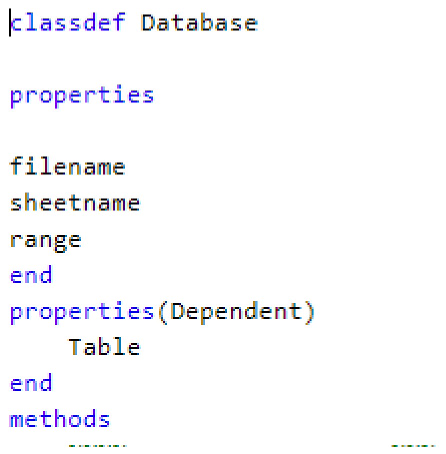
A1 1. GENERIC SYNTAX FOR CLASSES ANDD OBJECTS

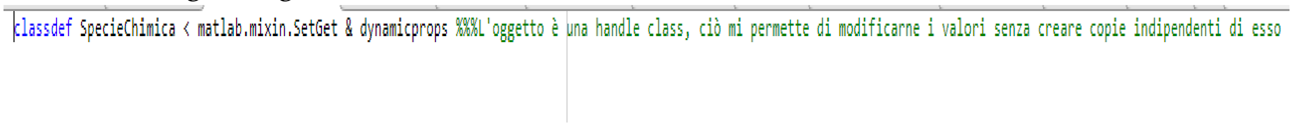


```
classdef Database

properties

filename
sheetname
range
end
properties(Dependent)
    Table
end
methods
```

A1 Figure 1 Database object class definition

The class/object can be constructed by defining the following parameters:

1) "classdef": as in the case of defining a function, the class must be initialized by typing *classdef* and closing with an *end*; everything between these two headlines will define the class; we start from the name of the class itself, which will be used to invoke the object in question whenever it is needed.
   In MATLAB, there are two types of classes: handle and value; all classes are initialized as value unless otherwise indicated; the main difference between a handle class and a value class is that the latter does not share the data stored within with copies of itself; this as it will be show later is a problem, especially if the objective is to implement a method of copying objects; So if one wants to initialize a class / object as a handle class one will need to import into it other built in class at the beginning:



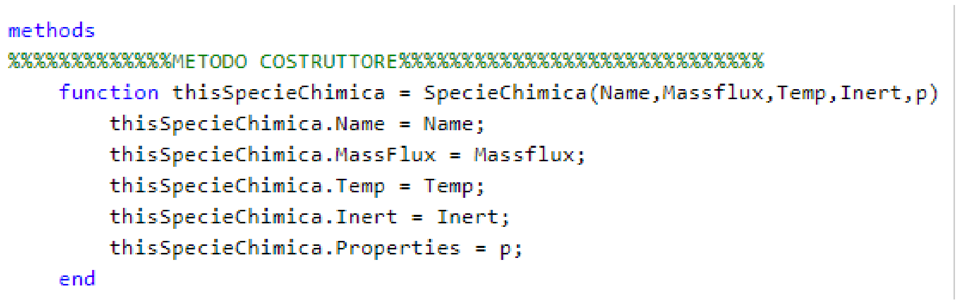
```
classdef SpecieChimica < matlab.mixin.SetGet & dynamicprops %%%L'oggetto è una handle class, ciò mi permette di modificarne i valori senza creare copie indipendenti di esso
```

A1 Figure 2 Imported MATLAB classes for the SPECIE CHIMICA object class

2) "properties": are the "static" properties of the object, i.e. those that are required for the creation of the object itself; they are obviously arbitrary and chosen by the user in the design phase; They will then be called in the constructor method.
3) "properties: dependent": by enabling the *dependent* property, the following property group is calculated during the creation of the object (if you invoke it); These properties depend precisely on static properties.
4) "methods": are the internal functions of the class, which are called automatically when the object is created, such as the constructor method, or when explicitly called by the user.

1.2 THE CONSTRUCTOR METHOD

It is the most important method of any class, since it is responsible for initializing it; It is extremely simple to define, as it is only necessary to associate a corresponding input value to each defined static property, as shown in the Figure A1 3:



```
methods
%%%%%%%%%%%%%METODO COSTRUTTORE%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    function thisSpecieChimica = SpecieChimica(Name,Massflux,Temp,Inert,p)
        thisSpecieChimica.Name = Name;
        thisSpecieChimica.MassFlux = Massflux;
        thisSpecieChimica.Temp = Temp;
        thisSpecieChimica.Inert = Inert;
        thisSpecieChimica.Properties = p;
    end
```

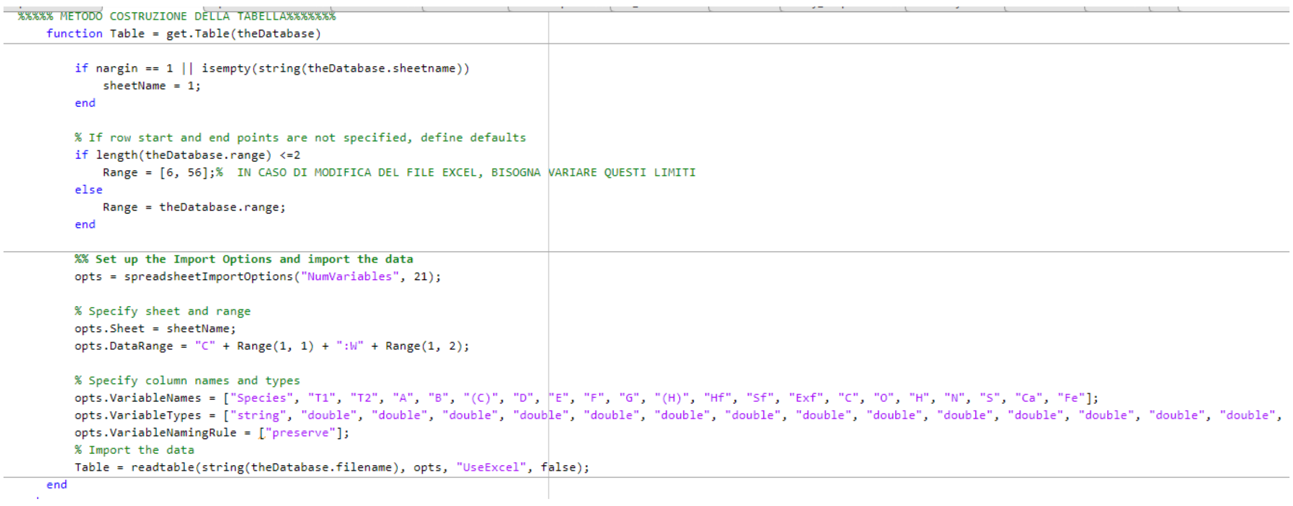
A1 Figure 3 Constructor method for the SPECIE CHIMICA object class

A1 2. DATABASE CLASS

2.1 GENERIC CLASS PROPERTIES AND STRUCTURE OF EXCEL FILE

The intent of this class is to make the Excel table of chemical species data "interactive". This allows a simpler and more immediate integration into other objects, especially in the iterative search for specific compounds under certain conditions.

The most important dependent property is the table itself, which is extracted using MATLAB's slightly modified export function:



```
%%%%% METODO COSTRUZIONE DELLA TABELLA%%%%%%%
    function Table = get.Table(theDatabase)

        if nargin == 1 || isempty(string(theDatabase.sheetname))
            sheetName = 1;
        end

        % If row start and end points are not specified, define defaults
        if length(theDatabase.range) <=2
            Range = [6, 56];%  IN CASO DI MODIFICA DEL FILE EXCEL, BISOGNA VARIARE QUESTI LIMITI
        else
            Range = theDatabase.range;
        end

        %% Set up the Import Options and import the data
        opts = spreadsheetImportOptions("NumVariables", 21);

        % Specify sheet and range
        opts.Sheet = sheetName;
        opts.DataRange = "C" + Range(1, 1) + ":W" + Range(1, 2);

        % Specify column names and types
        opts.VariableNames = ["Species", "T1", "T2", "A", "B", "(C)", "D", "E", "F", "G", "(H)", "Hf", "Sf", "Exf", "C", "O", "H", "N", "S", "Ca", "Fe"];
        opts.VariableTypes = ["string", "double", "double", "double", "double", "double", "double", "double", "double", "double", "double", "double", "double", "double", "double",
        opts.VariableNamingRule = ["preserve"];
        % Import the data
        Table = readtable(string(theDatabase.filename), opts, "UseExcel", false);
    end
```

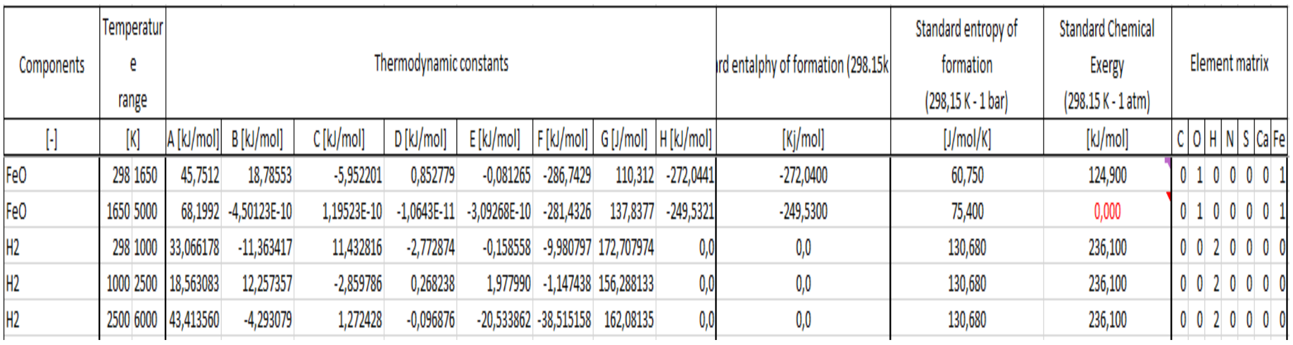
A1 Figure 4 Table Import function

The table stored in the object is itself a built in "table" object, an extension of the cell array.

Suppose you want to add a new chemical species to the database; to do this one will have to edit the Excel file itself, adding a row to the pre-existing table; this must then be followed by a simple modification of the ranges as indicated in the comments to the code (simply add a unit to the right limit of the range).

If one were to add more than one line about the same chemical species, care must be taken to respect the general structure of the Excel file; this is necessary otherwise the *getvec* method, discussed in the next paragraph, may not work.

Each chemical compound must be added to the Excel indicating its formula, the temperature range for which the Shomate equation constants are valid, the various above-mentioned constants, enthalpies and standard entropies at 25 C, and atomic compositions as shown; if multiple rows of the same compound are added, but with different temperature ranges, it is extremely important that they are all added in such a way that they are continuous, following a progressive order of temperature ranges, as shown in Figure A1 5



Components	Temperature range		Thermodynamic constants								rd entalphy of formation (298.15k	Standard entropy of formation (298,15 K - 1 bar)	Standard Chemical Exergy (298.15 K - 1 atm)	Element matrix						
[-]	[K]		A [kJ/mol]	B [kJ/mol]	C [kJ/mol]	D [kJ/mol]	E [kJ/mol]	F [kJ/mol]	G [J/mol]	H [kJ/mol]	[Kj/mol]	[J/mol/K]	[kJ/mol]	C	O	H	N	S	Ca	Fe
FeO	298	1650	45,7512	18,78553	-5,952201	0,852779	-0,081265	-286,7429	110,312	-272,0441	-272,0400	60,750	124,900	0	1	0	0	0	0	1
FeO	1650	5000	68,1992	-4,50123E-10	1,19523E-10	-1,0643E-11	-3,09268E-10	-281,4326	137,8377	-249,5321	-249,5300	75,400	0,000	0	1	0	0	0	0	1
H2	298	1000	33,066178	-11,363417	11,432816	-2,772874	-0,158558	-9,980797	172,707974	0,0	0,0	130,680	236,100	0	0	2	0	0	0	0
H2	1000	2500	18,563083	12,257357	-2,859786	0,268238	1,977990	-1,147438	156,288133	0,0	0,0	130,680	236,100	0	0	2	0	0	0	0
H2	2500	6000	43,413560	-4,293079	1,272428	-0,096876	-20,533862	-38,515158	162,08135	0,0	0,0	130,680	236,100	0	0	2	0	0	0	0

A1 Figure 5 Extract of the Database's table.

If you want to create an Excel file from scratch, use the provided Excel file as a reference.

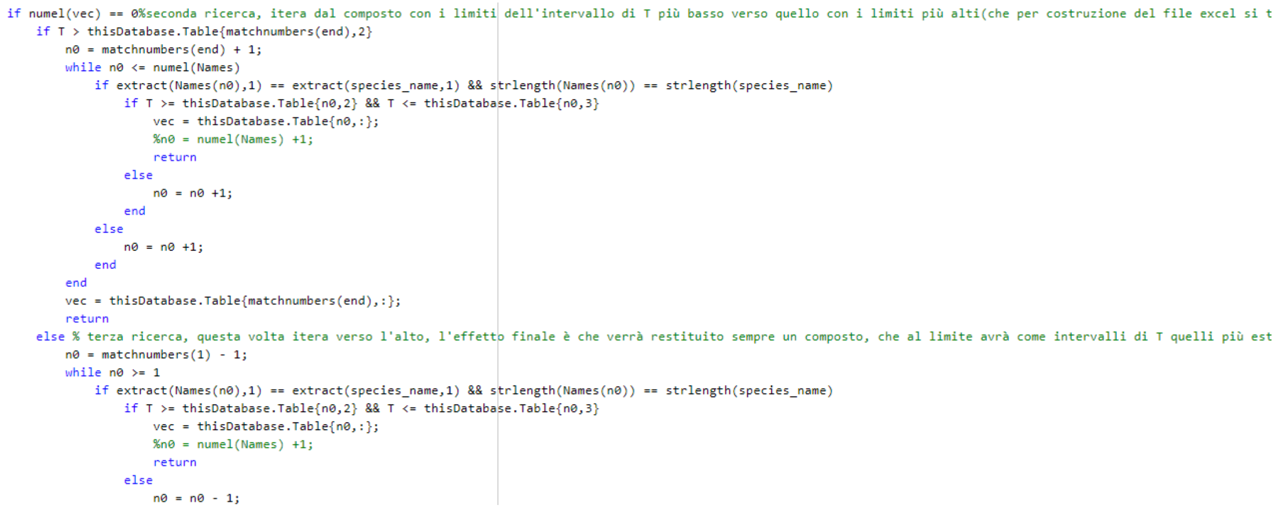
the most important methods and how they work will be mentioned below:

2.2 GETVEC

It is the method of searching for the chemical species; it is divided into two parts:

1) Search by direct string: using the user-provided string, all the indices of the vector "species" (first column of the Excel table) that contain the aforementioned string are found. An iteration is made on these indices, where it is checked whether the supplied temperature is within the range indicated in the table.
2) Check extreme temperature: if the first search fails, unless the string itself has been mistyped, the reason is certainly due to a temperature out of range; the thermal ranges are then verified; thanks to the structure of the Excel file, all species with the same formula are grouped together; therefore we iterate once towards the "bottom" of the table, that is, towards thermal intervals at higher temperatures, and if this fails we go "up"; at the same time we compare the shape of the given string with that of the Excel file, taking care that they match together at least in the final two

letters; If all this fails, unless there are errors in the formula, then the program returns the properties taken at one of the extremes, that is, with the thermal interval with lower or higher temperatures depending on where the program stops.



```
if numel(vec) == 0%seconda ricerca, itera dal composto con i limiti dell'intervallo di T più basso verso quello con i limiti più alti(che per costruzione del file excel si t
    if T > thisDatabase.Table{matchnumbers(end),2}
        n0 = matchnumbers(end) + 1;
        while n0 <= numel(Names)
            if extract(Names(n0),1) == extract(species_name,1) && strlength(Names(n0)) == strlength(species_name)
                if T >= thisDatabase.Table{n0,2} && T <= thisDatabase.Table{n0,3}
                    vec = thisDatabase.Table{n0,:};
                    %n0 = numel(Names) +1;
                    return
                else
                    n0 = n0 +1;
                end
            else
                n0 = n0 +1;
            end
        end
        vec = thisDatabase.Table{matchnumbers(end),:};
        return
    else % terza ricerca, questa volta itera verso l'alto, l'effetto finale è che verrà restituito sempre un composto, che al limite avrà come intervalli di T quelli più est
        n0 = matchnumbers(1) - 1;
        while n0 >= 1
            if extract(Names(n0),1) == extract(species_name,1) && strlength(Names(n0)) == strlength(species_name)
                if T >= thisDatabase.Table{n0,2} && T <= thisDatabase.Table{n0,3}
                    vec = thisDatabase.Table{n0,:};
                    %n0 = numel(Names) +1;
                    return
                else
                    n0 = n0 - 1;
```

A1 Figure 6 getvec algorithm

The author strongly recommends that you do not modify this algorithm in any way.

2.3 HOW TO CREATE A DATABASE

Assuming one wants to use the Excel provided, to create a database object the general syntax for generating objects in MATLAB will have to be used, so in the console it must be typed:

"A = Database("Thermodynamics_data2.xlsx","Date",[6,56])"

Where the first term in parentheses indicates the name of the Excel file, the second the name of the datasheet of the Excel file.

A1 3. CHEMICAL SPECIES CLASS

It is the object class on which the entire model is based; it is also a handle class, therefore it has a copy method; this is necessary because like all handle classes, if you change a value of a property to an instance of it, the change is transmitted to all of them; this is not ideal if you want to iterate on several objects of this type. The main parameters are:

1) compound formula
2) temperature
3) whether or not it should be considered inert in the reaction
4) initial moles, which can be inserted as a flow rate or as a static quantity; It is an optional parameter, and if you do not enter it the object considered will be treated as a product.

3.1 CALCDG METHOD

As can be seen from the properties, the dG, the Gibbs free energy, is only initialized but not calculated at the time of creation of the object; to calculate it you have to invoke the calcDG method, which returns the standard free energy at a given temperature. It is calculated in turn by invoking internally the two methods intgH and intgS, which calculate the enthalpy and entropy integrals starting from an ambient temperature of 25 C to the desired temperature.

3.2 INTGH AND INTGS METHOD

The integral is calculated based on the constants of the Shomate equation provided by the NIST database and collected in the local database in Excel; both rely on the getvec method, so they are dependent on the Database object.

3.3 HOW TO CREATE A CHEMICAL SPECIES OBJECT

Instead of using the syntax for generating objects, it is better to use the appropriate *createEl* function that takes as input the same parameters necessary for the creation of the SPECIE_CHIMICA object; so if for example one wanted to create the compound CH4 at 823 Kelvin non-inert and with 1,070 kmol / h one will have to write in the console:

"CH4 = createEl(A,"CH4",823.15,false,1.0707)"

Where with A is the variable where the reference DATABASE object is saved (in this case, the name of the variable is arbitrarily chosen by the user).

Alternatively, if one want to create the same compound but would like to consider it as a reaction product, one will write:

"CH4 = createEl(A,"CH4",823.15,false)"

If one wanted to create an inert n, for example nitrogen, the syntax is as follows:

N2 = createEl(A,"N2",1123,true,3.21227)

A1 4. MING2 AND MING3 FUNCTION

4.1 REQUIREMENTS AND INPUTS

Before one can use any function of the model, it must be made sure that the "global optimization toolbox" is installed in your MATLAB client, which brings with it features such as searching for global minimums and maximums etc.

MING2 is the function that must be called if you want to calculate the equilibrium composition of one or more reactions under ISOTHERMAL conditions (MING3 also calculates the heat of reaction). The inputs to be provided are:

1) database object, from which the various properties of the objects are fished.

2) reaction/reactor pressure.

3) at least two objects of chemical species class; These can be introduced as input in two different ways: listing them one by one.

"test_3 = minG2(A,1,CH4,CO,CO2,H2,H2O)"

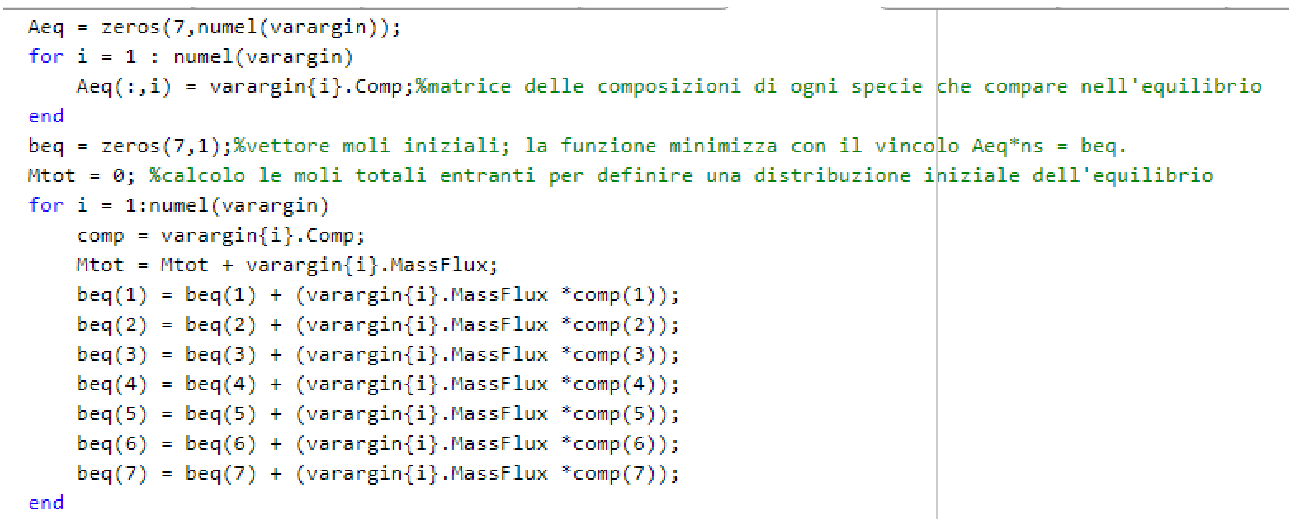
Or group the individual objects inside a cell array and use it as input:

"array = {CH4,CO,CO2,H2,H2O}"

IMPORTANT: do not enter INERT objects; the program is not designed to run with them, as it makes no sense to "react" an inert object.

4.2 GENERAL FUNCTIONING OF THE PROGRAMM

The program is essentially based on the function and *fmincon* of MATLAB, so much of it is dedicated to extracting and supplying input data to it; the matrix Aeq and the vector of the initial moles beq are then defined, which constitute the mass constraint:



```
Aeq = zeros(7,numel(varargin));
for i = 1 : numel(varargin)
    Aeq(:,i) = varargin{i}.Comp;%matrice delle composizioni di ogni specie che compare nell'equilibrio
end
beq = zeros(7,1);%vettore moli iniziali; la funzione minimizza con il vincolo Aeq*ns = beq.
Mtot = 0; %calcolo le moli totali entranti per definire una distribuzione iniziale dell'equilibrio
for i = 1:numel(varargin)
    comp = varargin{i}.Comp;
    Mtot = Mtot + varargin{i}.MassFlux;
    beq(1) = beq(1) + (varargin{i}.MassFlux *comp(1));
    beq(2) = beq(2) + (varargin{i}.MassFlux *comp(2));
    beq(3) = beq(3) + (varargin{i}.MassFlux *comp(3));
    beq(4) = beq(4) + (varargin{i}.MassFlux *comp(4));
    beq(5) = beq(5) + (varargin{i}.MassFlux *comp(5));
    beq(6) = beq(6) + (varargin{i}.MassFlux *comp(6));
    beq(7) = beq(7) + (varargin{i}.MassFlux *comp(7));
end
```

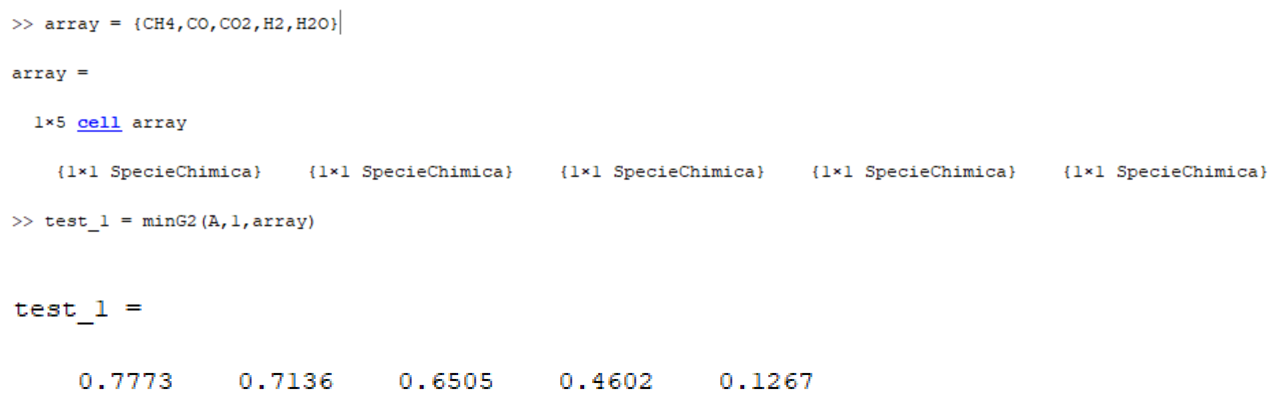
A1 Figure 7 Construction of the mass constraint

An initial value is then defined for the solution, which is obtained by making a simple arithmetic mean between the initial total moles and the number of objects in the reaction.

Finally, there is the minimization part where the syntax introduced by the toolbox is used; essentially, here two additional objects must be defined, optimotions and problem; refer to the specific documentation of the toolbox if one wants to change something, but there should be no need.

4.3 OUTPUT

The output is a row vector consisting of the molar composition at equilibrium of the mixture, in the case of minG3 the last value represents the heat of reaction. The order of appearance of the various elements corresponds to the order in which the chemical compounds were introduced at the beginning:



```
>> array = {CH4,CO,CO2,H2,H2O}

array =

  1×5 cell array

    {1×1 SpecieChimica}    {1×1 SpecieChimica}    {1×1 SpecieChimica}    {1×1 SpecieChimica}    {1×1 SpecieChimica}

>> test_1 = minG2(A,1,array)


test_1 =

    0.7773    0.7136    0.6505    0.4602    0.1267
```

A1 Figure 8 Typical results format of a calculation

A1 5 FINDCOMP AND FINDCOMPV2 FUNCTION

FINDCOMP is the first function that must be called if one wants to find the equilibrium composition of a reaction in an adiabatic reactor.

5.1 REQUIREMENTS AND INPUTS

Before one can use any model function, it must be ensured that the "global optimization toolbox" is installed in your MATLAB client, because the MING2 function is called and dependent on it during its operation.

The necessary inputs are:

1) Database object.
2) Reaction pressure.
3) At least two chemical objects; These can be entered in the two ways discussed in 4.1.

It is possible to enter inert objects, and it is possible to enter objects with "final" temperature values different from each other (in case you want to simulate mass flows entering at different temperatures in the reactor, as in the case of regeneration. Note that the exchanged heat won’t be factored in the heat of reaction calculation).

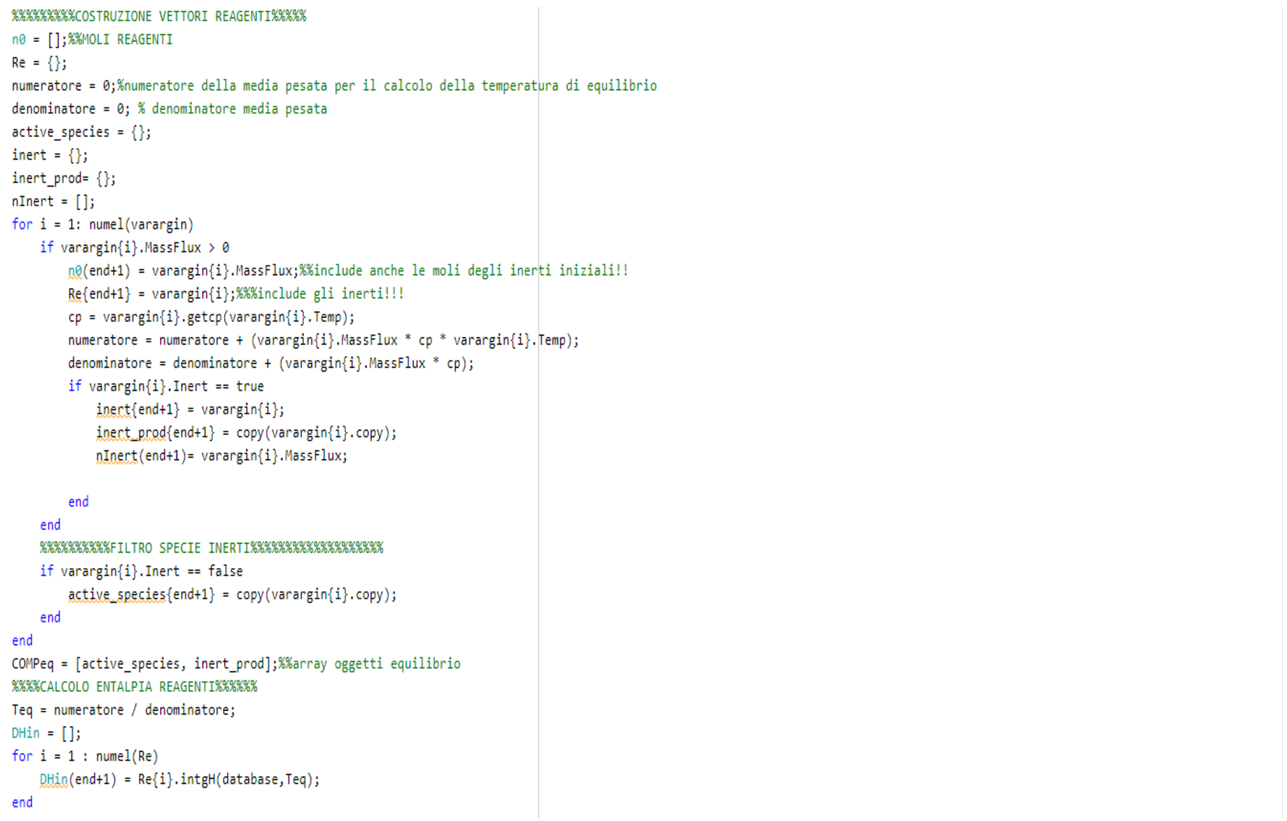
5.2 GENERAL FUNCTIONING OF THE PROGRAM

Since it is in the adiabatic case, the mass constraint is no longer sufficient; It is therefore necessary to introduce the enthalpy constraint; for the reasons discussed in Chapter 4, a while cycle is needed where the constraint is applied and used to find the unknown temperature that satisfies it; this temperature is then used as input to MING2, which will return a molar distribution at that temperature, used in the enthalpy constraint. The while loop is interrupted if:

1) Error tolerance is achieved.
2) Maximum number of iterations is reached.
3) The average solution is found, in the case of oscillation of the solution.

5.3 INPUT PROCESSING

Before the while loop, inputs must be processed to be used. There is therefore an initial filter, which distinguishes between active species (which participate in the reaction) and inert ones; the program produces a copy of these two groups so that it can use them in the iteration without changing the initial objects; In addition, the temperature is also normalized: if one has objects as input at different temperatures, an equilibrium temperature will be set; this is obtained from a simple weighted average, using a fixed specific heat for each compound:

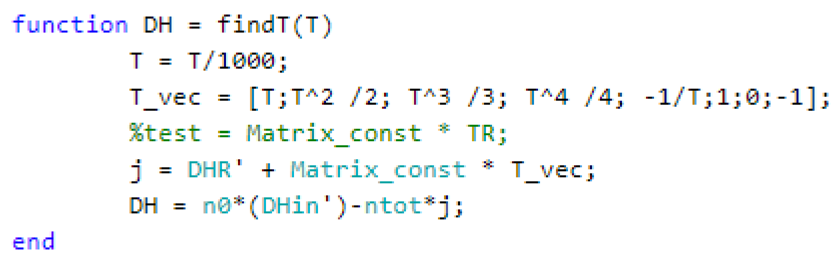


```
%%%%%%%%%COSTRUZIONE VETTORI REAGENTI%%%%%
n0 = [];%%MOLI REAGENTI
Re = {};
numeratore = 0;%numeratore della media pesata per il calcolo della temperatura di equilibrio
denominatore = 0; % denominatore media pesata
active_species = {};
inert = {};
inert_prod= {};
nInert = [];
for i = 1: numel(varargin)
    if varargin{i}.MassFlux > 0
        n0(end+1) = varargin{i}.MassFlux;%%include anche le moli degli inerti iniziali!!
        Re{end+1} = varargin{i};%%%include gli inerti!!!
        cp = varargin{i}.getcp(varargin{i}.Temp);
        numeratore = numeratore + (varargin{i}.MassFlux * cp * varargin{i}.Temp);
        denominatore = denominatore + (varargin{i}.MassFlux * cp);
        if varargin{i}.Inert == true
            inert{end+1} = varargin{i};
            inert_prod{end+1} = copy(varargin{i}.copy);
            nInert(end+1)= varargin{i}.MassFlux;

        end
    end
    %%%%%%%%%%FILTRO SPECIE INERTI%%%%%%%%%%%%%%%%%%
    if varargin{i}.Inert == false
        active_species{end+1} = copy(varargin{i}.copy);
    end
end
COMPeq = [active_species, inert_prod];%%array oggetti equilibrio
%%%%CALCOLO ENTALPIA REAGENTI%%%%%%
Teq = numeratore / denominatore;
DHin = [];
for i = 1 : numel(Re)
    DHin(end+1) = Re{i}.intgH(database,Teq);
end
```

A1 Figure 9 FindComp data pre-processing

5.4 WHILE CYCLE

At each iteration, since a new temperature is found, all the thermochemical properties of the objects involved (inert and non-inert) must be updated, only then can MING2 be invoked; the constraint is then applied, which is treated as a system to be solved for T; the system is solved via the built-in *lsqnonlin* function which requires a target function, findT:

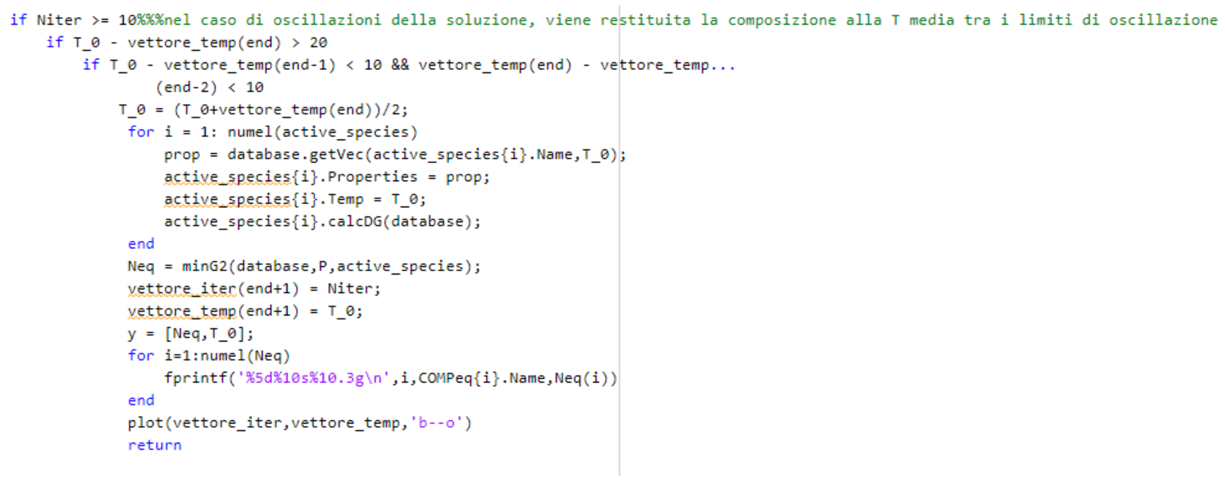


```
function DH = findT(T)
        T = T/1000;
        T_vec = [T;T^2 /2; T^3 /3; T^4 /4; -1/T;1;0;-1];
        %test = Matrix_const * TR;
        j = DHR' + Matrix_const * T_vec;
        DH = n0*(DHin')-ntot*j;
end
```

A1 Figure 10 Enthalpic constraint

Finally, the value of T is updated.

It may happen that the while loop returns temperature values (and therefore molar compositions) that oscillate between two extremes; to return a "credible" value, the program after ten iterations checks if there have been oscillations, checking how the temperature values have changed, and in this case takes as a solution the average of the extreme temperatures of the oscillation, with the respective molar composition:



```
if Niter >= 10%%%nel caso di oscillazioni della soluzione, viene restituita la composizione alla T media tra i limiti di oscillazione
    if T_0 - vettore_temp(end) > 20
        if T_0 - vettore_temp(end-1) < 10 && vettore_temp(end) - vettore_temp...
                (end-2) < 10
            T_0 = (T_0+vettore_temp(end))/2;
             for i = 1: numel(active_species)
                 prop = database.getVec(active_species{i}.Name,T_0);
                 active_species{i}.Properties = prop;
                 active_species{i}.Temp = T_0;
                 active_species{i}.calcDG(database);
             end
             Neq = minG2(database,P,active_species);
             vettore_iter(end+1) = Niter;
             vettore_temp(end+1) = T_0;
             y = [Neq,T_0];
             for i=1:numel(Neq)
                 fprintf('%5d%10s%10.3g\n',i,COMPeq{i}.Name,Neq(i))
             end
             plot(vettore_iter,vettore_temp,'b--o')
             return
```

A1 Figure 11 Oscillation control

The vettore_iter and vettore_temp vectors are "system" vectors, that is, they are used only to monitor the program during debugging and to monitor the trend of temperatures found during iterations at the end of the program.

5.5 OUTPUT

The program returns a row vector composed of the molar composition found, and the corresponding temperature; for V2 it also returns the heat of reaction. The same rules of reading order as mentioned in 4.3 apply.

A1 6. FUNCTION FFA_MING

This is the second function to be used for the adiabatic case, in case one wants to refine the solution or simply compare it. It is based on the heuristic method of fireflies, and as such, its accuracy depends on the number of iterations and size of the initial population. For a detailed discussion of the algorithm, please refer to the appropriate paragraph in Chapter 4.

6.1 REQUIREMENTS AND INPUTS

Before one can use any model function, it must be ensured that the "global optimization toolbox" is installed in your MATLAB client, because the FINDCOMP function is called and dependent on it during its operation.

The necessary inputs are:

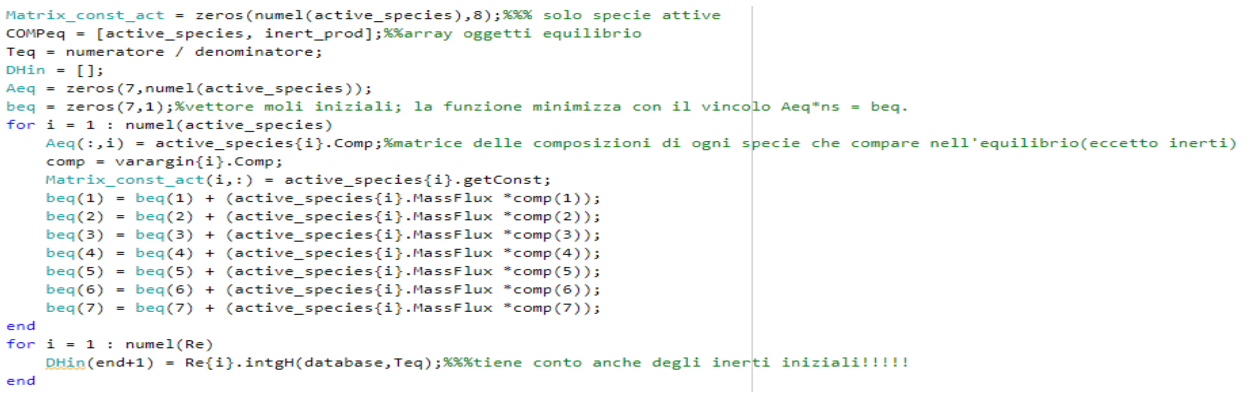
1) Database object.
2) Reaction pressure.
3) At least two chemical objects; These can be entered in the two ways discussed in 4.1.

It is possible to enter inert objects, and it is possible to enter objects with "final" temperature values different from each other (in case one want to simulate mass flows entering at different temperatures in the reactor, as in the case of regeneration. Note that the heat exchanged won’t be factored in the heat of formation calculation).

6.2 GENERAL FUNCTIONING OF THE PROGRAM

As anticipated, it is based on the heuristic method of fireflies; the program itself does not calculate the solution, but has the function of initializing the values, pre-processing them, and defining the problem, i.e. the objective function and constraints; for this reason almost all of the program is identical to the pre-processing part of FINDCOMP. Please refer to 5.2 for further clarification.

The main differences are in the fact that there is no update of the thermochemical properties of the inputs; therefore, the specific heat is calculated on the basis of the initial temperature (or Teq), from which a fixed matrix of the specific heat results, which is used

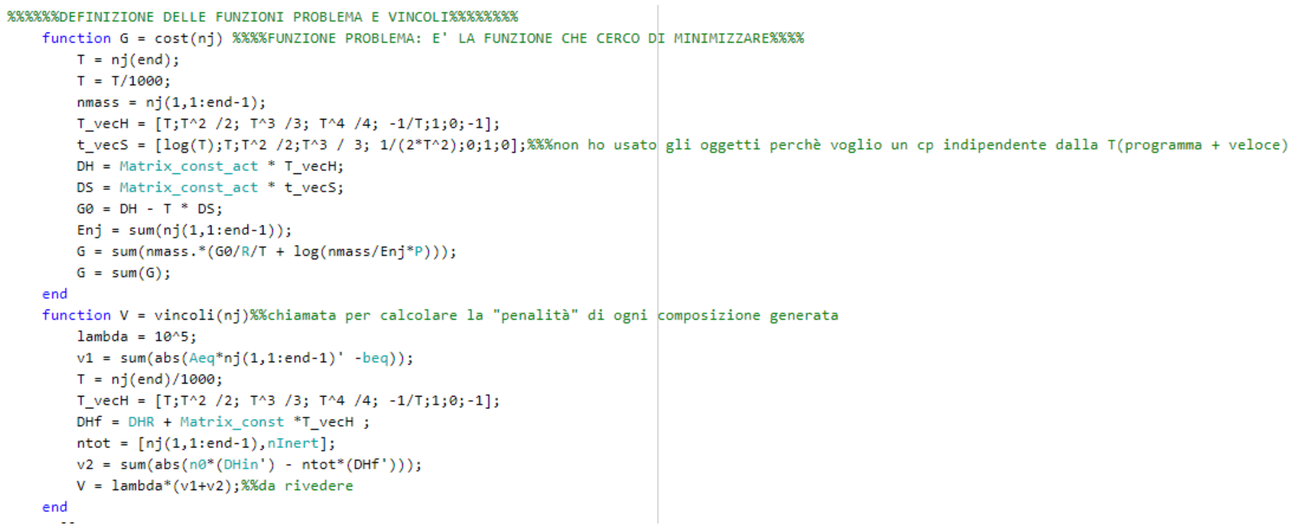


```
Matrix_const_act = zeros(numel(active_species),8);%%% solo specie attive
COMPeq = [active_species, inert_prod];%%array oggetti equilibrio
Teq = numeratore / denominatore;
DHin = [];
Aeq = zeros(7,numel(active_species));
beq = zeros(7,1);%vettore moli iniziali; la funzione minimizza con il vincolo Aeq*ns = beq.
for i = 1 : numel(active_species)
    Aeq(:,i) = active_species{i}.Comp;%matrice delle composizioni di ogni specie che compare nell'equilibrio(eccetto inerti)
    comp = varargin{i}.Comp;
    Matrix_const_act(i,:) = active_species{i}.getConst;
    beq(1) = beq(1) + (active_species{i}.MassFlux *comp(1));
    beq(2) = beq(2) + (active_species{i}.MassFlux *comp(2));
    beq(3) = beq(3) + (active_species{i}.MassFlux *comp(3));
    beq(4) = beq(4) + (active_species{i}.MassFlux *comp(4));
    beq(5) = beq(5) + (active_species{i}.MassFlux *comp(5));
    beq(6) = beq(6) + (active_species{i}.MassFlux *comp(6));
    beq(7) = beq(7) + (active_species{i}.MassFlux *comp(7));
end
for i = 1 : numel(Re)
    DHin(end+1) = Re{i}.intgH(database,Teq);%%%tiene conto anche degli inerti iniziali!!!!!
end
```

A1 Figure 12 FFA_minG data preprocessing

for the calculation of the integrals of enthalpy and entropy; the reason for the choice was for performance reasons.

The objective and constraint functions are used in the penalty minimization method, where the penalty parameter has been assigned an indicative value of 10^5.



```
%%%%%%DEFINIZIONE DELLE FUNZIONI PROBLEMA E VINCOLI%%%%%%%%
    function G = cost(nj) %%%%FUNZIONE PROBLEMA: E' LA FUNZIONE CHE CERCO DI MINIMIZZARE%%%%
        T = nj(end);
        T = T/1000;
        nmass = nj(1,1:end-1);
        T_vecH = [T;T^2 /2; T^3 /3; T^4 /4; -1/T;1;0;-1];
        t_vecS = [log(T);T;T^2 /2;T^3 / 3; 1/(2*T^2);0;1;0];%%%non ho usato gli oggetti perchè voglio un cp indipendente dalla T(programma + veloce)
        DH = Matrix_const_act * T_vecH;
        DS = Matrix_const_act * t_vecS;
        G0 = DH - T * DS;
        Enj = sum(nj(1,1:end-1));
        G = sum(nmass.*(G0/R/T + log(nmass/Enj*P)));
        G = sum(G);
    end
    function V = vincoli(nj)%%chiamata per calcolare la "penalità" di ogni composizione generata
        lambda = 10^5;
        v1 = sum(abs(Aeq*nj(1,1:end-1)' -beq));
        T = nj(end)/1000;
        T_vecH = [T;T^2 /2; T^3 /3; T^4 /4; -1/T;1;0;-1];
        DHf = DHR + Matrix_const *T_vecH ;
        ntot = [nj(1,1:end-1),nInert];
        v2 = sum(abs(n0*(DHin') - ntot*(DHf')));
        V = lambda*(v1+v2);%%da rivedere
    end
```

A1 Figure 13 "Cost" function and penalty functions

Finally, the upper and lower boundary limits for moles and temperature are defined, together with an initial solution on which to center the random generation of fireflies; this solution is obtained by calling FINDCOMP.

6.3 FFA

It is the function that adapts the firefly algorithm to the architecture of species-chemical objects. We will limit ourselves to indicating what are the various inputs to be varied, if desired, and refer to the discussion of the functioning of the algorithm itself in Chapter 4.

1) Alpha, beta, gamma; They regulate the movement of the firefly. Higher alpha values make movement more random. It is recommended that you leave the values given, or use commented expressions.
2) N = population size.

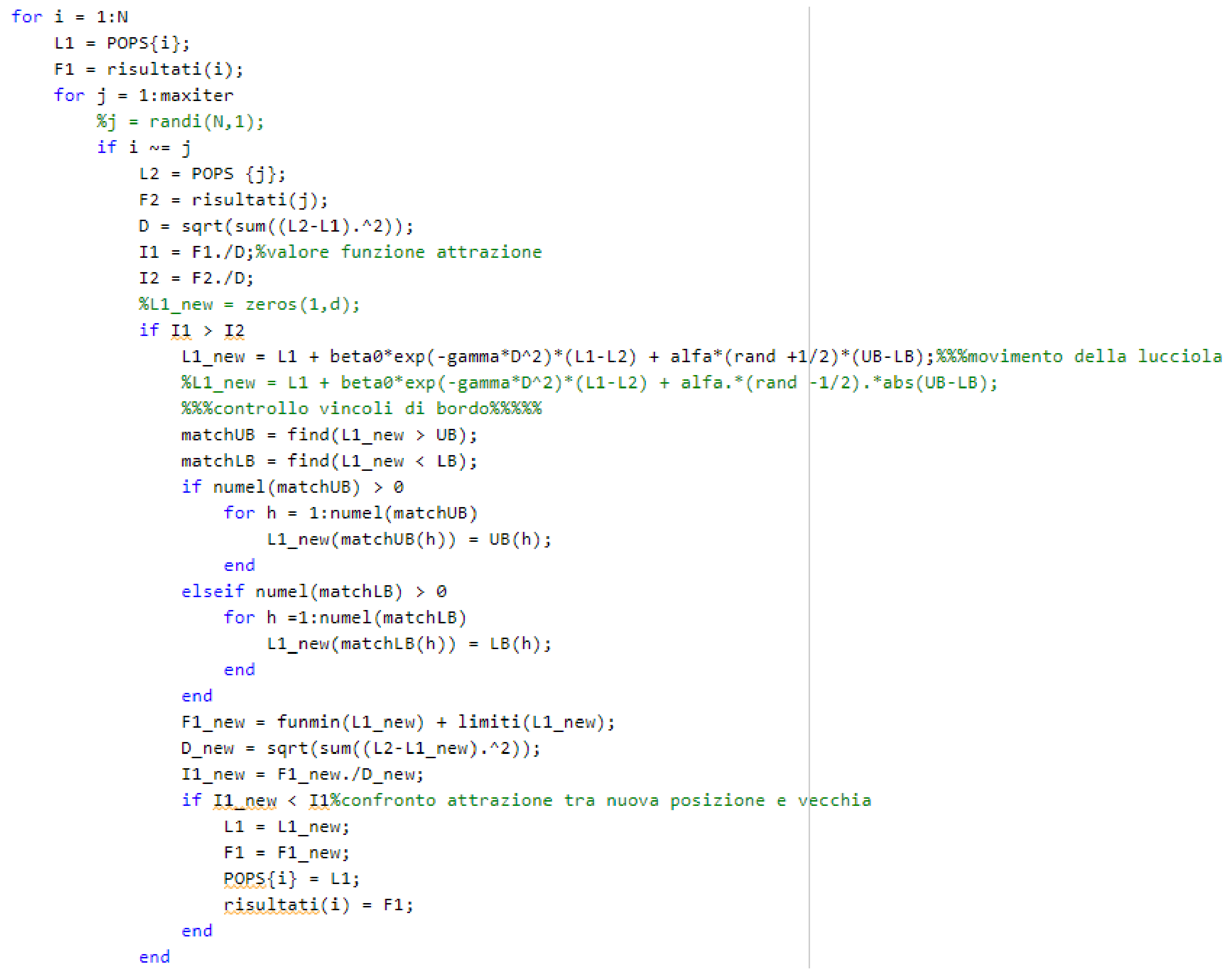
6.3.1 INPUT

FFA should never be called alone, it should be called ffa_minG. The input values of FFA are therefore:

1) Mini Upgrade Function.
2) Constraint function: If you want more than one constraint, enter the sum function of those constraints.
3) Upper limit for mass and temperature.
4) Lower limit for mass and temperature.
5) Size of the random vector to be generated (firefly).
6) Initial solution: acts as a central point for random generation.

6.3.2 GENERAL OPERATION OF THE PROGRAM

FFA generates N random vectors within the given limits, and initially associates to each of them the value of the objective function; with a for cycle, it compares the "Brightness" of a firefly with the rest of the population, deciding from time to time whether to move the firefly or not based on its brightness value:



```
for i = 1:N
    L1 = POPS{i};
    F1 = risultati(i);
    for j = 1:maxiter
        %j = randi(N,1);
        if i ~= j
            L2 = POPS {j};
            F2 = risultati(j);
            D = sqrt(sum((L2-L1).^2));
            I1 = F1./D;%valore funzione attrazione
            I2 = F2./D;
            %L1_new = zeros(1,d);
            if I1 > I2
                L1_new = L1 + beta0*exp(-gamma*D^2)*(L1-L2) + alfa*(rand +1/2)*(UB-LB);%%%movimento della lucciola
                %L1_new = L1 + beta0*exp(-gamma*D^2)*(L1-L2) + alfa.*(rand -1/2).*abs(UB-LB);
                %%%controllo vincoli di bordo%%%%%
                matchUB = find(L1_new > UB);
                matchLB = find(L1_new < LB);
                if numel(matchUB) > 0
                    for h = 1:numel(matchUB)
                        L1_new(matchUB(h)) = UB(h);
                    end
                elseif numel(matchLB) > 0
                    for h =1:numel(matchLB)
                        L1_new(matchLB(h)) = LB(h);
                    end
                end
                F1_new = funmin(L1_new) + limiti(L1_new);
                D_new = sqrt(sum((L2-L1_new).^2));
                I1_new = F1_new./D_new;
                if I1_new < I1%confronto attrazione tra nuova posizione e vecchia
                    L1 = L1_new;
                    F1 = F1_new;
                    POPS{i} = L1;
                    risultati(i) = F1;
                end
            end
```

A1 Figure 14 Firefly algorithm

Each update of the position is followed by a check of the upper and lower constraints (to ascertain that no solution generated is outside of them), and a check on the new brightness value; only if the latter is found to be lower (because it is minimizing) then the position of the firefly is actually updated.

At the end of each iteration for each firefly its best value is recorded, and compared with the overall best value, which is eventually updated.

6.4 OUTPUT

The combined program returns a row vector, consisting of the molar composition and the corresponding temperature; the output of FFA instead is given by the aforementioned vector and by the final value of the objective function obtained, necessary to monitor the trend of it based on the number of changes in the bestfit value.

A1 7. EX FUNCTION

7.1 REQUIREMENTS AND INPUTS

The EX function serves for heat the exergy of a given reaction; it requires as input:

1) Database.
2) Row cell vector containing all chemical objects that took part in the reaction.
3) Mass distribution/moles vector, obtained from other functions or constructed by the user.
4) Optional, enthalpy line vector, where any heat exchanges that the reactants undergo (any heating or cooling towards the reaction temperature) must be reported composed.

The function is designed to work with the results coming from the FINDCOMP and FFA_MING functions, but it is also possible to launch it with the results coming from minG2 after manual processing of the inputs: the mass distribution vector must contain the reaction temperature as the last value, so it will be necessary for the user to add this temperature to the output of minG2 (in the case of minG3 instead it will be necessary to replace the last value of the vector, the heat of reaction, with the reaction temperature). It is important that all components of the vectors involved follow the same order in which they were entered into the various previous functions:

ES.

test2 = minG3(A,1,C10H22,CO,H2,H2O)

.

.

zheat = [0,zheatCO,zheatH2,0]

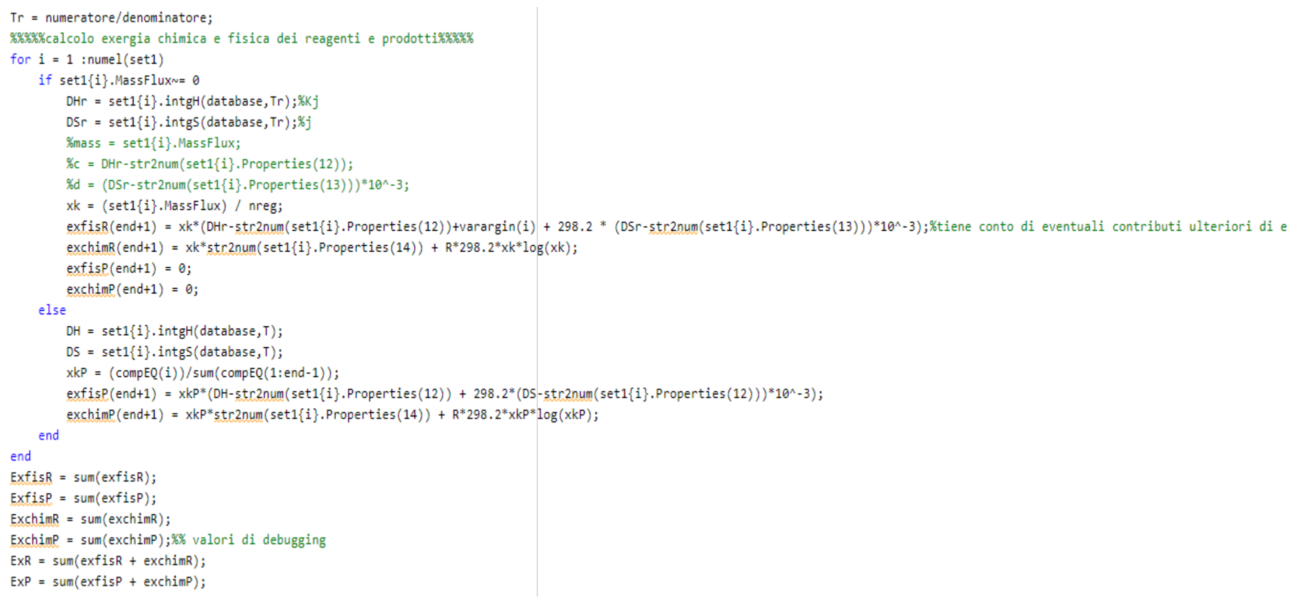
set = {C10H22,CO,H2,H2O}

ex =EX(A,set,k,zheat)

7.2 GENERAL OPERATION OF THE PROGRAM

The program is nothing more than a calculation of two exergy balances, one for the reactants and the other for the products; the exergy is divided into two components: chemistry and physics; therefore, in the program 4 values are assigned to be calculated.

It is expected that the various chemical species enter the reactor at different temperatures (as in the adiabatic case), managed by a short calculation of an equilibrium temperature.



```
Tr = numeratore/denominatore;
%%%%%calcolo exergia chimica e fisica dei reagenti e prodotti%%%%%
for i = 1 :numel(set1)
    if set1{i}.MassFlux~= 0
        DHr = set1{i}.intgH(database,Tr);%Kj
        DSr = set1{i}.intgS(database,Tr);%j
        %mass = set1{i}.MassFlux;
        %c = DHr-str2num(set1{i}.Properties(12));
        %d = (DSr-str2num(set1{i}.Properties(13)))*10^-3;
        xk = (set1{i}.MassFlux) / nreg;
        exfisR(end+1) = xk*(DHr-str2num(set1{i}.Properties(12))+varargin(i) + 298.2 * (DSr-str2num(set1{i}.Properties(13)))*10^-3);%tiene conto di eventuali contributi ulteriori di e
        exchimR(end+1) = xk*str2num(set1{i}.Properties(14)) + R*298.2*xk*log(xk);
        exfisP(end+1) = 0;
        exchimP(end+1) = 0;
    else
        DH = set1{i}.intgH(database,T);
        DS = set1{i}.intgS(database,T);
        xkP = (compEQ(i))/sum(compEQ(1:end-1));
        exfisP(end+1) = xkP*(DH-str2num(set1{i}.Properties(12)) + 298.2*(DS-str2num(set1{i}.Properties(12)))*10^-3);
        exchimP(end+1) = xkP*str2num(set1{i}.Properties(14)) + R*298.2*xkP*log(xkP);
    end
end
ExfisR = sum(exfisR);
ExfisP = sum(exfisP);
ExchimR = sum(exchimR);
ExchimP = sum(exchimP);%% valori di debugging
ExR = sum(exfisR + exchimR);
ExP = sum(exfisP + exchimP);
```

A1 Figure 15 Exergy calculation algorithm

The calculation of chemical exergy is identical for both reactants and products, while it differs for physics in that the possible contribution of the enthalpy vector to the reagent balance must be added.

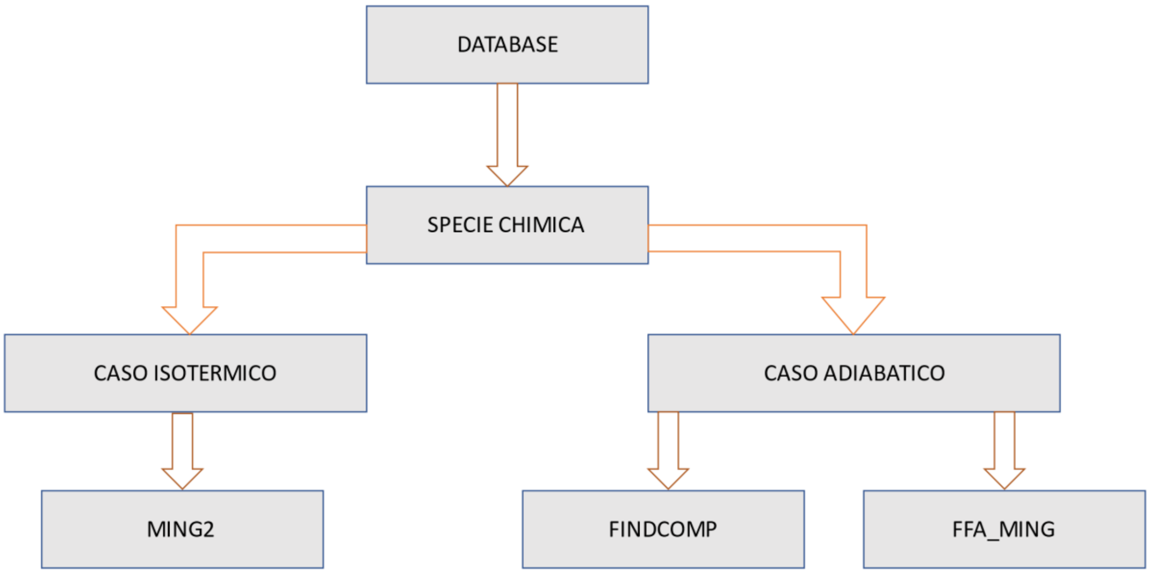
7.3 OUTPUT

The output returned will be an efficiency obtained from the ratio of total exergy (chemical + physical) of the products to that of the reactants[15].

A1 8. STANDARD PROCEDURE FOR THE MODEL

On the operational side, the operations to follow in order to use the program are therefore very simple; They are:

1) Initialize the database by creating the DATABASE object.
2) Initialize the various compounds present in the reaction to be studied, creating an object CHEMICAL SPECIES for each of them.
3) Use the appropriate function depending on the case study.



A1 Figure 16 Model Process Flow