COBEM 2021 26th International Congress of Mechanical Engineering
ABCM Associação Brasileira de Engenharia e Ciências Mecânicas
November 22-26, 2021, Virtual Congress, Brazil

COB-2021-1238
Laine: Building an Open Source Web App for Equation Oriented Modeling

Rafael Nogueira Nakashima[1]
Silvio de Oliveira Junior[2]
Department of Mechanical Engineering, Polytechnic School, University of São Paulo
Av. Prof. Mello Moraes, 2231, Cidade Universitária, São Paulo, Brazil
rafaelnnakashima@usp.br[1]
soj@usp.br[2]

Abstract. *In general, problems in Mechanical Engineering are solved with the assistance of computer software or a programming language. This approach can provide accurate and fast results for problems with different levels of complexity, but often requires a deep knowledge of specific tools for mathematical modeling. An interesting alternative is to declare problems as mathematical equations and compute solutions by using a root-finding algorithm (e.g. Newton's method). Commercial software based on this method has already been employed in several Mechanical Engineering courses and research papers. However, free software options compatible with multiple platforms (e.g., web or mobile) are not available to this day, which significantly limits the use of equation oriented modeling on teaching and research. Thus, this study proposes a simple algorithm to reduce the complexity of large systems of equations and improve solution convergence without the need of user-specified guesses. Furthermore, an open-source web application, Laine (`https://laine.com.br`), was developed based on the solutions presented in this research. Performance parameters, such as computational speed and convergence, are compared with the established commercial software EES. The results indicate that the proposed solution is able to solve a wide variety of problems without specifying initial guesses, differently from other software. Thus, this novel approach offers a simpler and more accessible solution that can be used to introduce mathematical modeling on engineering courses and be a gateway to undergrad scientific research.*

Keywords: *equation oriented, root-finding algorithms, thermodynamics, open source, free software*

1. INTRODUCTION

Engineering problems can often be described as a set of equations and constraints to be solved or optimized in order to provide viable solutions. Since the procedure to state and solve problems may require several repetitive tasks and calculations, computer software has been developed to automate this process and reduce the workload of engineers. For instance, root-finding and optimization algorithms are available for most programming languages (e.g., MINPACK, IPOPT, GLPK, Scipy) (More *et al.*, 1980; Wächter and Biegler, 2006; Makhorin, 2010; SciPy 1.0 Contributors *et al.*, 2020), offering several options to model and extend problems, but require a deep knowledge of a specific programming language (e.g., Fortran, C, Python, Julia). Since modeling a problem in a programming language can be troublesome, several commercial software (e.g., EES, GAMS, AMPL) focus on providing a simplified solution to state and solve problems (Klein, 1993; Bussieck and Meeraus, 2004; Fourer *et al.*, 1990). The Engineering Equation Solver (EES), for example, allows the user to input a problem as a set of equations written in any order or format, which are simultaneously solved using a root-finding algorithm (Klein, 1993). This procedure significantly reduces the programming task of mathematical modeling and increases the flexibility of model analysis. In Thermodynamics, this approach is often called equation oriented modeling or simultaneous solution approach (Knopf, 2012), while in computer science it may be referred to as an algebraic modeling language (Fourer *et al.*, 1990).

Although these solutions are widely employed in academia and industry, there seems to be a lack of free software in this area which could be used cross-platform, for instance, on different desktop and mobile operating systems. This significantly limits the potential of using modern tools for teaching engineering courses focused on mathematical modeling, such as Thermodynamics, Fluid Mechanics and Dynamics. In addition, most root-finding algorithms require that the user provides a good initial guess for model conversion, a non-trivial task for most inexperienced users that can lead to convergence failure. Thus, this research aims to fulfill these gaps and difficulties by developing simple methods to solve equation oriented models with an automated procedure to reduce the problem complexity and provide good initial guesses. Furthermore, the proposed methodology is employed on an open-source web application, named Laine (`https://laine.com.br`), to demonstrate its capacities and provide an accessible framework of equation oriented modeling for engineering teaching and research.

2. METHODS

A simple framework for equation oriented modeling can be divided into two parts: a parser and a root-finding algorithm, as illustrated on Figure 1. The parser reads a user input and converts it into a set of instructions on the programming language for the root-finding algorithm; in other words, it "translates" the user language (equations) into programming code (e.g. abstract syntax tree). On the other hand, the root-finding algorithm is a procedure that aims to find a set of variables (roots) that nulls a given set of functions. They are usually based on the Newton's method with modifications to improve convergence (e.g., line-search) or reduce the computation time of the Jacobian matrix (e.g., Broyden's method) (Klein, 1993).

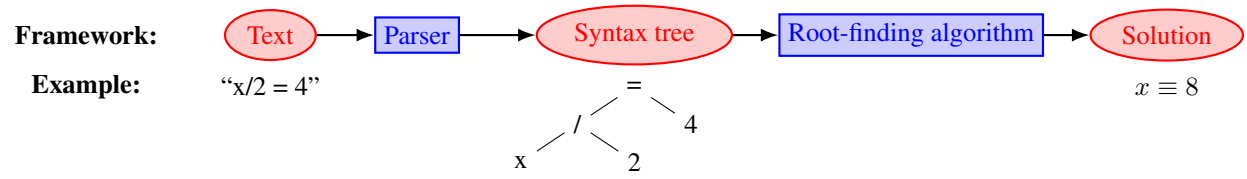


Figure 1: Simple framework for equation oriented modeling

2.1 Algebraic substitutions

A possible improvement on the simple framework presented in Figure 1 is to include algebraic substitutions. This process consists of reducing the number of unknown parameters of an equation by substituting the value of a variable by an expression given by another equation. For instance, Eq. (1) shows the reaction of carbon with oxygen and dissociation of carbon dioxide. In addition, Equations (2)-(5) present a chemical equilibrium problem of four variables and four equations, in which x_i are the molar fractions for component "i", z is the reaction extent and K is the equilibrium constant.

$$C + O_2 \rightleftharpoons (1 - 2z)CO_2 + (2z)CO + (1 + z)O_2 \tag{1}$$

$$x_{co_2} = \frac{1 - 2z}{2 + z} \tag{2}$$

$$x_{co} = \frac{2z}{2 + z} \tag{3}$$

$$x_{o_2} = \frac{1 + z}{2 + z} \tag{4}$$

$$K = \frac{x_{co}^2 x_{o_2}}{x_{co_2}^2} = 0.1089 \tag{5}$$

However, it is possible to substitute Eqs. (2)-(4) on Eq. (5) to create a subproblem of one variable and one problem, as shown in Eq. (6), which is simpler to solve. Thus, Eq. (6) can be solved first to determine the value of the reaction extent (z) and the molar fractions (x_{co_2}, x_{co}, x_{o_2}) can be solved sequentially with Eqs. (2)-(4). Besides reducing the number of function evaluations to calculate the Jacobian matrix, reducing the problem to a single variable also enables the use of different root-finding methods (e.g. Brent's method). Moreover, after solving the subproblem, some equations may not even require finding a root, but rather just evaluate the right-hand side expression of the equation (e.g., Eqs. 2 - 4 if z is known). Thus, the algebraic substitution can improve the convergence and computational speed of the root-finding algorithm.

$$\left(\frac{2z}{1 - 2z}\right)^2 \left(\frac{1 + z}{2 + z}\right) = 0.1089 \tag{6}$$

Listing 1 shows a pseudocode for a simple algebraic substitution algorithm. An algorithm for algebraic substitutions can be divided into three parts: (i) find the possible substitutions among all equations, (ii) apply substitutions between themselves and (iii) apply substitutions for the remaining equations. First, the algorithm identifies equations in which a variable is described as an expression of other variables. If the text has more than one expression for a single variable (e.g., $x = 2a$ and $x = a^2$), only the first equation is stored for algebraic substitutions. Next, the selected substitutions are applied between themselves to achieve the lowest number of dependent variables. Lastly, the substitutions are applied to the remaining equations in order to reduce the number of variables of these equations.

Listing 1: Pseudocode for an algebraic substitution algorithm

```
algorithm algebraic substitution is
  input: set of equations E
  output: set of equations E, S

  {Note: separate substitutions from original equations}
  S ← ∅
  for each e ∈ E do
    if(e.lhs.vars = variable v and                    ▷ if the left-hand side (LHS) of e is a variable v
       v ∉ e.rhs.vars and                             ▷ and v is not in the right-hand side (RHS) of e
       v ∉ s.lhs.vars ∀ s ∈ S) then                   ▷ and v is not in the LHS of all equations in set S then
         S ← S ∪ {e}                                  ▷ include e in S
         E ← E \ {e}                                  ▷ remove e from E

  {Note: apply substitutions between themselves}
  P ← ∅
  do
    flag ← false
    for each s_i ∈ S do
      for each s_j ∈ S do
        v_i ← s_i.lhs
        if(i != j and                                 ▷ if s_i and s_j are different
           (i,j) ∉ P and                              ▷ and the pair is not included in P
           v_i ∈ s_j.rhs.vars) then                   ▷ and the RHS of s_j has v_i
             s_j.rhs.vars.v_i ← s_i.rhs                ▷ substitute v_i in s_j for the RHS of s_i
             P ← P ∪ {(i,j),(j,i)}                     ▷ include the pair in the set P then
             flag ← true                               ▷ set the loop flag
  while (flag = true)                                 ▷ repeat while no further changes are possible

  {Note: apply substitutions for the remaining equations}
  for each e in E do
    for each s in S do
      v ← s.lhs
      if (v ∈ e.lhs.vars) then                        ▷ if the LHS of e has v then
        e.lhs.vars.v ← s.rhs                          ▷ substitute v in the LHS of e for the RHS of s_i
      if (v ∈ e.rhs.vars) then                        ▷ if the RHS of e has v then
        e.rhs.vars.v ← s.rhs                          ▷ substitute v in the RHS of e for the RHS of s_i

  return E,S
```

2.2 Random initial guesses at specific orders of magnitude

There seems to be no general strategy to find good initial guesses for root-finding algorithms when no information about the problem is available (e.g., bounds or guesses). However, just resourcing on "trial and error" with random guesses can be a very time consuming task. Thus, a software usually rely on default guesses or a user input to successfully solve a problem. However, for complex problems a default guess may not solve the problem and an inexperienced user could have difficulties to find a good initial guess (Klein, 1993).

The alternative proposed in this study is to assume that most solutions will probably have a similar order of magnitude. Then, it is possible to test and compare random guesses of different order of magnitude by evaluating the absolute difference between the sides of each equation. For instance, the problem stated in Eqs. (2)-(5) can be rewritten as Eqs. (9)-(10) to evaluate the absolute difference between the equation sides (Δ_i). At the solution, the absolute differences are null, thus if a particular set of guesses are closer to zero they may be good candidates for initial guesses. Table 1 shows the sum of absolute difference of Eqs. (9)-(10) for guesses of multiple orders of magnitude, as well as the number of iterations necessary for the Newton's method to converge.

$$\Delta_1 = \left| x_{co_2} - \frac{1-2z}{2+z} \right| \tag{7}$$

$$\Delta_2 = \left| x_{co} - \frac{2z}{2+z} \right| \tag{8}$$

$$\Delta_3 = \left| x_{o_2} - \frac{1+z}{2+z} \right| \tag{9}$$

$$\Delta_4 = \left| \frac{x_{co}^2 x_{o_2}}{x_{co_2}^2} - 0.1089 \right| \tag{10}$$

Table 1: Examples of different initial guesses and their impact on Newton's method conversion

Order of magnitude	x_{CO_2}	x_{CO}	x_{O_2}	z	$\sum_i \Delta_i$	Newton's method iterations
10^{-5}	0.00001	0.00002	0.00001	0.00002	1.1047	5
10^{-3}	0.00191	0.00110	0.00183	0.00129	1.1035	5
10^{-1}	0.16718	0.16601	0.16897	0.12550	0.6514	5
10^{0}	1.17681	1.44012	1.80026	1.92538	6.0043	7
10^{3}	1138.41	1909.55	1832.60	1787.90	10035	fails
Solution	0.31981	0.14415	0.53604	0.15535	0.0000	N/A

The example presented in Table 1 shows that, among the options evaluated, the best guesses are those with order of magnitude 10^{-1}. It is possible to observe a general correlation between the number of iterations taken by the Newton's method and the sum of absolute differences. However, it is important to highlight that a lower value of absolute differences does not guarantee the convergence of the Newton's method nor is it possible to prove that all solutions have similar orders of magnitude. The advantage of this method is that a wide spectrum of possibilities (10^{-5} to 10^{3} for this example) can be covered with a minimal number of evaluations (5 x 4 functions calls). Furthermore, these guesses can be compared and sorted based on the absolute difference between the equations sides, therefore they can be sequentially tested accordingly with this criteria. In practice, this procedure leads a higher success rate of the Newton's method compared with the traditional approach of relying on default guesses and minimizes the need for user-defined guesses.

A pseudocode of the initial guess search proposed in this study is described in Listing 2. In this example, the algorithm evaluates the sum of absolute differences for each order of magnitude and returns a sorted list of guess options. Then, the Newton's method can try the best guesses to find the real solutions. The algorithm described in Listing 2 may not provide good initial guesses if the variable have different orders of magnitude or if the equations have discontinuities and a limited domain. A simple workaround is to fix one or more variables and apply the algorithm for the remaining variables.

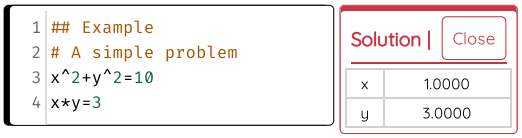
Listing 2: Pseudocode for an initial guess algorithm

```
algorithm initial guess is
  input: set of equations E, set of orders of magnitude M
  output: array of guesses G

  {Note: evaluates the residual for each order of magnitude}
  for each m ∈ M do
    for each variable v ∈ E do
      v ← m * (1 + random())                    ▷ assign a random value of order m for each variable
    m.error ← 0
    for each e ∈ E do
      m.error ← m.error + abs(e.lhs.value - e.rhs.value)      ▷ sum the absolute value of residuals

  G = M sorted by ascending m.error                            ▷ sort M by ascending error
  return G
```

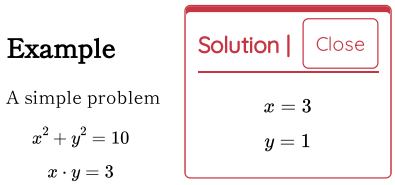
3. RESULTS

The solution strategies described in this study are implemented in an open-source web application, called Laine, available in the address '`https://laine.com.br`'. The website can be accessed by a modern browser and installed in desktops or smartphones to be used without internet connection. The software is developed in JavaScript and the user interface is designed with HTML and CSS. The source code can be downloaded via GitHub at the address '`https://github.com/srnogueira/laine`'. Besides solving nonlinear equations, Laine also includes two libraries of thermodynamic properties, CoolProp (Bell *et al.*, 2014) and NASA (McBride *et al.*, 2002), as well as the Lee-Kesler correlations for simple fluids (Lee and Kesler, 1975). These libraries enable Laine to solve Thermodynamic problems and can be further extended to include functions of other topics in Mechanical Engineering, such as Fluid Mechanics,

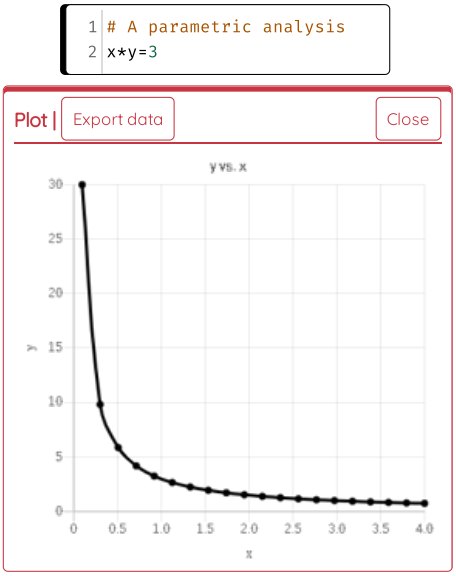
Heat Transfer, among others. Figure 2 shows the current user interface of the software and some of the functionalities described. A more detailed description of the Laine functionalities can be consulted in the project's GitHub page at 'https://github.com/srnogueira/laine'.



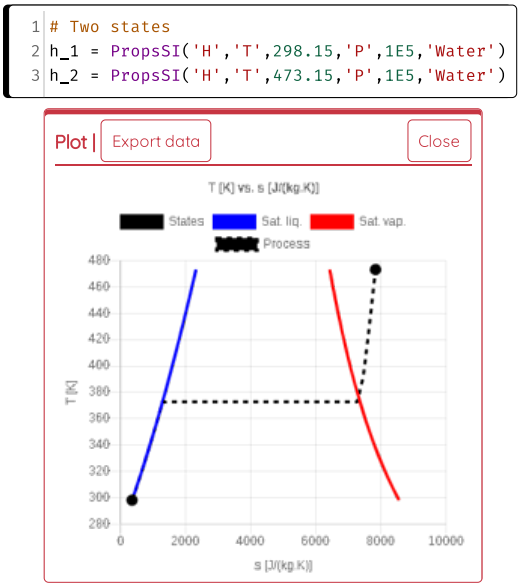
(a) Equations solver



(b) Markdown and LaTeX rendering



(c) Parametric analysis and plots



(d) Property plots

Figure 2: User interface and functionalities of Laine

3.1 Benchmark problems

In order to evaluate the convergence and computation speed of Laine, which implements the techniques proposed in this study, six problems were evaluated in the web application. These problems were also modeled in the commercial software EES and in the C++ programming language to provide a basis for comparison. The following list briefly describes each problem:

1. MINPACK nonlinear system: A problem of 9 equations and 9 variables proposed in the MINPACK documentation (More *et al.*, 1980).

2. Colebrook equation: Determines the friction factor and volumetric flow rate for a specific set of conditions. The problem was adapted from Example 6.9 of White (1999).

3. Chemical equilibrium of simultaneous reactions: Determines the yield of two simultaneous reactions based on their equilibrium constants. The problem was adapted from Example 14.9 of Moran *et al.* (2011).

4. Psychometric problem: A supermarket cooling system with bypass. This problem was proposed by Klein (1993) to showcase the use of EES software on Thermodynamic courses.

5. Vapor compression cycle problem: A heat pump cycle with heat exchange considerations. As Problem (4), this problem was also proposed by Klein (1993) to showcase the use of EES software on thermodynamic courses.

6. CGAM problem (optimal solution): A gas turbine cycle with steam cogeneration proposed by Valero *et al.* (1994) to discuss different methods for Thermoeconomic optimization.

The Appendix A provides the complete set of equations for each problem. The equations were adapted for each platform since the property functions are different for each software. Table 2 shows the average evaluation time for 50 evaluations of each problem and software analyzed in this study. Since EES was not able to solve some problems without an initial guess or variable limits, Table 2 also presents the average time when user-defined guesses are provided.

Table 2: Benchmark problems results [1]

Problem	Laine [2]	EES (with user-defined guesses)	C++ code [3]
(1)	25 ms	fails (34 ms)	2 ms
(2)	25 ms	fails (23 ms)	2 ms
(3)	19 ms	fails (23 ms)	1 ms
(4)	416 ms	82 ms	266 ms
(5)	375 ms	46 ms	151 ms
(6)	46 ms	33 ms	5 ms

As it can be observed in Table 2, Laine can achieve convergence for all problems without user-defined guesses in a reasonable evaluation time ($<$ 1 second). This can be partially attributed to the techniques proposed in this study, which expand the initial guess search and reduce the problem complexity. The results also indicate that the convergence speed can be significantly improved by using the same techniques in the C++ programming language. In practice, the ability to avoid user-defined guesses can simplify the user experience and avoid common frustrations with equation oriented modeling.

It is important to highlight that, although Laine is a web application written in JavaScript, the average evaluation time for the majority of benchmark problems is similar to those for EES. However, Laine may not be very efficient for problems with several functions calls for thermodynamic properties, as demonstrated by Problems (4) and (5). If necessary, the convergence speed can be improved by converting Laine to the C++ language or by using a different root-finding algorithm (e.g. Broyden–Fletcher–Goldfarb–Shanno algorithm). In general, the accessibility of the web platform does not compromise the software speed for the majority of problems, which is not what one may expect by only comparing the programming languages and environments.

4. CONCLUSIONS

In this research, two methods to improve solution convergence and reduce the complexity of nonlinear systems of equations are proposed and applied to develop an open-source software, called Laine (`https://laine.com.br`). The results indicate that these algorithms can improve the convergence of a root-finding algorithm (e.g., Newton's method) and reduce or eliminate the necessity of user-defined initial guesses. Moreover, the use of an open-source software available on the web platform can significantly increase the access of students and new researchers to equation oriented modeling, without drastically reducing the software performance. Thus, Laine can be used in teaching to introduce topics of mathematical modeling on several Mechanical Engineering courses and research topics (e.g., Thermodynamics, Fluid Mechanics, Dynamics, etc.). The source code is openly distributed (`https://github.com/srnogueira/laine`) and can be extended to include other problems (e.g., optimization, data visualization) and specific functions (e.g., transport properties, heat exchanger properties, etc.).

5. ACKNOWLEDGEMENTS

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. The first author acknowledges CAPES for his PhD grant and thanks Elaine dos Santos Bergamaschi for providing the artwork used in the Laine software. The second author acknowledges CNPq (Brazilian National Council for Scientific and Technological Development) for the grant 306484/2020-0

6. APPENDIX A - BENCHMARK PROBLEMS

The MINPACK nonlinear system (More *et al.*, 1980) was modeled in Laine and EES as follows:

```
(3-2*x_1)*x_1-2*x_2=-1
-x_1+(3-2*x_2)*x_2-2*x_3=-1
-x_2+(3-2*x_3)*x_3-2*x_4=-1
```

[3] Debian 10 Intel(R) Core(TM) i5-7200U CPU 2.50GHz
[3] Firefox 78.12.0 esr (64-bit)
[3] Available in `https://github.com/srnogueira/laine-solver`

```
-x_3+(3-2*x_4)*x_4-2*x_5=-1
-x_4+(3-2*x_5)*x_5-2*x_6=-1
-x_5+(3-2*x_6)*x_6-2*x_7=-1
-x_6+(3-2*x_7)*x_7-2*x_8=-1
-x_7+(3-2*x_8)*x_8-2*x_9=-1
-x_8+(3-2*x_9)*x_9=-1
```

The problem with the Colebrook equation White (1999) was modeled in Laine and EES as follows:

```
rho = 950
nu = 2E-5
d = 0.3
L = 100
epsod = 0.0002
hf = 8.0
g = 9.81
pi = 3.141593
Re = V*d/nu
Q=V*pi*d^2/4
f = (-2.0*log10(epsod/3.7+2.51/Re/f^0.5))^(-2)
hf = f*L/d*V^2/2/g
```

The problem about the chemical equilibrium of simultaneous reactions Moran *et al.* (2011) was modeled in Laine and EES as follows:

```
K_1=a/(1-a)*((1+a-b)/(4+a))^0.5
K_2=2*b/((1+a-b)*(1-b))^0.5
K_1=10^(-0.485)
K_2=10^(-0.913)
```

The EES code for the psychometric problem can be consulted in Klein (1993) and was adapted to Laine as follows:

```
Vol_5 = 4000*4.719474E-4 # m3/s
T_5=(62-32)*5/9+273.15 # K
rh_5 = 0.55 # [-]
P_atm = 1.01E5 # [Pa]
v_5=HAPropsSI('V','P',P_atm,'T',T_5,'R',rh_5) # m3/kg_dryAir
h_5=HAPropsSI('H','P',P_atm,'T',T_5,'R',rh_5) # J/kg
w_5=HAPropsSI('W','P',P_atm,'T',T_5,'R',rh_5) # kg_H2O/kg_dryAir
m_5 = Vol_5/v_5 # [kg]
T_6=(74-32)*5/9+273.15 # [K]
rh_6=0.54 # [-]
h_6=HAPropsSI('H','P',P_atm,'T',T_6,'R',rh_6) # J/kg
w_6=HAPropsSI('W','P',P_atm,'T',T_6,'R',rh_6) # kg_H2O/kg_dryAir
m_1=0.15*m_5 # [kg]
T_1=(82-32)*5/9+273.15 # [K]
rh_1=0.48 # [-]
h_1=HAPropsSI('H','P',P_atm,'T',T_1,'R',rh_1) # J/kg
w_1=HAPropsSI('W','P',P_atm,'T',T_1,'R',rh_1) # kg_H2O/kg_dryAir

# Bypass
byPass = 0 # [-]
# Mix return and outdoor
m_7 = 0.85*m_5*(1-byPass) # [kg]
m_2=m_1+m_7 # [kg]
w_2*m_2=m_1*w_1+m_7*w_6 # [kg]
h_2*m_2=m_1*h_1+m_7*h_6 # [kg]

# Cooling Coil
rh_3=1.0
m_3=m_2
Q_C=m_3*(h_3-h_2)
w_3=HAPropsSI('W','P',P_atm,'H',h_3,'R',rh_3) # J/kg
w_3=HAPropsSI('W','P',P_atm,'T',T_3,'R',rh_3) # J/kg
```

```
# Mix coil outlet with bypass
m_8=0.85*m_5*byPass
m_4=m_3+m_8
w_6*m_8+w_3*m_3=m_4*w_4
m_8*h_6+m_3*h_3=m_4*h_4

# Reheat coil
w_5=w_4
Q_H = (h_5-h_4)*m_5
```

The vapor compression cycle problem was adapted from the original EES code of Klein (1993) to Laine as follows:

```
T_amb = -15+273.15 # [K]
COP = Q_evap/W_c

# Evaporator
Alpha = 0.75E3
Q_evap = m*(h_1-h_4)
Q_evap = Alpha*(T_amb-T_1)

# Compressor
x_1=1
P_1=PropsSI("P","T",T_1,"Q",x_1,"R12")
h_1=PropsSI("H","T",T_1,"Q",x_1,"R12")
s_1=PropsSI("S","T",T_1,"Q",x_1,"R12")
s_2ID=s_1
P_2=PropsSI("P","T",T_3,"Q",1,"R12")
h_2ID=PropsSI("H","P",P_2,"S",s_2ID,"R12")
W_cID = -(h_2ID-h_1)*m
ComEff = 0.6
W_c=W_cID/ComEff
h_2=h_1-W_c/m
VolFlow = m/PropsSI("D","T",T_1,"Q",x_1,"R12")
VolFlow = 4.3E-3

# Condenser
T_H=20+273.15
Beta=1.75E3
Q_H=Beta*(T_3-T_H)
Q_H=(h_2-h_3)*m
h_3=PropsSI("H","T",T_3,"Q",0,"R12")
# Valve
h_4=h_3
P_4=P_1
x_4=PropsSI("Q","H",h_4,"P",P_4,"R12")
```

Lastly, the CGAM problem was modeled in Laine and EES (substituting “#” by “//”) as follows:

```
# Optimal conditions
P_ratio = 8.5234
P_ratio = P_2/P_1
eta_ac = 0.8468
T_3 = 914.28 # K
eta_gt = 0.8786
T_4 = 1492.63 # K

# Reference state
T_0 = 298.15 # K
P_0 = 1.013E5 # Pa

# Air
c_pa = 1.004 # kJ/(kgK)
gamma_a = 1.4
R_a = 0.287 # kJ/(kgK)
h_3 = c_pa*(T_3-T_0)
```

```
P_7 = P_0

# Combustion gas
c_pg=1.17 # kJ/(kgK)
gamma_g = 1.33
R_g = 0.290 # kJ/(kgK)
h_4 = c_pg*(T_4-T_0)

# Water
T_8 = 298.15 # K
P_8 = 20E5 # Pa
h_8 = 106.68 # kJ/kg
T_9 = 485.53 # K
P_9 = 20E5 # Pa

# Air compressor
T_2 = T_1*(1+1/eta_ac*((P_2/P_1)^((gamma_a-1)/gamma_a)-1))
T_1 = T_0 # K
P_1 = P_0 # Pa
W_ac = m_a*c_pa*(T_2-T_1)

# Combustion chamber
m_g = m_a + m_f
m_a*h_3 + m_f*LHV = m_g*h_4 + Q_lcc
LHV = 50000 # kJ/kg
Q_lcc = m_f*LHV*(1-eta_cc)
eta_cc = 0.98
P_4 = P_3*(1-$DeltaP_cc)
$DeltaP_cc = 0.05

# Preheater
m_a*c_pa*(T_3-T_2)=m_g*c_pg*(T_5-T_6)
P_3 = P_2*(1-$DeltaP_a_aph)
$DeltaP_a_aph = 0.05
P_6 = P_5*(1-$DeltaP_g_aph)
$DeltaP_g_aph = 0.03

# Gas turbine
T_5 = T_4*(1-eta_gt*(1-(P_4/P_5)^((1-gamma_g)/gamma_g)))
W_gt = m_g*c_pg*(T_4-T_5)
W_net = W_gt-W_ac
W_net = 30E3 # W

# HRSG
T_8p = T_9 - $DeltaT_a
$DeltaT_a = 15
m_g*c_pg*(T_6-T_7p) = m_s*(h_9-h_8p)
m_s = 14 # kg/s
h_9-h_8p = 1956 #kJ/kg
$DeltaT_p = T_7p-T_9
T_7=T_6-m_s*(h_9-h_8)/(m_g*c_pg)
h_9-h_8=2690 #kJ/kg
P_0=P_6*(1-$DeltaP_hrsg)
$DeltaP_hrsg = 0.05
```

7. REFERENCES

Bell, I.H., Wronski, J., Quoilin, S. and Lemort, V., 2014. "Pure and Pseudo-pure Fluid Thermophysical Property Evaluation and the Open-Source Thermophysical Property Library CoolProp". *Industrial & Engineering Chemistry Research*, Vol. 53, No. 6, pp. 2498–2508. ISSN 0888-5885, 1520-5045. doi:10.1021/ie4033999. URL https://pubs.acs.org/doi/10.1021/ie4033999.

Bussieck, M.R. and Meeraus, A., 2004. "General Algebraic Modeling System (GAMS)". In P.M. Pardalos, D.W. Hearn

and J. Kallrath, eds., *Modeling Languages in Mathematical Optimization*, Springer US, Boston, MA, Vol. 88, pp. 137–157. ISBN 978-1-4613-7945-4 978-1-4613-0215-5. doi:10.1007/978-1-4613-0215-5_8. URL http://link.springer.com/10.1007/978-1-4613-0215-5_8. Series Title: Applied Optimization.

Fourer, R., Gay, D.M. and Kernighan, B.W., 1990. "A Modeling Language for Mathematical Programming". *Management Science*, Vol. 36, No. 5, pp. 519–554. ISSN 0025-1909, 1526-5501. doi:10.1287/mnsc.36.5.519. URL http://pubsonline.informs.org/doi/abs/10.1287/mnsc.36.5.519.

Klein, S.A., 1993. "Development and integration of an equation-solving program for engineering thermodynamics courses: EES for engineering thermodynamics". *Computer Applications in Engineering Education*, Vol. 1, No. 3, pp. 265–275. ISSN 10613773. doi:10.1002/cae.6180010310. URL http://doi.wiley.com/10.1002/cae.6180010310.

Knopf, F.C., 2012. *Modeling, analysis, and optimization of process and energy systems*. Wiley, Hoboken, N.J. ISBN 978-0-470-62421-0.

Lee, B.I. and Kesler, M.G., 1975. "A generalized thermodynamic correlation based on three-parameter corresponding states". *AIChE Journal*, Vol. 21, No. 3, pp. 510–527. ISSN 0001-1541, 1547-5905. doi:10.1002/aic.690210313. URL http://doi.wiley.com/10.1002/aic.690210313.

Makhorin, A., 2010. "Modeling Language GNU MathProg - Language Reference". Technical report, Department for Applied Informatics, Moscow Aviation Institute, Moscow, Russia. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.709.9342&rep=rep1&type=pdf.

McBride, B.J., Zehe, M.J. and Gordon, S., 2002. "NASA Glenn Coefficients for Calculating Thermodynamic Properties of Individual Species". Technical Report NASA/TP-2002-211556, NASA Glenn Research Center, Cleveland, OH United States.

Moran, M.J., Shapiro, H.N., Boettner, D.D. and Bailey, M.B., 2011. *Fundamentals of engineering thermodynamics*. Wiley, Hoboken, N.J.?, 7th edition. ISBN 978-0-470-91768-8 978-0-470-49590-2.

More, J., Garbow, B. and Hillstrom, K., 1980. "User guide for MINPACK-1. [In FORTRAN]". Technical Report ANL–80-74, 6997568, Argonne National Laboratory. doi:10.2172/6997568. URL http://www.osti.gov/servlets/purl/6997568/.

SciPy 1.0 Contributors, Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S.J., Brett, M., Wilson, J., Millman, K.J., Mayorov, N., Nelson, A.R.J., Jones, E., Kern, R., Larson, E., Carey, C.J., Polat, İ., Feng, Y., Moore, E.W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E.A., Harris, C.R., Archibald, A.M., Ribeiro, A.H., Pedregosa, F. and van Mulbregt, P., 2020. "SciPy 1.0: fundamental algorithms for scientific computing in Python". *Nature Methods*, Vol. 17, No. 3, pp. 261–272. ISSN 1548-7091, 1548-7105. doi:10.1038/s41592-019-0686-2. URL http://www.nature.com/articles/s41592-019-0686-2.

Valero, A., Lozano, M.A., Serra, L., Tsatsaronis, G., Pisa, J., Frangopoulos, C. and von Spakovsky, M.R., 1994. "CGAM problem: Definition and conventional solution". *Energy*, Vol. 19, No. 3, pp. 279–286. ISSN 03605442. doi:10.1016/0360-5442(94)90112-0. URL https://linkinghub.elsevier.com/retrieve/pii/0360544294901120.

Wächter, A. and Biegler, L.T., 2006. "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming". *Mathematical Programming*, Vol. 106, No. 1, pp. 25–57. ISSN 0025-5610, 1436-4646. doi:10.1007/s10107-004-0559-y. URL http://link.springer.com/10.1007/s10107-004-0559-y.

White, F.M., 1999. *Fluid mechanics*. McGraw-Hill, Boston; Toronto, 4th edition. ISBN 978-0-07-228192-7 978-0-07-847965-6. OCLC: 299846668.

8. RESPONSIBILITY NOTICE

The authors are solely responsible for the printed material included in this paper.