO3 at HPC output
 */
mclArrayAssign1(
```

```
  CL0_3,
  mclMrdivide(
    mclMtimes(mclVa(spNO3_5, "spNO3_5"), mclVg(&Fin_5, "Fin_5")),
    mclVg(&Fin_3, "Fin_3")),
  mclPlus(mclVg(&NG_3, "NG_3"), _mxarray0_));
/*
 * CL0_3(NG_3+1) = CL0_3(NG_3+1) + initNO2;  % NO2 from previous step
 */
mclArrayAssign1(
  CL0_3,
  mclPlus(
    mclArrayRef1(
      mclVv(*CL0_3, "CL0_3"), mclPlus(mclVg(&NG_3, "NG_3"), _mxarray1_)),
    mclVa(initNO2, "initNO2")),
  mclPlus(mclVg(&NG_3, "NG_3"), _mxarray1_));
/*
 * % Initial state and output vectors of Nitri (steady state)
 * % =========================================================
 * if idisp, CG0_3,CL0_3,end
 */
if (mlfTobool(mclVg(&idisp, "idisp"))) {
    mclPrintArray(mclVv(*CG0_3, "CG0_3"), "CG0_3");
    mclPrintArray(mclVv(*CL0_3, "CL0_3"), "CL0_3");
}
/*
 * [X0_3, Y0_3, dX0_3] = stesta_3( ...
 */
mlfAssign(
  X0_3,
  mlfStesta_3(
    Y0_3,
    dX0_3,
    mclVg(&NL_3, "NL_3"),
    mclVg(&NG_3, "NG_3"),
    mclVg(&NS_3, "NS_3"),
    mclVg(&NB_3, "NB_3"),
    mclVg(&NX_3, "NX_3"),
    mclVg(&NO_3, "NO_3"),
    mclVg(&NI_3, "NI_3"),
    mclVg(&NV_3, "NV_3"),
    mclVg(&Ae_3, "Ae_3"),
    mclVg(&Be_3, "Be_3"),
    mclVg(&Ce_3, "Ce_3"),
    mclVg(&De_3, "De_3"),
    mclVg(&E_3, "E_3"),
    mclVg(&WX_3, "WX_3"),
    mclVg(&WYG_3, "WYG_3"),
    mclVg(&WYL_3, "WYL_3"),
    mclVg(&iO2_3, "iO2_3"),
    mclVg(&iCO2_3, "iCO2_3"),
    mclVg(&iNH3_3, "iNH3_3"),
    mclVg(&iNO2_3, "iNO2_3"),
    mclVg(&iSub_3, "iSub_3"),
    mclVg(&iXNs_3, "iXNs_3"),
    mclVg(&iXNb_3, "iXNb_3"),
    mclVg(&iXag_3, "iXag_3"),
    mclVg(&KlNs_3, "KlNs_3"),
    mclVg(&KlNb_3, "KlNb_3"),
    mclVg(&KmNs_3, "KmNs_3"),
    mclVg(&KmNb_3, "KmNb_3"),
    mclVg(&mumax_3, "mumax_3"),
    mclVg(&maint_3, "maint_3"),
    mclVg(&Yx_3, "Yx_3"),
    mclVg(&Yx1_3, "Yx1_3"),
    mclVg(&Ym1_3, "Ym1_3"),
    mclVv(*CG0_3, "CG0_3"),
    mclVv(*CL0_3, "CL0_3"),
    mclVg(&RL_3, "RL_3"),
    mclVg(&indG_3, "indG_3"),
    mclVg(&indL_3, "indL_3")));
/*
```

```
 * NL_3, NG_3, NS_3, NB_3, NX_3, NO_3, NI_3, NV_3, ...
 * Ae_3, Be_3, Ce_3, De_3, E_3, WX_3, WYG_3, WYL_3, ...
 * iO2_3, iCO2_3, iNH3_3, iNO2_3, iSub_3, iXNs_3, iXNb_3, iXag_3, ...
 * KlNs_3, KlNb_3, KmNs_3, KmNb_3, mumax_3, maint_3, Yx_3, Yx1_3, Ym1_3, ...
 * CG0_3, CL0_3, RL_3, indG_3, indL_3);
 * if isempty(X0_3), break,  end
 */
if (mlfTobool(mlfIsempty(mclVv(*X0_3, "X0_3")))) {
  goto return_;
}
/*
 * if idisp,X0_3p = [X0_3(indL_3);X0_3(indL_3(3)+1);X0_3(indL_3(3)+2)],end
 */
if (mlfTobool(mclVg(&idisp, "idisp"))) {
  mlfAssign(
    &X0_3p,
    mlfVertcat(
      mclArrayRef1(mclVv(*X0_3, "X0_3"), mclVg(&indL_3, "indL_3")),
      mclArrayRef1(
        mclVv(*X0_3, "X0_3"),
        mclPlus(
          mclIntArrayRef1(mclVg(&indL_3, "indL_3"), 3), _mxarray1_)),
      mclArrayRef1(
        mclVv(*X0_3, "X0_3"),
        mclPlus(
          mclIntArrayRef1(mclVg(&indL_3, "indL_3"), 3), _mxarray0_)),
      NULL));
  mclPrintArray(mclVv(X0_3p, "X0_3p"), "X0_3p");
}
/*
 * %>>> SPIRU
 * % Initial concentrations in the incoming Gas and Liquid flows of Spiru
 * % ====================================================================
 * CGL0 = link3_4(Y0_3); % G and L from Nitri
 */
mlfAssign(&CGL0, mlfLink3_4(mclVv(*Y0_3, "Y0_3")));
/*
 * %1. Gas phase :
 * CG0_4 = CGL0(1:NG_4);   % mol/l
 */
mlfAssign(
  CG0_4,
  mclArrayRef1(
    mclVv(CGL0, "CGL0"), mlfColon(_mxarray1_, mclVg(&NG_4, "NG_4"), NULL)));
/*
 * %2. Liquid phase [O2 CO2 NO3 SO4 PO4]
 * CL0_4 = CGL0(NG_4+1:2*NG_4+NL_4);    % mol/l
 */
mlfAssign(
  CL0_4,
  mclArrayRef1(
    mclVv(CGL0, "CGL0"),
    mlfColon(
      mclPlus(mclVg(&NG_4, "NG_4"), _mxarray1_),
      mclPlus(
        mclMtimes(_mxarray0_, mclVg(&NG_4, "NG_4")), mclVg(&NL_4, "NL_4")),
      NULL)));
/*
 * % Initial light flux and state vector of Spiru (steady state)
 * % ============================================================
 * if idisp, CG0_4,CL0_4,end
 */
if (mlfTobool(mclVg(&idisp, "idisp"))) {
  mclPrintArray(mclVv(*CG0_4, "CG0_4"), "CG0_4");
  mclPrintArray(mclVv(*CL0_4, "CL0_4"), "CL0_4");
}
/*
 * [FR0_4, X0_4, Y0_4] = stesta_4( ...
 */
mlfAssign(
  FR0_4,
```

```
 mlfStesta_4(
   X0_4,
   Y0_4,
   mclVg(&NG_4, "NG_4"),
   mclVg(&NL_4, "NL_4"),
   mclVg(&NO_4, "NO_4"),
   mclVg(&vNB_4, "vNB_4"),
   mclVg(&vNM_4, "vNM_4"),
   mclVg(&Fin_4, "Fin_4"),
   mclVg(&VL_4, "VL_4"),
   mclVg(&Yx1_4, "Yx1_4"),
   mclVg(&A2_4, "A2_4"),
   mclVg(&B2_4, "B2_4"),
   mclVg(&GG1_4, "GG1_4"),
   mclVg(&GG2_4, "GG2_4"),
   mclVv(*CG0_4, "CG0_4"),
   mclVv(*CL0_4, "CL0_4"),
   mclIntArrayRef1(mclVa(EE, "EE"), 1),
   mclVg(&FRmin_4, "FRmin_4"),
   mclVg(&FRmaxc_4, "FRmaxc_4"),
   mclVg(&fI_4, "fI_4"),
   mclVg(&zPC_4, "zPC_4"),
   mclVg(&zCH_4, "zCH_4"),
   mclVg(&RT_4, "RT_4"),
   mclVg(&Ea_4, "Ea_4"),
   mclVg(&Es_4, "Es_4"),
   mclVg(&muM_4, "muM_4"),
   mclVg(&muEPS_4, "muEPS_4"),
   mclVg(&Kj_4, "Kj_4"),
   mclVg(&KjEPS_4, "KjEPS_4"),
   mclVg(&Fmin_4, "Fmin_4"),
   mclVg(&zmin_I_4, "zmin_I_4"),
   mclVg(&KSNO3_4, "KSNO3_4"),
   mclVg(&KSSO4_4, "KSSO4_4"),
   mclVg(&KSPO4_4, "KSPO4_4"),
   mclVg(&KSPC_4, "KSPC_4")));
/*
 * NG_4, NL_4, NO_4, vNB_4, vNM_4, Fin_4, VL_4, Yx1_4, ...
 * A2_4, B2_4, GG1_4, GG2_4, CG0_4, CL0_4, EE(1), ...
 * FRmin_4, FRmaxc_4, fI_4, zPC_4, zCH_4, ...
 * RT_4, Ea_4, Es_4, muM_4, muEPS_4, Kj_4, KjEPS_4, Fmin_4, zmin_I_4, ...
 * KSNO3_4, KSSO4_4, KSPO4_4, KSPC_4);
 * if (isempty(FR0_4) | isempty(X0_4)), break,      end
 */
 {
   mxArray * a_ = mclInitialize(mlfIsempty(mclVv(*FR0_4, "FR0_4")));
   if (mlfTobool(a_)
       || mlfTobool(mclOr(a_, mlfIsempty(mclVv(*X0_4, "X0_4"))))) {
      mxDestroyArray(a_);
      goto return_;
   } else {
      mxDestroyArray(a_);
   }
 }
 /*
 * if idisp, X0_4, FR0_4, end
 */
 if (mlfTobool(mclVg(&idisp, "idisp"))) {
    mclPrintArray(mclVv(*X0_4, "X0_4"), "X0_4");
    mclPrintArray(mclVv(*FR0_4, "FR0_4"), "FR0_4");
 }
 /*
 *
 * %>>> HPC
 * % Concentrations in the incoming Gas and Liquid flows of HPC
 * % ========================================================
 * %1. Gas phase [O2 CO2 NH3]:
 * indN1 = (NB_3+2)*NG_3; % index of NH3 G in output vector of Nitri
 */
 mlfAssign(
  &indN1,
```

```
  mclMtimes(
    mclPlus(mclVg(&NB_3, "NB_3"), _mxarray0_), mclVg(&NG_3, "NG_3")));
/*
 * %CG0_5 = [Y0_4(1)+addO2/Gin_5;Y0_4(NG_4)+addCO2/Gin_5;Y0_3(indN1)]; % O2 CO2 NH3
 * CG0_5 = [Y0_4(1:NG_4)+[addO2;addCO2]/Gin_5;Y0_3(indN1)]; % O2 CO2 NH3
 */
mlfAssign(
  CG0_5,
  mlfVertcat(
    mclPlus(
      mclArrayRef1(
        mclVv(*Y0_4, "Y0_4"),
        mlfColon(_mxarray1_, mclVg(&NG_4, "NG_4"), NULL)),
      mclMrdivide(
        mlfVertcat(mclVa(addO2, "addO2"), mclVa(addCO2, "addCO2"), NULL),
        mclVg(&Gin_5, "Gin_5")))),
    mclArrayRef1(mclVv(*Y0_3, "Y0_3"), mclVv(indN1, "indN1")),
    NULL));
/*
 * %2. Liquid phase [O2 CO2 NH3 NO3 SO4 PO4]
 * CL0_5 = link245(X0_2,Y0_3,X0_4,share2,NG_2,Kdis_2,...
 */
mlfAssign(
  CL0_5,
  mlfLink245(
    mclVv(*X0_2, "X0_2"),
    mclVv(*Y0_3, "Y0_3"),
    mclVv(*X0_4, "X0_4"),
    mclVa(share2, "share2"),
    mclVg(&NG_2, "NG_2"),
    mclVg(&Kdis_2, "Kdis_2"),
    mclVg(&NG_3, "NG_3"),
    mclVg(&ind_3u5, "ind_3u5"),
    mclVg(&Kdis_3, "Kdis_3"),
    mclVg(&NG_4, "NG_4"),
    mclVg(&NL_4, "NL_4"),
    mclVg(&Kdis_4, "Kdis_4"),
    mclVg(&NG_5, "NG_5"),
    mclVg(&Kdis_5, "Kdis_5")));
/*
 * NG_3,ind_3u5,Kdis_3,NG_4,NL_4,Kdis_4,NG_5,Kdis_5);
 * % Initial state and variation rates vectors of HPC (steady state)
 * % ================================================================
 * if idisp, CG0_5,CL0_5,end
 */
if (mlfTobool(mclVg(&idisp, "idisp"))) {
  mclPrintArray(mclVv(*CG0_5, "CG0_5"), "CG0_5");
  mclPrintArray(mclVv(*CL0_5, "CL0_5"), "CL0_5");
}
/*
 * [X0_5, ri_5] = stesta_5(...
 */
mlfAssign(
  &X0_5,
  mlfStesta_5(
    ri_5,
    mclVg(&NG_5, "NG_5"),
    mclVg(&NL_5, "NL_5"),
    mclVg(&NS_5, "NS_5"),
    mclVg(&VG_5, "VG_5"),
    mclVg(&VL_5, "VL_5"),
    mclVg(&Fin_5, "Fin_5"),
    mclVg(&Yx1e_5, "Yx1e_5"),
    mclVg(&Yx1w_5, "Yx1w_5"),
    mclVg(&Diete_5, "Diete_5"),
    mclVg(&Dietw_5, "Dietw_5"),
    mclVa(cover, "cover"),
    mclVg(&A2_5, "A2_5"),
    mclVg(&B2_5, "B2_5"),
    mclVv(*CG0_5, "CG0_5"),
    mclVv(*CL0_5, "CL0_5")));
```

```
/*
 * NG_5, NL_5, NS_5, VG_5, VL_5, Fin_5, Yx1e_5, Yx1w_5, ...
 * Diete_5, Dietw_5, cover, A2_5, B2_5, CG0_5, CL0_5);
 * Y0_5=X0_5;
 */
mlfAssign(Y0_5, mclVv(X0_5, "X0_5"));
/*
 * if idisp, Y0_5, end
 */
if (mlfTobool(mclVg(&idisp, "idisp"))) {
   mclPrintArray(mclVv(*Y0_5, "Y0_5"), "Y0_5");
}
/*
 *
 */
return_:
mclValidateOutput(CG0_0, 1, nargout_, "CG0_0", "simpro");
mclValidateOutput(*Y0_0, 2, nargout_, "Y0_0", "simpro");
mclValidateOutput(*Y0_1, 3, nargout_, "Y0_1", "simpro");
mclValidateOutput(*CG0_2, 4, nargout_, "CG0_2", "simpro");
mclValidateOutput(*CL0_2, 5, nargout_, "CL0_2", "simpro");
mclValidateOutput(*FR0_2, 6, nargout_, "FR0_2", "simpro");
mclValidateOutput(*X0_2, 7, nargout_, "X0_2", "simpro");
mclValidateOutput(*Y0_2, 8, nargout_, "Y0_2", "simpro");
mclValidateOutput(*CG0_3, 9, nargout_, "CG0_3", "simpro");
mclValidateOutput(*CL0_3, 10, nargout_, "CL0_3", "simpro");
mclValidateOutput(*X0_3, 11, nargout_, "X0_3", "simpro");
mclValidateOutput(*Y0_3, 12, nargout_, "Y0_3", "simpro");
mclValidateOutput(*dX0_3, 13, nargout_, "dX0_3", "simpro");
mclValidateOutput(*CG0_4, 14, nargout_, "CG0_4", "simpro");
mclValidateOutput(*CL0_4, 15, nargout_, "CL0_4", "simpro");
mclValidateOutput(*FR0_4, 16, nargout_, "FR0_4", "simpro");
mclValidateOutput(*X0_4, 17, nargout_, "X0_4", "simpro");
mclValidateOutput(*Y0_4, 18, nargout_, "Y0_4", "simpro");
mclValidateOutput(*CG0_5, 19, nargout_, "CG0_5", "simpro");
mclValidateOutput(*CL0_5, 20, nargout_, "CL0_5", "simpro");
mclValidat eOutput(*Y0_5, 21, nargout_, "Y0_5", "simpro");
mclValidateOutput(*ri_5, 22, nargout_, "ri_5", "simpro");
mxDestroyArray(ans);
mxDestroyArray(Y_1);
mxDestroyArray(NY1);
mxDestroyArray(NY);
mxDestroyArray(fm_CO2_2);
mxDestroyArray(fm_NH3_2);
mxDestroyArray(fm_Ac_2);
mxDestroyArray(fm_Bu_2);
mxDestroyArray(ind);
mxDestroyArray(aa);
mxDestroyArray(cX0_2);
mxDestroyArray(dX0_2);
mxDestroyArray(X0_3p);
mxDestroyArray(CGL0);
mxDestroyArray(indN1);
mxDestroyArray(X0_5);
mxDestroyArray(cover);
mxDestroyArray(share2);
mxDestroyArray(spNO3_5);
mxDestroyArray(YCO2_5);
mxDestroyArray(YO2_5);
mxDestroyArray(Yx_AcOH);
mxDestroyArray(addO2);
mxDestroyArray(addCO2);
mxDestroyArray(initLiqueL);
mxDestroyArray(initBuOH);
mxDestroyArray(initAcOH);
mxDestroyArray(initNO2);
mxDestroyArray(initNH3);
mxDestroyArray(rCO2_0);
mxDestroyArray(rO2_0);
mxDestroyArray(fm_CO2_0);
mxDestroyArray(fm_O2_0);
```

```
    mxDestroyArray(EE);
    mclSetCurrentLocalFunctionTable(save_local_function_table_);
    return CG0_0;
}
```