CSCE 155N: Matlab

Final Project - Allen

Wednesday, April 25th, 2018

"*Balancing Equations Using Matlab*"

1. General idea of the program

Similar to the mathematical equations, chemical equations must be balanced. The chemical equation must be balanced with respect to the number of atoms of each chemical element involved in the reaction, and the electrical charge on both sides of the equation. Our program only works with non-ionic equation. To put it another way, the electrical charge balancing is neglected. As mentioned in our proposal, the determination of the stoichiometric coefficients in a chemical equation is mathematically equivalent to solving a system of linear algebraic equations. Using matrix is a non-brainer choice to solve this problem.

Here is an example:

x1CH4 + x2O2 → x3CO2 + x4H2O

To determine the unknown x1, x2, x3, x4, we need to first write a balance equation for each of these elements:

Carbon (C): 1⋅x1 + 0⋅x2 = 1⋅x3 + 0⋅x4

Hydrogen (H): 4⋅x1 + 0⋅x2 = 0⋅x3 + 2⋅x4

Oxygen (O): 0⋅x1 + 2⋅x2 = 2⋅x3 + 1⋅x4

We write these as homogeneous equations, each having zero on its right-hand side:

x1 – x3 = 0

4x1 – 2x4 = 0

2x2 – 2x3 – x4 = 0

We can then choose an auxiliary equation by arbitrarily choosing a value for one of the coefficients. Here, pick x4 = 1

The complete system of equations can be written in matrix form as Ax = b, where

A = [1 0 –1   0

       4 0 0 –2

       0 2 –2 –1

       0 0 0   1]

x = [x1

      x2

      x3

      x4]

b = [0

        0

        0

        1]

Solving for the x matrix will give us the coefficients we need.

The calculation above is called the linear system Ax = b calculation. MATLAB has provided us with the *rref* function that would perfectly carry out the calculation, so our task was to figure out the matrix from the input chemical equation

1. Process of implementation (functions, control statement)
   1. OVERALL BRAINSTORM AND IMPLEMENTATION
      1. The input expected is a cell array. Take the equation {‘CH4’, ‘O2’, ‘CO2’, ‘H2O’} for example
      2. For starter, the idea is to split each element and their atom counts accordingly
         1. For example, if the program is given CH4, it would first split the string into 'C' 'H' '4', then from there, putting them into a structure
         2. This is achieved using the function *structConvert*
      3. *structConvert* only computes the struct for ONE species of the given array. Now another function is needed to compute the overall struct. *counAtom* was implemented to achieve this goal.
         1. *countAtom* calls the ‘function’ function *structConvert* to compute the overall struct
      4. *myMat* computes the matrix based on the struct obtained
      5. *molecularWeight* computes the molecular weights of each species
      6. *chemBalance* performs the linear system calculation, returns the coefficients, and displays the graphs and molecular weight values
   2. WHAT FUNCTIONS AND CONTROL STATEMENTS WERE USED?
      1. *regexp:* One function used that was not discussed in class was regular ‘regexp’. *Regexp* was used to match any chemical element symbol with the input string. (<https://www.johndcook.com/blog/2016/02/04/regular-expression-to-match-a-chemical-element/)>
      2. *cellfun:* Applies function to each cell in the cell array. *Cellfun* was used in *structConvert* to get the atomic symbol and the atomic counts to each atom in the input string
      3. *str2double:* In *structConvert,* because everything was in terms of string, it was necessary to convert string to a number is order to assign the number of atoms
      4. *strcmp:* Again in *structConvert,* used *strcmp* to compare the input atom and the element expression. If they matched, make the struct
      5. *fields* and *isfield*
      6. Recursive was also used. This is because elements are not always on their own, they can be grouped together in parenthesis. Therefore we need to do a recursive to assign atomic numbers correctly
         1. For example: Ca(OH)2 has 1Ca, 2O, and 2H
      7. *union*: Because we found the struct for each species, there should be repeated fields. Use *union* to set union of 2 arrays with no repetition
      8. *ismember*: This has functionality like *strcmp*, but used *ismember* because I was working with cell array, not string
      9. *reshape:* Reshape the structure array so that it has the same size as the cell array input. In *countAtom,* if the input cell array has size 1x3, the output struct should also has size 1x3
      10. *xlsread, xlswrite:* These functions were used because we chose to use excel file as our input and output files
      11. *bar, title, legend, set:* Used to make the graphs
      12. *rref:* MATLAB’s handy built in function that allows us to perform the balancing calculation.
      13. *uicontrol:* Used to make the GUI
      14. A lot of *if* and *for* statements were used.
          1. *for* was mostly used to fill in values of the desired struct, matrix, etc.
2. Errors that a user can expect to perform and how those are handled
   1. Elements, by convention, consist of one or two character abbreviations. The character is capitalized. If present, the second character is a lowercase letter. Therefore, error messages will be generated for invalid formulas input by the user.
      1. For example, if the user input ‘LA’ instead of ‘La’, the program will throw an error message.
   2. Error message will be generated if the user input an element name that does exist.
      1. For example, ‘An’ is NOT an element. If the user input ‘An’, error message will also be displayed
3. Original reviewed and commented proposal

