Visual basic for design assignment 2 – chemical equilibrium – NEWTON’S METHOD WITH BOUNDARIES

Bethany Mulliner

|  |  |
| --- | --- |
| **Topic** | **Page Number** |
| **Title Page** | 1 |
| **Contents Page** | 2 |
| **Introduction** | 3 |
| **Boundaries & Limitations** | 5 |
| **Determining the Sign of n0\*** | 7 |
| **Iteration Safety Feature** | 7 |
| **Calculating ñ\*** | 8 |
| **Appendix I** | 13 |
| **References** | 14 |

**Contents**

**Introduction**

For the generic chemical equilibrium reaction:

Where:

νi – Stoichiometric coefficient of the corresponding component

Ci – Component

The gas phase reaction equilibrium is given by the reaction quotient:

Where:

Xi – Mole fractions, given by:

Where:

ni – Number of moles of component i

To determine the composition of the equilibrium system, we solve:

To do this, we must take the reaction progress and stoichiometry into account via:

Where:

Δni – Change in moles of component i due to reaction progress

n\* - Extent of reaction

Hence:

And thus, the composition of the equilibrium system can be determined. However, this can also be done a different way:

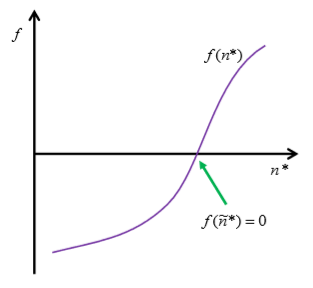
1. Guess a value for n\* and call it n0\*.
2. Calculate ni(n0\*).
3. Calculate Xi(n0\*).
4. Test to see if n0\* = n\*. If it is, then the following must be true:

For the second method, it is likely that the initial guess of n0\* is incorrect and the previous equation will not be true. In order to find a better value for a guess for n\*, say n1\*, we can use Newton’s method to find roots. To do this, we view the RHS of the following equation as a function of n\*:

Hence:

And hence, the value ñ\* for which f(ñ\*)=0 will solve this equation.

This can be graphically represented as:



The overall task of this assignment is to use Visual Basic to design and implement a function that calculates ñ\* from a generic chemical equation with six components. In order to do this, we must first identify the boundaries and limitations of the problem. Next, we must determine the sign of n0\* in order to come to a suitable starting value. Finally, an iterative coded solution can be created with several features to determine when a suitable value of ñ\* has been calculated.

(Bock, 2018)

N.B: Line numbers for code are obtained using Notepad++. (Ho, 2016)

**Boundaries & Limitations**

The generic chemical equation provided has six components. Which ones are reactants and which ones are products is not specified, therefore, for the sake of ease of explanation, it has been assumed for this section that components A, B, C and D are reactants, and components E, and F are products. However, the explanation should prove sufficient enough that it can be expanded to any number of components with any number of reactants and products.

As such, our generic equation is therefore:

Firstly, it is useful to break down the reaction in order to identify anything we should be aware of when deciding the boundaries of the problem:

|  |  |  |  |
| --- | --- | --- | --- |
| **Stoichiometric Coefficients, ν** | **Initial Number of Moles, n0** | **Change in the Number of Moles, Δn, with respect to the Extent of Reaction, n\*** | **Final Number of Moles, ni** |
| a | na | -an\* | na – an\* |
| b | nb | -bn\* | nb – bn\* |
| c | nc | -cn\* | nc – cn\* |
| d | nd | -dn\* | nd – dn\* |
| e | 0 | en\* | en\* |
| f | 0 | fn\* | fn\* |

Hence:

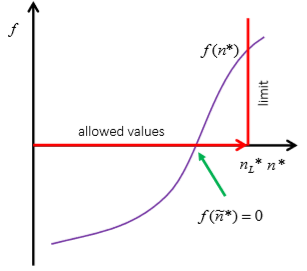
And:

From this, in order for the reaction to take place and to stay firmly within the realms of reality, we must assume that na, nb, nc, nd, ne and nf are all equal to or greater than zero. Similarly, since n\* represents the extent of the reaction, it also must be non-negative. All instances of the final number of moles must also be non-negative for the same reason.

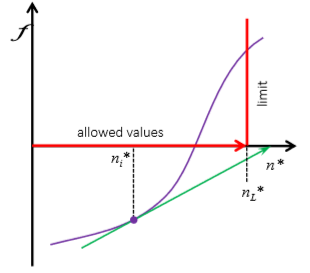
For this to be true, na must be equal to or greater than an\* and that na/n must be equal to or greater than n. Following this through for the other reactants, we come to the same conclusions.

In order to constrain the reaction, we require a limiting reagent, say B, hence setting the boundaries as 0 and nb/b, the latter of which we call nL\*. (Falenta, 2018)

Hence, we determine that the limits on n\* must be graphically represented as follows:



An issue that arises with the use of iterative methods is that values may be suggested that lie outside of the specified boundaries:



If such a case arises, an alternative value should be chosen according to:

(Bock, 2018)

**Determining the Sign of n0\***

Once again considering our generic equation:

To determine the sign of n0\*, we must make several assumptions.

Firstly, for the reactants, we can assume that at equilibrium, the stoichiometric coefficient, νi, is negative. This therefore implies that Δni is negative as there is a decrease in the number of moles of reactants due to the reaction occurring. We previously stated in the **introduction** that:

Therefore, by inspection we can ascertain that n\* must be positive in order for the equation to be true.

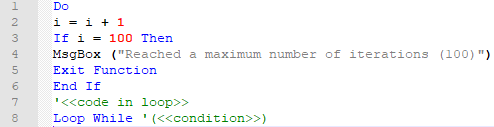
Secondly, for the products, we can assume that at equilibrium, the stoichiometric coefficient, νi, is positive. This therefore implies that Δni is positive as there is an increase in the number of moles of products due to the reaction occurring.

Once again considering the previously stated equation, we can again ascertain by inspection that n\* must be positive in order for the equation to be true.

(Falenta, 2018)

**Iteration Safety Feature**

After 100 iterations, it is likely that a satisfactory result will have been reached, therefore some code was required to stop the code at this point and send a message to inform the user that this had taken place. Therefore, the following code was devised:



(Falenta, 2018)

**Line-by-line Explanation**

|  |  |
| --- | --- |
| **Line Number** | **Explanation** |
| 1 | Opens the loop. |
| 2 | Adds 1 to the variable i and stores the result to the same variable. |
| 3-6 | Once the variable i is equal to 100, (i.e., the code has been run 100 times), send a message to the user informing them that 100 iterations have occurred. The function then exits as it is completed. |
| 7 | The code to be run in the loop goes here. This will be discussed later. |
| 8 | Causes the code to loop while a certain condition is true. Again, this will be specified later. |

(Microsoft, 2018)

**Calculating ñ\***

A solution can now be coded in order to calculate ñ\*. The Newton-Raphson method is used for the iteration and is given by:

(Falenta, 2018)

Since this is an iterative method, the exact answer for ñ\* will never be reached as k→∞. Therefore, a suitable condition at which to stop the iteration must be chosen. A good condition is given by:

(Bock, 2018)

Now, whilst keeping these in mind, as well as the iteration safety feature, the sign of no\* and the boundaries and limitations, a solution was written.

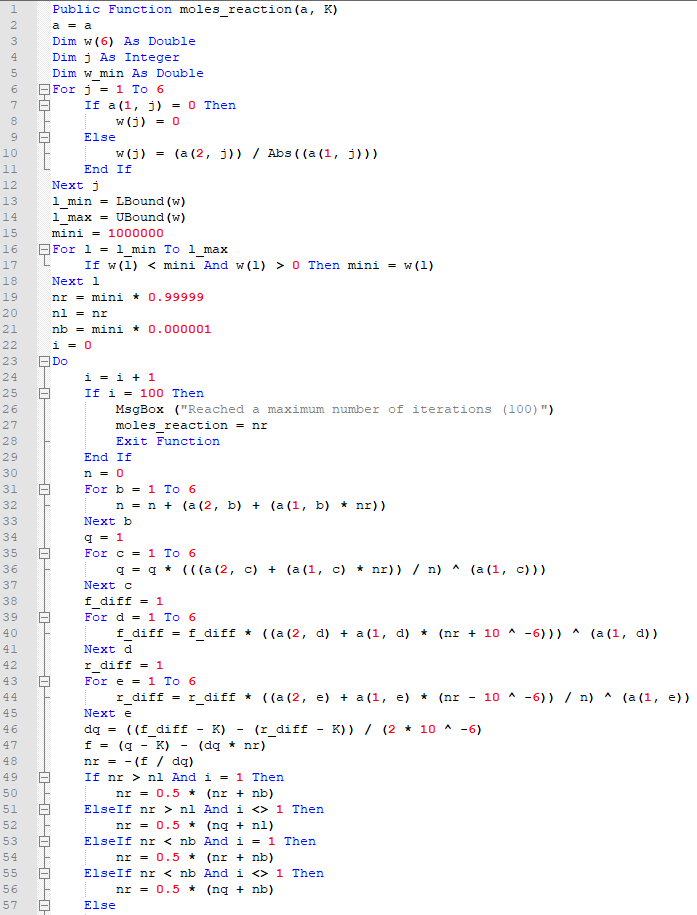
**General Settings**

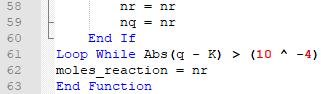
The top of the code contains the following statement:



This setting changes the base that Visual Basic will start any operations involving counts (such as the filling of arrays) from 0 to 1. This was necessary as errors were found to occur on the default setting of Option Base 0.

**Function 1 - moles\_reaction**

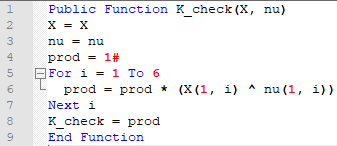




**Line-by-line Explanation**

|  |  |
| --- | --- |
| **Line Number** | **Explanation** |
| 1 | Function declaration with an argument of a 2D array, containing the ν and n0 values, and of the cell containing K. |
| 2 | Create and store the ‘proper’ 2D array. |
| 3 | Define a new 1D array, w, as having 6 double values. |
| 4 | Define a new variable, j, as an integer. |
| 5 | Define a new variable, w\_min, as a double. |
| 6-12 | Being the code to decide the limiting reagent. If a zero value is located in the stoichiometry, ignore that value by writing a zero value to the array as it’s not involved in the reaction (this avoids errors occurring from dividing by zero).  Else, the moles of the component are divided by its stoichiometry and the result is written to the array, w. Notice the abs() operator. This is needed because the reactants will have negative stoichiometry values, but the reaction extent must be positive.  So, once this is completed, the array, w, will contain 6 values corresponding to the maximum extent of reaction for that component, which are greater than or equal to 0. |
| 13-14 | Define the upper and lower bounds of the array, w. |
| 15 | Define a new variable, mini, containing an arbitrary large number. (1000000 is more than sufficient). |
| 16-18 | From the upper to the lower bound of the array inclusive, check whether each stored value is less than the variable mini and is greater than zero (zeros are ignored as they do not take place in the reaction or are products).  If the checked array value fulfils these conditions, it becomes the new value of the variable mini. |
| 19 | It is advisable to choose an n0\* value close to the maximum n\* value, so a value of n0\* = 0.99999xn\* is used. |
| 20 | This is used as the upper bound of the allowed values. |
| 21 | The lower bound of the allowed values is found as 0.000001xn\*. |
| 22 | Define a new variable, i. |
| 23 | Start the main iterative loop. |
| 24 | Define the number of iterations that will have taken place after this loop. |
| 25-29 | See **Iteration Safety Feature**. An extra line (line 27) was needed to actually give an answer if this safety feature was needed. |
| 30-33 | Calculate the sum equilibrium moles for each element of the array according to:  Where: |
| 34-37 | Calculate ,  Where: |
| 38-41 | Calculates the numerical derivative of using a distance of 10-6 between the two points. N.B: f\_diff must be 1 rather than 0 else the total product will be 0. |
| 42-45 | Calculates the relative difference between the values. |
| 46-48 | Calculates the new value for n\* to finish the iteration. |
| 49-60 | Tests if the new value is suitable by comparing it to the boundaries.  This equation can be used for i values greater than 1 where the calculated value is greater than the upper bound. However, this may occur within the first iteration, therefore a new equation must be used, which I have taken as the mean of the calculated value and the lower bound.  If the calculated value is less than the lower bound (extremely rare), then a new value is chosen between the lower bound and previously calculated value. Again, this could theoretically occur within the first iteration, so the aforementioned first iteration equation is used again.  If the calculated value does not apply to any of these, it is suitable and is taken as such, and is written to a new variable to be stored as the previously calculated value for the next iteration’s value tests. |
| 61 | Sets the loop condition according to: |
| 62 | Writes the final result for ñ\* to the spreadsheet. |
| 63 | Ends the function. |

**Function 2 – K\_check**

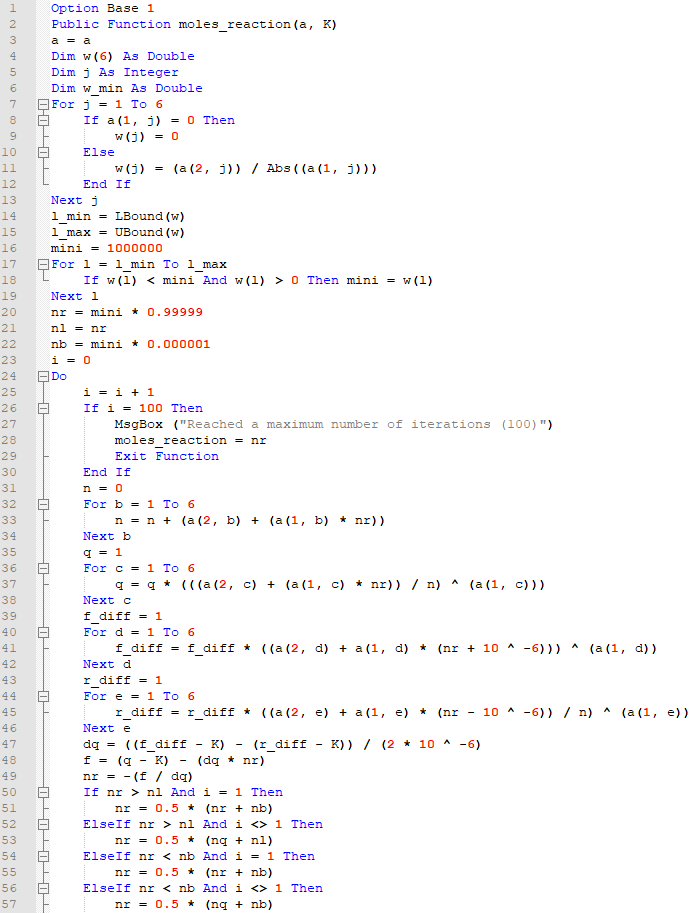


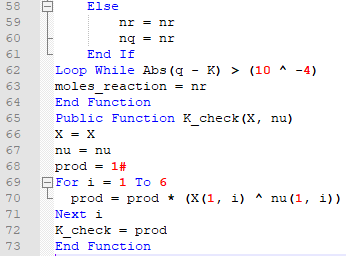
**Line-by-line Explanation**

|  |  |
| --- | --- |
| **Line Number** | **Explanation** |
| 1 | Declares a new function, K\_check, with an argument of an array of mole fractions and an array of corresponding ν values. |
| 2-3 | Create and store the ‘proper’ arrays. |
| 4 | Define a new variable, prod, as having a value of 1 as a double. |
| 5-7 | For each component in the array perform: |
| 8 | Writes the result for the check of K. |
| 9 | Ends the function. |

**Appendix I**

**Full Code**





**References**

Bock, H. (2018). *Visual Basic for Design - Assignment 2 - Chemical Equilibrium - Newton's Method with Boundaries.* Edinburgh: Heriot-Watt University.

Falenta, N. (2018). *Assignment 2 - Tasks: 1-4 Guidelines (NOT Answers).* Edinburgh: Heriot-Watt University.

Ho, D. (2016). *About*. Retrieved from Notepad++: https://notepad-plus-plus.org/

Microsoft. (2018). *Visual Basic Guide*. Retrieved from Microsoft: https://docs.microsoft.com/en-us/dotnet/visual-basic/