Supplementary materials for Propane Pre-Reforming into Methane-Rich Gas over Ni Catalyst: Experiment and Kinetics Elucidation via Genetic Algorithm

Sergey I. Uskov [1,2], Dmitriy I. Potemkin [1,2,3,*], Leniza V. Enikeeva [2,4], Pavel V. Snytnikov [1,2], Irek M. Gubaydullin [5,6] and Vladimir A. Sobyanin [1]

[1] Boreskov Institute of Catalysis, Pr. Lavrentieva 5, Novosibirsk 630090, Russia; serg5810@gmail.com (S.I.U.); pvsnyt@catalysis.ru (P.V.S.); sobyanin@catalysis.ru (V.S.)
[2] Energy-Saving Catalytic Processes Laboratory, Novosibirsk State University, Pirogova St. 2, Novosibirsk 630090, Russia; leniza.enikeeva@yandex.ru
[3] Novosibirsk State Technical University, Department of Environmental Engineering, Karl Marx Pr. 20, Novosibirsk 630073, Russia
[4] Mathematical department, Ufa State Petroleum Technological University, Kosmonavtov St. 1, Ufa 450062, Russia
[5] Technological department, Ufa State Petroleum Technological University, Kosmonavtov St. 1, Ufa 450062, Russia; irekmars@mail.ru
[6] Institute of Petrochemistry and Catalysis of RAS, October Pr. 141, Ufa 450075, Russia
* Correspondence: potema@catalysis.ru; Tel.: +7-913-932-4620

Description of the proposed genetic algorithm

A genetic algorithm (GA) is a heuristic search algorithm that is used to solve optimization and simulation problems by randomly selecting, combining, and varying the desired parameters using mechanisms similar to natural selection in nature.

GA is an iterative process and iterations are called generations. The main process of GA is presented on Fig. 1. The process of genetic algorithm includes 5 stages:

- Initial population
- Selecting the most powerful solutions
- Exchange of characteristics between two individuals
- Mutation
- New iteration with creation of the initial population

The process begins with a set of individuals called a population. Each individual is a solution to the problem that has been posed. The idea of selection is to select the strongest individuals and pass their genes on to the next generation of individuals.

Crossover is the main part of the genetic algorithm. In this work, as a crossover function we used Scattered ('crossoverscattered'), the default crossover function for problems without linear constraints. Crossover creates a random binary vector and selects the genes where the vector is a 1 from the first parent, and the genes where the vector is a 0 from the second parent, and combines the genes to form the child. For example, if p1 and p2 are the parents

p1 = [a b c d e f g h]

p2 = [1 2 3 4 5 6 7 8]

and the binary vector is [1 1 0 0 1 0 0 0], the function returns the following child:

child1 = [a b 3 4 e 6 7 8]

To maintain the diversity of the population, a process of mutation of new individuals is carried out, where some part of its genotype is randomly changed.

Example of mutation of an individual:

Before mutation: [1 1 1 0 0 0]

After mutation: [1 1 **0 1 1** 0]

This is supposed to include some randomness into the process and to widen the search space. All new children represent the new generation and replace the old population. The whole process is then repeated until one of the stopping criteria is reached. When the algorithm is completed, the output is a set of optimal solutions to the given problem.

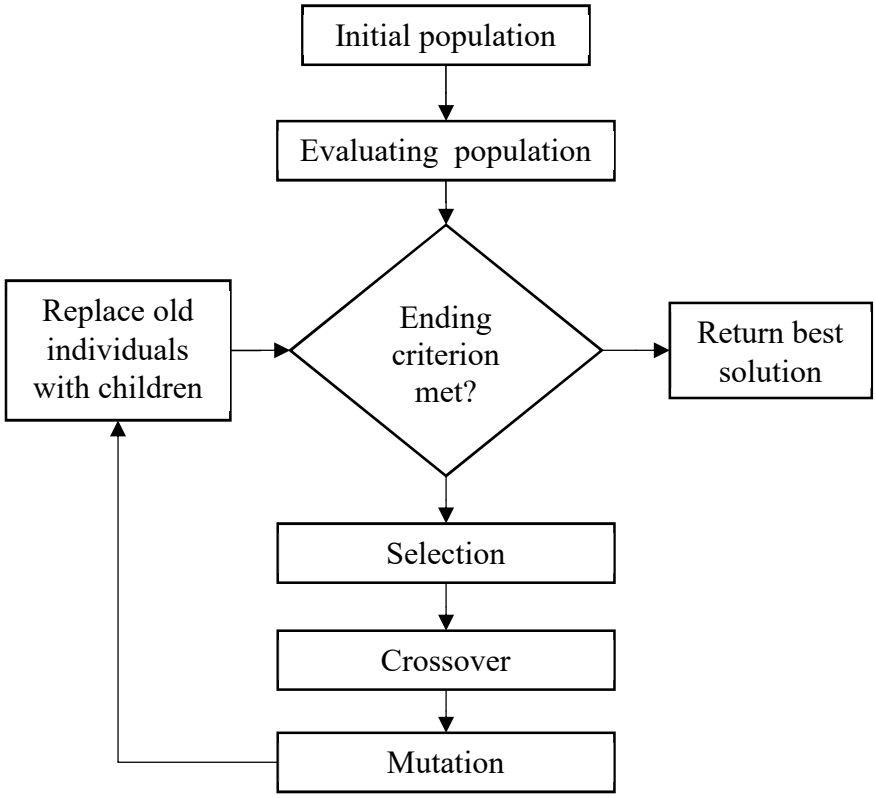

Fig. 1. Process of genetic algorithm

In this paper, population sizes between 50 and 250 were used. The adaptation of each parameter-set is measured with a fitness function, which is defined here as the error between the calculations and the experimental measurement – equation (4). The algorithm is stopped when no further improvement of the fitness occurs after several hundred generations.

The program code in the Matlab programming environment is given below [1]. The program is started by calling the main function **startGA_steam_ref**. In the 3rd line of the file **startGA_steam_ref.m**, the parameters of the genetic algorithm are set. The other parameters of the algorithm remain by default. 'PlotFcns' is a function that plots data computed by the algorithm: 'gaplotbestf' plots the best score value and mean score versus generation, 'gaplotbestindiv' plots the vector entries of the individual with the best fitness function value in each generation. ‘PopulationSize’ is a size of the population. Parameter ‘Generations’ is a number of iterations before the algorithm halts. ‘EliteCount’ is a positive integer specifying how many individuals in the current generation are guaranteed to survive to the next generation. In the fifth line of code, we

run a genetic algorithm with the parameters specified in the previous line to minimize the function @objective for the number of variables equal to 5, and the following restrictions on variables:

[90 10 30 4 -2] is the upper bound ,[150 7 60 7 2] is the lower bound, which is equivalent to the following inequalities:

$90 \le E_{ref}(kJ/mol) \le 150$; $10 \le \log_{10}(k_{ref}) \le 13$; $30 \le E_{met}(kJ/mol) \le 60$; $4 \le \log_{10}(k_{met}) \le 7$; $-2 \le n \le 2$,

where n is effective reaction order.

After completing the algorithm, we will see the solution **x** (a set of optimized parameters) and the value **fval** of the functional (1) of the deviation of the experimental data from the calculated data (lines 7 and 8).

startGA_steam_ref.m

```
1  clear all; close all;
2  format shortE
3  options = gaoptimset('PlotFcns',{@gaplotbestf,@gaplotbestindiv }, 'PopulationSize',100,'Generations',
4  1000, 'EliteCount',1);
5  [x fval, exitflag, output]=ga(@objective,5,[],[],[],[],[90 10 30 4 -2],[150 13 60 7 2],[],options);
6
7  x
8  fval
```

The file **arren.m** implements the Arrhenius equation.

arren.m

```
1  function y = arren(k0,Ea,T)
2     R = 8.314;
3     T = T + 273.15;
4     y = k0*exp(-Ea/(R*T));
```

The function **Ea_maker** converts the energy value from kJ/mol to J/mol.

Ea_maker.m

```
1  function y = Ea_maker(x)
2    y = x*1000;
```

The function **k_eq** calculates the equilibrium constant for a given temperature.

k_eq.m

```
1  function y = k_eq(T)
2     R = 8.314;
3     v = [-1 -4 1 2]; % CO2 H2 CH4 H2O
4
5     delH0F = [-393.51 0 -74.85 -241.81]*10^3;
6     S0_298 = [213.66 130.52 188.72 186.27];
7     a = [44.14 27.28 14.32 30.00];
8     b = [9.04 3.26 74.66 10.71]*(10^(-3));
9     c = [-8.54*10^5 0.5*10^5 -17.43/10^6 0.33*10^5];
10
11    delFH(1) = delH0F(1) + a(1)*(T-298)+b(1)*(T^2-298^2)/2-c(1)*(T^(-1)-298^(-1)); % CO2
12    delFH(2) = delH0F(2) + a(2)*(T-298)+b(2)*(T^2-298^2)/2-c(2)*(T^(-1)-298^(-1)); % H2
13    delFH(3) = delH0F(3) + a(3)*(T-298)+b(3)*(T^2-298^2)/2+c(3)*(T^3-298^3)/3;  % CH4
14    delFH(4) = delH0F(4) + a(4)*(T-298)+b(4)*(T^2-298^2)/2-c(4)*(T^(-1)-298^(-1)); % H2O
15
16    delRH = sum(v.*delFH);
17    delFS(1) = S0_298(1) + a(1)*(log(T)-log(298))+b(1)*(T-298)-(c(1)/2)*(T^(-2)-298^(-2)); % CO2
18    delFS(2) = S0_298(2) + a(2)*(log(T)-log(298))+b(2)*(T-298)-(c(2)/2)*(T^(-2)-298^(-2));      % H2
```

```
19    delFS(3) = S0_298(3) + a(3)*(log(T)-log(298))+b(3)*(T-298)+(c(3)/2)*(T^2-298^2);  % CH4
20    delFS(4) = S0_298(4) + a(4)*(log(T)-log(298))+b(4)*(T-298)-(c(4)/2)*(T^(-2)-298^(-2)); % H2O
21    delRS = sum(v.*delFS);
22    delRG = delRH - T*delRS;
23    y = exp(-delRG/(R*T));
```

The function **k0_maker** calculates the values of pre-exponential factors from the $\log_{10}(k)$.

k0_maker.m

```
1  function y = k0_maker(x)
2    y = 10^(x);
```

The file **objective.m** defines the target (minimized) function. The value of the target function is the sum of the values for each experiment (4 experiments are taken into account) (line 285). The experimental conditions are indicated in lines 9 - 18 for the first experiment, 80 - 94 for the second, 153 - 164 for the third, and 221 - 231 for the fourth experiment. The experimental conditions include a set of temperatures – an array of 'temper', concentrations of reaction components 'xexp', flow rate 'GHSV', height of the catalyst layer 'L', and pressure 'P0'. After the initial data is set, the concentrations of components along the length of the reactor are calculated for each temperature value (cycle 'for j = 1:length(temper')). Component concentrations are calculated by solving a system of ordinary differential equations:

[t,c]=ode23s('ode_steam_ref',tspan,yi0');

'ode23s' is a solver based on a modified Rosenbrock formula of order 2, 'ode_steam_ref' is a function that evaluates the right-hand side of the differential equations, 'tspan' is a vector specifying the interval of integration, 'yi0' is a vector of initial conditions.
After integration, the numerical value at the last integration point is compared with the experimental data, forming the value of the deviation functional:

```
for ii=1:(length(m)-1)
    F_temp = F_temp + abs(outlet(ii) - xexp(ii,j));
end
```

We get 4 values of the deviation functional (for each experiment) F1, F2, F3, F4, after which the arithmetic mean is calculated (line 282):

```
FF = (F1 + F2 + F3 + F4)/4;
```

objective.m

```
1   function y = objective(x)
2   global k;
3   global curretn_T; %[K]
4   global const_count;
5
6   int_step = 0.2 % step of integration
7
8   % exp1
9   temper = [223 233 245 253 260 267 277 290 293 306 319 334 353]; % C
10  xexp = [
11     86.425 80.285 72.215 64.544 59.181 52.588 41.744 30.487 28.250 20.715 11.281  5.098  1.101; %C3H8
12      3.298  4.569  6.082  7.289  8.418  9.450 11.100 11.863 13.210 14.586 15.867 17.147 17.740; %CO2
13      6.139  8.018 10.602  9.211 11.801 12.237 11.573  3.341 12.143 10.737 10.037 10.919 10.186; %H2
14      4.008  6.933 10.831 18.492 20.167 25.260 35.053 53.883 45.900 53.517 62.496 66.652 70.908 %CH4
15     ]./100;
16  GHSV = 4000;
17  L = 3.3; % height of the catalyst layer, cm
18  P0 = 1.05;
19
20  tspan = 0:int_step:L;
21  %  ----------------------------------------------
```

```
const_count = 2;

for j = 1:const_count
 Ea(j) = Ea_maker(x(2*j-1));
 k0(j) = k0_maker((x(2*j)));
end

% C3H8, CO2, H2, CH4, H2O
c0_m = [25 0 0 0 75];
m = [44 44 2 16 18]*10^(-3);

xi0 = c0_m./100;
yi0 = xi0.*m/sum(xi0.*m);

F = 0;
k(const_count+2) = GHSV;
k(const_count+3) = L;
k(const_count+4) = P0;
k(const_count+5) = degree;

for j = 1:length(temper)
   curretn_T = temper(j)+273.15;

   for i = 1:const_count
      k(i) = arren(k0(i),Ea(i),temper(j));
   end

   k(const_count+1) = k_eq(curretn_T);

   [t,c]=ode23s('ode_steam_ref',tspan,yi0');

   last = int64(L/int_step+1); % last point of integration
   temp_sum = 0;
   for i = 1:(length(m)-1)
      temp_sum = temp_sum + c(last,i)/m(i);
   end

   outlet = (c(last,:)./m)/temp_sum;

   F_temp = 0;

   for ii=1:(length(m)-1)
        F_temp = F_temp + abs(outlet(ii) - xexp(ii,j));
   end

   F_temp = F_temp/(length(m)-1);

   F = F + F_temp;

end

F1 = F/length(temper);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% exp2
temper = [235 249 264 274 283 290 298 306 313 317 331 338 347 360 374 384]; % C

xexp = [
   89.685 82.697 74.778 67.309 62.231 55.728 47.813 40.215 32.368 26.326 13.824 10.190 7.788 5.184
2.996 1.526; %C3H8
   2.923 4.285 5.800 7.200 8.330 9.293 10.560 11.709 13.031 13.528 15.973 16.648 17.168 17.961 18.051
18.389; %CO2
```

```
    5.363 7.690 10.426 12.262 14.010 13.859 13.859 12.910 13.120 12.047 12.847 13.504 14.403 15.520
16.802 16.783; %H2
    1.982 5.220 8.835 13.027 15.230 20.860 27.479 34.905 41.230 47.837 57.214 59.547 60.554 61.279
62.117 63.302 %CH4
    ]./100;
GHSV = 12000;
L = 3.3;
P0 = 1.05;

tspan = 0:int_step:L;
%  ------------------------------------------------

const_count = 2;

for j = 1:const_count
 Ea(j) = Ea_maker(x(2*j-1));
 k0(j) = k0_maker((x(2*j)));
end

c0_m = [25 0 0 0 75];
m = [44 44 2 16 18]*10^(-3);

xi0 = c0_m./100;
yi0 = xi0.*m/sum(xi0.*m);

F = 0;
k(const_count+2) = GHSV;
k(const_count+3) = L;
k(const_count+4) = P0;
k(const_count+5) = degree;

for j = 1:length(temper)
    curretn_T = temper(j)+273.15;

    for i = 1:const_count
        k(i) = arren(k0(i),Ea(i),temper(j));
    end

    k(const_count+1) = k_eq(curretn_T);

    [t,c]=ode23s('ode_steam_ref',tspan,yi0');

    last = int64(L/int_step+1);
    temp_sum = 0;
    for i = 1:(length(m)-1)
        temp_sum = temp_sum + c(last,i)/m(i);
    end

    outlet = (c(last,:)./m)/temp_sum;

    F_temp = 0;

    for ii=1:(length(m)-1)
           F_temp = F_temp + abs(outlet(ii) - xexp(ii,j));
    end

    F_temp = F_temp/(length(m)-1);

    F = F + F_temp;
end

F2 = F/length(temper);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
148
149  % exp3
150  temper = [291 308 325 341 356 372]; % C
151
152  % C3H8, CO2, H2, CH4, H2O
153  xexp = [
154     55.494 37.138 17.886 6.972 5.363 0.624; %C3H8
155     7.373 10.436 13.518 15.166 15.522 16.125; %CO2
156     5.246 6.372 6.560 6.217 6.318 6.263; %H2
157     31.887 46.054 62.036 71.644 72.796 76.988 %CH4
158     ]./100;
159  GHSV = 4000/5;
160  L = 2.3;
161  P0 = 5;
162
163  tspan = 0:int_step:L;
164  %  ------------------------------------------------
165
166  const_count = 2;
167
168  for j = 1:const_count
169   Ea(j) = Ea_maker(x(2*j-1));
170   k0(j) = k0_maker((x(2*j)));
171  end
172
173  % C2H6, C3H8, CO2, H2, CH4, H2O
174  c0_m = [25 0 0 0 75];
175  m = [44 44 2 16 18]*10^(-3);
176
177  xi0 = c0_m./100;
178  yi0 = xi0.*m/sum(xi0.*m);
179
180  F = 0;
181  k(const_count+2) = GHSV;
182  k(const_count+3) = L;
183  k(const_count+4) = P0;
184  k(const_count+5) = degree;
185
186  for j = 1:length(temper)
187     curretn_T = temper(j)+273.15;
188
189     for i = 1:const_count
190        k(i) = arren(k0(i),Ea(i),temper(j));
191     end
192
193     k(const_count+1) = k_eq(curretn_T);
194
195     [t,c]=ode23s('ode_steam_ref',tspan,yi0');
196
197     last = int64(L/int_step+1);
198     temp_sum = 0;
199     for i = 1:(length(m)-1)
200        temp_sum = temp_sum + c(last,i)/m(i);
201     end
202
203     outlet = (c(last,:)./m)/temp_sum;
204     F_temp = 0;
205
206     for ii=1:(length(m)-1)
207           F_temp = F_temp + abs(outlet(ii) - xexp(ii,j));
208     end
209
210     F_temp = F_temp/(length(m)-1);
```

```
    F = F + F_temp;
end

F3 = F/length(temper);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% exp4
temper = [312 328 343 375 384]; % C

xexp = [
    77.272 49.980 25.788 5.035 0.235; %C3H8
    4.652 8.442 12.386 15.821 16.251; %CO2
    10.396 8.982 7.126 9.807 7.647; %H2
    7.679 32.596 54.699 69.337 75.866 %CH4
    ]./100;
GHSV = 12000/5;
L = 2.3;
P0 = 5;

tspan = 0:int_step:L;
%  ------------------------------------------------

const_count = 2;

for j = 1:const_count
 Ea(j) = Ea_maker(x(2*j-1)); %1 3 5 7
 k0(j) = k0_maker((x(2*j))); % 2 4 6 8
end

% C2H6, C3H8, CO2, H2, CH4, H2O
c0_m = [25 0 0 0 75];
m = [44 44 2 16 18]*10^(-3);

xi0 = c0_m./100;
yi0 = xi0.*m/sum(xi0.*m);

F = 0;
k(const_count+2) = GHSV;
k(const_count+3) = L;
k(const_count+4) = P0;
k(const_count+5) = degree;

for j = 1:length(temper)
    curretn_T = temper(j)+273.15;

    for i = 1:const_count
        k(i) = arren(k0(i),Ea(i),temper(j));
    end

    k(const_count+1) = k_eq(curretn_T);

    [t,c]=ode23s('ode_steam_ref',tspan,yi0');

    last = int64(L/int_step+1);
    temp_sum = 0;
    for i = 1:(length(m)-1)
        temp_sum = temp_sum + c(last,i)/m(i);
    end

    outlet = (c(last,:)./m)/temp_sum;
    F_temp = 0;

    for ii=1:(length(m)-1)
```

```
274             F_temp = F_temp + abs(outlet(ii) - xexp(ii,j));
275         end
276
277         F_temp = F_temp/(length(m)-1);
278         F = F + F_temp;
279     end
280
281     F4 = F/length(temper);
282     FF = (F1 + F2 + F3 + F4)/4;
283     y = FF;
284     end
```

File **ode_steam_ref.m** defines function that evaluates the right-hand side of the differential equations (6). In lines 25 – 32, the mass flow of the gas mixture is calculated. 'v_ref1' and 'v_met' are stoichiometric coefficients of reactions (2) and (3). w_ref1 and w_met are the reaction rates (4) and (5).

ode_steam_ref.m

```
1   function [out1]=ode_steam_ref(t,x)
2   global k;
3   global curretn_T; %[K]
4   global const_count;
5
6   R = 8.314;
7   P0 = k(const_count+4);
8
9   P = P0*10^5; % pressure
10  m = [44 44 2 16 18]*10^(-3);
11  k_eq = k(const_count+1);
12
13  yi = (x'./m)*P0/sum(x'./m);
14  % C3H8, CO2, H2, CH4, H2O
15
16  Ci = yi*P/(R*curretn_T);
17
18  p_H2O = yi(length(m));
19  p_CO2 = yi(2);
20  p_H2 = yi(3);
21  p_CH4 = yi(4);
22
23  GHSV = k(const_count+2);
24  L = k(const_count+3);
25  Diam_reac = 0.008; % 8 mm
26  S_react = 3.14*Diam_reac^2/4;
27  GHSV = GHSV/3600; %s^(-1)
28  V_react = S_react*L;
29  u0 = V_react*GHSV;
30  F = u0*P/(R*298);
31  g1 = sum(yi.*m)*F;
32  G = g1/S_react;
33
34  v_ref1 = [-1 3 10 0 -6];
35  v_met  = [0 -1 -4 1 2];
36  % C3H8, CO2, H2, CH4, H2O
37
38  degree = k(const_count+5);
39  w_ref1 = k(1)*Ci(1)^degree;
40
41  C_H2 = Ci(3);
42  w_met = k(const_count)*C_H2*(1-(p_CH4*p_H2O^2)/(k_eq*p_CO2*p_H2^4));
43
```

```
xdot = (1/G)*((v_met*w_met+v_ref1*w_ref1).*m)';
out1=xdot;
```

The given program code solves the inverse problem with power law dependence. For the Langmuir-Hinshelwood model, the code is similar except for the expression of the reaction rate Wref and the number of optimized parameters. In this case, the optimization parameters are as follows: Eref, Emet, kref, kmet, m and B (see formula (7)).

[1] https://www.mathworks.com/