**DAILY ASSESSMENT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **08/07/2020** | **Name:** | **Gaganashree P** |
| **Course:** | **Matlab from mathworld** | **USN:** | **4AL15EC 024** |
| **Topic:** | **Importing data,Lofical arrays** | **Semester & Section:** | **8th - A** |
| **GitHub Repository:** | **Gaganashree-P** |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
|  |

**REPORT –**

**Import Data from Files**

Use [table](https://in.mathworks.com/help/matlab/ref/table.html) to store tabular data that you can use later in fitting and analysis at the command line. Use [readtable](https://in.mathworks.com/help/matlab/ref/readtable.html) to import data without NONMEM® interpretation of column headers. The readtable function lets you use name-value pair arguments in which you can specify options such as the type of delimiter and whether the first row contains header names. If you want to use the data for fitting using sbiofit or sbiofitmixed, convert it to a groupedData format using [groupedData](https://in.mathworks.com/help/simbio/ref/groupeddata.html).

To prepare the data file for import, remove any comments that are present at the beginning of the file.

For example:

% Text files

data = readtable('tobramycin.txt');

% Text files with . in place of missing values

data = readtable('tobramycin.txt', 'TreatAsEmpty', '.');

**Importing Data from NONMEM-Formatted Files**

Use the sbionmimport function to import data from NONMEM formatted files. To import the data without NONMEM interpretation of column headers, see [Import Data from Files](https://in.mathworks.com/help/simbio/ug/importing-data.html#bsehazt-24).

To prepare the data file for import, remove any comments that are present at the beginning of the file and select one of the following methods to import your data:

* If the data file contains only the column header values shown in [Support for Importing NONMEM Formatted Files](https://in.mathworks.com/help/simbio/ug/importing-data-supported-files-and-data-types.html#bsehazs-20), use the syntax shown in the following example:
* filename = 'C:\work\datafiles\dose.xls';

ds = sbionmimport(filename);

* If the data file has column header labels different from the table shown in [Support for Importing NONMEM Formatted Files](https://in.mathworks.com/help/simbio/ug/importing-data-supported-files-and-data-types.html#bsehazs-20) and you want to apply NONMEM interpretation of headers:
  1. Create a NONMEM file definition object. This object lets you define what the column headers in the data file mean in SimBiology®. In the following example, the column containing response values is CP, whereas in NONMEM formatted files the column is labelled DV.

To use the tobramycin data set [[1]](https://in.mathworks.com/help/simbio/ug/pharmacokinetic-modeling-functionality.html#bsnipts), create a NONMEM file definition object and define the following:

def = sbionmfiledef;

def.DoseLabel = 'DOSE';

def.GroupLabel = 'ID';

def.TimeLabel = 'TIME';

def.DependentVariableLabel = 'CP';

def.MissingDependentVariableLabel = 'MDV';

def.EventIDLabel = 'EVID';

def.ContinuousCovariateLabels = {'WT', 'HT', 'AGE', 'SEX', 'CLCR'};

Your file can contain any name for column headings. See [sbionmfiledef](https://in.mathworks.com/help/simbio/ref/sbionmfiledef.html) for the list of properties you can configure in the NONMEM file definition object.

* 1. Use the sbionmimport function to import your data file with the column header definitions as specified in the NONMEM file definition object. For example, browse to *matlabroot*/toolbox/simbio/simbiodemos/ (where *matlabroot* is the folder where MATLAB® is installed).
  2. [data, pkDataObject] = sbionmimport('tobramycin.txt', def, ...

'TreatAsEmpty', '.');

This example shows you how to obtain the PKData object, PKDataObj, while importing, since you will use the PKData object in fitting the model later.

The sbionmimport function accepts property-name-value pairs accepted by dataset. For example, if the data set does not contain column headers, use 'ReadVarNames', false to specify that sbionmimport should read values from the first row of the file.

For information about creating a model to fit the data, see [Create a Pharmacokinetic Model Using the Command Line](https://in.mathworks.com/help/simbio/ug/creating-pharmacokinetic-models.html#bsehazz-50).

**Other Resources for Importing Data**

For detailed information about supported data formats and the functions for importing data into the MATLAB Workspace, see the [Methods for Importing Data](https://in.mathworks.com/help/matlab/import_export/recommended-methods-for-importing-data.html) (MATLAB).

You also can import data using the MATLAB Import Wizard (see [Import Images, Audio, and Video Interactively](https://in.mathworks.com/help/matlab/import_export/import-images-audio-and-video-using-import-wizard.html) (MATLAB). Use the Import Wizard, to import data as text files (such as .txt and .dat), MAT-files, and spreadsheet files, (such as .xls).

The MATLAB Import Wizard processes the data source. The wizard recognizes data delimiters, as well as row or column headers, to facilitate the process of data selection and importation into the MATLAB Workspace. You can import the data to the SimBiology Model Analyzer app from the MATLAB Workspace.

**Logical Arrays**

This example shows how to filter the elements of an array by applying conditions to the array. For instance, you can examine the even elements in a matrix, find the location of all 0s in a multidimensional array, or replace NaN values in data. You can perform these tasks using a combination of the relational and logical operators. The relational operators (>, <, >=, <=, ==, ~=) impose conditions on the array, and you can apply multiple conditions by connecting them with the logical operators and, or, and not, respectively denoted by the symbols &, |, and ~.

### Apply a Single Condition

To apply a single condition, start by creating a 5-by-5 matrix that contains random integers between 1 and 15. Reset the random number generator to the default state for reproducibility.

rng default

A = randi(15,5)

A = 5×5

13 2 3 3 10

14 5 15 7 1

2 9 15 14 13

14 15 8 12 15

10 15 13 15 11

Use the relational less than operator, <, to determine which elements of A are less than 9. Store the result in B.

B = A < 9

B = 5x5 logical array

0 1 1 1 0

0 1 0 1 1

1 0 0 0 0

0 0 1 0 0

0 0 0 0 0

The result is a logical matrix. Each value in B represents a logical 1 (true) or logical 0 (false) state to indicate whether the corresponding element of A fulfills the condition A < 9. For example, A(1,1) is 13, so B(1,1) is logical 0 (false). However, A(1,2) is 2, so B(1,2) is logical 1 (true).

Although B contains information about which elements in A are less than 9, it doesn’t tell you what their values are. Rather than comparing the two matrices element by element, you can use B to index into A.

A(B)

ans = 8×1

2

2

5

3

8

3

7

1

The result is a column vector of the elements in A that are less than 9. Since B is a logical matrix, this operation is called **logical indexing**. In this case, the logical array being used as an index is the same size as the other array, but this is not a requirement. For more information, see [Array Indexing](https://in.mathworks.com/help/matlab/math/array-indexing.html).

Some problems require information about the locations of the array elements that meet a condition rather than their actual values. In this example, you can use the find function to locate all of the elements in A less than 9.

I = find(A < 9)

I = 8×1

3

6

7

11

14

16

17

22

The result is a column vector of linear indices. Each index describes the location of an element in A that is less than 9, so in practice A(I) returns the same result as A(B). The difference is that A(B) uses logical indexing, whereas A(I) uses linear indexing.

### Apply Multiple Conditions

You can use the logical and, or, and not operators to apply any number of conditions to an array; the number of conditions is not limited to one or two.

First, use the logical and operator, denoted &, to specify two conditions: the elements must be **less than 9**and**greater than 2**. Specify the conditions as a logical index to view the elements that satisfy both conditions.

A(A<9 & A>2)

ans = 5×1

5

3

8

3

7

The result is a list of the elements in A that satisfy both conditions. Be sure to specify each condition with a separate statement connected by a logical operator. For example, you cannot specify the conditions above by A(2<A<9), since it evaluates to A(2<A | A<9).

Next, find the elements in A that are **less than 9** and **even numbered**.

A(A<9 & ~mod(A,2))

ans = 3×1

2

2

8

The result is a list of all even elements in A that are less than 9. The use of the logical NOT operator, ~, converts the matrix mod(A,2) into a logical matrix, with a value of logical 1 (true) located where an element is evenly divisible by 2.

Finally, find the elements in A that are **less than 9** and **even numbered** and **not equal to 2**.

A(A<9 & ~mod(A,2) & A~=2)

ans = 8

The result, 8, is even, less than 9, and not equal to 2. It is the only element in A that satisfies all three conditions.

Use the find function to get the index of the element equal to 8 that satisfies the conditions.

find(A<9 & ~mod(A,2) & A~=2)

ans = 14

The result indicates that A(14) = 8.

### Replace Values That Meet a Condition

Sometimes it is useful to simultaneously change the values of several existing array elements. Use logical indexing with a simple assignment statement to replace the values in an array that meet a condition.

Replace all values in A that are greater than 10 with the number 10.

A(A>10) = 10

A = 5×5

10 2 3 3 10

10 5 10 7 1

2 9 10 10 10

10 10 8 10 10

10 10 10 10 10

Next, replace all values in A that are not equal to 10 with a NaN value.

A(A~=10) = NaN

A = 5×5

10 NaN NaN NaN 10

10 NaN 10 NaN NaN

NaN NaN 10 10 10

10 10 NaN 10 10

10 10 10 10 10

Lastly, replace all of the NaN values in A with zeros and apply the logical NOT operator, ~A.

A(isnan(A)) = 0;

C = ~A

C = 5x5 logical array

0 1 1 1 0

0 1 0 1 1

1 1 0 0 0

0 0 1 0 0

0 0 0 0 0

The resulting matrix has values of logical 1 (true) in place of the NaN values, and logical 0 (false) in place of the 10s. The logical NOT operation, ~A, converts the numeric array into a logical array such that A&C returns a matrix of logical 0 (false) values and A|C returns a matrix of logical 1 (true) values.

**DAILY ASSESSMENT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **8/07/2020** | **Name:** | **Gaganashree P** |
| **Course:** | **IOT** | **USN:** | **4AL15EC024** |
| **Topic:** | **Chapter 5** | **Semester & Section:** | **8TH SEM &A Section** |
| **Github Repository:** | **Gaganashree-P** |  |  |

|  |
| --- |
| **AFTERNOON SESSION DETAILS**   ****IoT:Everything needs to be secure**** **1. IoT endpoint security vs network security**  Securing IoT devices is a real challenge. IoT devices are highly diversified, with a wide variety of operating systems (real-time operating systems, Linux-based or bare-metal), communication protocols and architectures. On top of the high diversity, comes the issues of low resources and lack of industry standards and regulations. Most security solutions today focus on securing the network (discover network anomalies and achieve visibility into IoT devices that are active in the network), while the understanding that the devices themselves must be protected is now establishing. The fact that IoT devices can be easily exploited makes them a very good target for attackers, [aiming to use the weak IoT device as an entry point to the entire enterprise network](https://www.computerworld.com/article/2487452/target-attack-shows-danger-of-remotely-accessible-hvac-systems.html), without being caught. Besides that, it’s important to remember that network solutions are irrelevant for distributed IoT devices (i.e., home medical devices), that has no network to protect them.  Manufacturers of IoT devices are therefore key for a secure IoT environment and more and more organizations are willing to pay more for built-in security into their smart devices.  **2. “Cryptography is typically bypassed, not penetrated” *Shamir’s law***  In recent years we see a lot of focus on IoT data integrity, which basically means encryption & authentication. Though very important by itself, it’s important to understand that encryption doesn’t mean full security. When focusing mainly on encryption & authentication, companies forget that the devices are still exposed to cybersecurity vulnerabilities that can be used to penetrate the device and receive access into the **decrypted information**, thus bypassing the authentication and encryption entirely. In other words, what’s known for years in the traditional cyber industry as Shamir’s law should  now make its way to the IoT security industry: “Cryptography is typically bypassed, not penetrated” and therefore companies must invest in securing their devices from cyber attacks and not just handle data integrity. To read more about that, please visit [**Sternum IoT Security two-part blog post**](https://www.sternumiot.com/blog/2019/4/8/is-encryption-enough-for-security-part-1).  **3. 3rd party IoT vulnerabilities**  One of the main issues in IoT security is the heavily reliance of IoT devices on third-party components for communication capabilities, cryptographic capabilities, the operating system itself etc. In fact, this reliance is so strong that it has reached a point where it’s unlikely to find an IoT device without third-party components within it. The fact that third-party libraries are commonly used across devices, combined with the difficulty to secure them, makes them a sweet spot for hackers to look for IoT vulnerabilities and exploit many IoT devices through such 3rd party component.  Vulnerability in third-party components is very dangerous. In many IoT devices, there is no separation and segmentation between processes and/or tasks, which means that even one vulnerability in a third-party library is compromising the entire device. This could lead to lethal results: attackers can leverage the third-party vulnerability to take control over the device and cause damage, steal information of perform a ransomware attack on the manufacturer.  it’s not only that third-party components are dangerous, but they are also extremely difficult to secure. Many third-party components are delivered in binary form, with no source code available. Even when the source code is available, it’s often hard to dive into it and asses the security level or vulnerabilities inside it. Either way, most developers use the open-source components as black-boxes. On top of that, static analysis tools and compiler security flags lack the ability to analyze and secure third-party components and most IoT security solutions cannot offer real-time protection into binary code.  **VXWORKS VULNERABILITIES**  A recent example of such third party vulnerability that affects millions of devices can be found in the security bugs found in the [**VxWorks embedded operating system**](https://www.wired.com/story/vxworks-vulnerabilities-urgent11/). These vulnerabilities exposed every manufacturer that used VxWorks operating system, even if security measures like penetration testing, static analysis, PKI and firmware analysis were taken.  To summarize, in order to provide strong and holistic IoT protection, you must handle and secure all parts of the device, including the third-party components. [**Sternum IoT security**](https://www.sternumiot.com/) solutions focus on holistically securing IoT devices from within and therefore offers a unique capability of embedding security protection & visibility into the device from end-to-end. Sternum’s solution is also operating during real-time execution of the device and prevents all attack attempts at the exact point of exploitation, while immediately alerting about the attack and its origins, including from within third-party libraries.  **4. Regulation is kicking in**  In the past two years, we’re seeing a across industries effort to create regulations and standards for IoT security. We are expecting to see more of these efforts shaping into real regulations that will obligate manufacturers to comply with them.  A good and important example is the [**FDA premarket cybersecurity guidance**](https://www.sternumiot.com/fda-compliance) that was published last year and is expected to become a formal guidance in 2020. The guidance includes different aspects of cybersecurity in medical devices (which is in many cases are essentially IoT devices) such as data integrity, Over-the-air updates, real-time protection, execution integrity, third-party liabilities and real-time monitoring of the devices.  Another example is the [**California Internet of Things cybersecurity law**](https://www.theverge.com/2018/9/28/17874768/california-iot-smart-device-cybersecurity-bill-sb-327-signed-law) that states: Starting on January 1st, 2020, any manufacturer of a device that connects “directly or indirectly” to the internet must equip it with “reasonable” security features, designed to prevent unauthorized access, modification, or information disclosure.  We expect to see more states and countries forming regulations around IoT security since these devices lack of security may have a dramatic effect on industry, cities, and people’s lives. Top two regulations that are about to be released are the new EU Cybersecurity Act (based on ENISA and ETSI standards) and the NIST IoT and Cybersecurity framework. |