

MTHE 393 Background

Department of Mathematics and Statistics
Queen's University
Kingston, Ontario, Canada

04/01/2022

Problem Definition

Determine a linear, time-invariant state-space or transfer function that heuristically models an unknown system.

The unknown system is a black box. Input signals can be sent to the system and output signals can be read from the system. But, you do not have any knowledge of its internal workings.



From the input-output behaviour of the black box, you must find a reasonable approximation of its inner workings.

Some of the main problems to be analysed include but are not limited to:

- What is a system and what are the types of systems?
- What can the input-output behaviour tell you about the system?
- How does your model take into account: non-linearity, time variance, uncertainty in the parameters, noise, etc.
- Why is your model heuristic?
- Does your model behave like the unknown system?

Once you have modeled the system, you must create a feedback loop that contains the black box and a controller of your own design. Your goal is to optimize the output when the input is a step function. Your controller must be chosen such that certain performance criteria is met. This criteria includes the steady state error, rise time, overshoot, and settling time. Each system will have specific targets that must be achieved. These will be given to you at some appropriate time.

Mathematical Background

Introduction and Terminology. For this course, there should be from the outset a picture you have in mind of what you are trying to accomplish. The picture is essentially given in Figure 1.

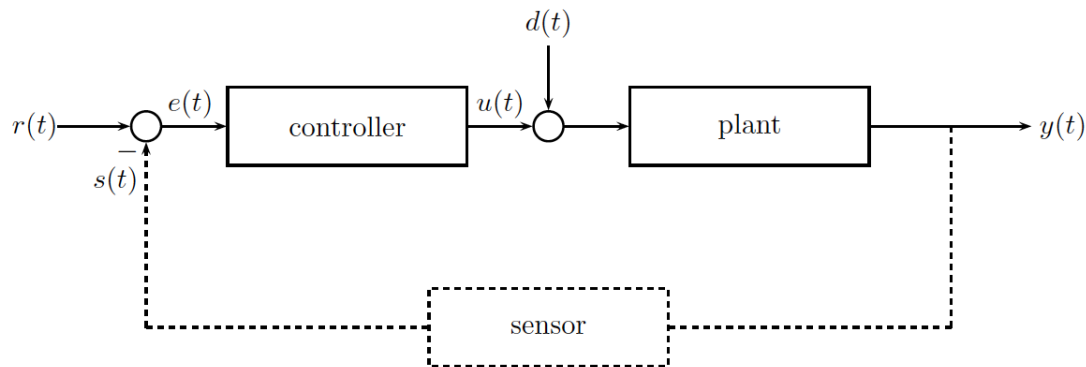


Figure 1: A basic control system schematic

The idea is that you are given a **plant**, which is the basic system, which has an output $y(t)$ that you'd like to do something with. For example, you may wish to track a **reference trajectory** $r(t)$. One way to do this would be to use an **open-loop** control design. In this case, one would omit that part of the diagram in Figure 1.1 which is dashed, and use a **controller** to read the reference signal $r(t)$ and use this to specify an **input** $u(t)$ to the plant which should give the desired output. This open-loop control design may well work, but it has some inherent problems. If there is a **disturbance** $d(t)$ which you do not know about, then this may well cause the output of the plant to deviate significantly from the reference trajectory $r(t)$. Another problem arises with plant **uncertainties**. One models the plant, typically via differential equations, but these are always an idealisation of the plant's actual behaviour. The reason for the problems is that the open-loop control law has no idea what the output is doing, and it marches on as if everything is working according to an idealised model, a model which just might not be realistic. A good way to overcome these difficulties is to use **feedback**. Here the output is read by **sensors**, which may themselves be modelled by differential equations, which produce a signal $s(t)$ which is subtracted from the reference trajectory to produce the **error** $e(t)$. The controller then make its decisions based on the error signal, rather than just blindly considering the reference signal. These are the basic components of a control system in block diagram form. These blocks can be moved around, there can be multiple sensors, multiple controllers, or no disturbance, but the basic elements of a control system are given in this model.

System Representations.

State-Space Representations

The state-space representation for a system is its representation in the time domain. The form of a **single-input, single-output** system is given as

$$\begin{aligned}\dot{\mathbf{x}}(t) &= \mathbf{A}\mathbf{x}(t) + \mathbf{B}u(t), \\ y(t) &= \mathbf{C}\mathbf{x}(t) + Du(t).\end{aligned}$$

Here $\mathbf{x} \in \mathbb{R}^n$ is the **state** of the system, $u \in \mathbb{R}$ is the **input**, and $y \in \mathbb{R}$ is the **output**. We call the system finite-dimensional because n is finite and time-invariant because \mathbf{A} , \mathbf{B} , \mathbf{C} and D are constant. For convenience this system is represented by $\Sigma=(\mathbf{A},\mathbf{B},\mathbf{C},D)$.

Transfer Functions

To define our transfer function, we first start with our state-space representation, and transform it into the Laplace domain. We suppose we are given a SISO linear system $\Sigma=(\mathbf{A},\mathbf{B},\mathbf{C},D)$, and we fiddle with Laplace transforms a bit for such systems. Note that one takes the Laplace transform of a vector function of time by taking the Laplace transform of each component. Thus we can take the left causal Laplace transform of the linear system

$$\begin{aligned}\dot{\mathbf{x}}(t) &= \mathbf{A}\mathbf{x}(t) + \mathbf{B}u(t), \\ y(t) &= \mathbf{C}\mathbf{x}(t) + Du(t).\end{aligned}$$

to get

$$\begin{aligned}s\mathcal{L}_{0-}^+(\mathbf{x})(s) &= \mathbf{A}\mathcal{L}_{0-}^+(\mathbf{x})(s) + \mathbf{B}\mathcal{L}_{0-}^+(u)(s), \\ \mathcal{L}_{0-}^+(y)(s) &= \mathbf{C}\mathcal{L}_{0-}^+(\mathbf{x})(s) + D\mathcal{L}_{0-}^+(u)(s).\end{aligned}$$

It is convenient to write this in the form

$$\begin{aligned}\mathcal{L}_{0-}^+(\mathbf{x})(s) &= (s\mathbf{I}_n - \mathbf{A})^{-1}\mathbf{B}\mathcal{L}_{0-}^+(u)(s), \\ \mathcal{L}_{0-}^+(y)(s) &= \mathbf{C}\mathcal{L}_{0-}^+(\mathbf{x})(s) + D\mathcal{L}_{0-}^+(u)(s).\end{aligned}$$

We note that we may, in the Laplace transform domain, solve explicitly for $\mathcal{L}_{0-}^+(y)$ in terms of $\mathcal{L}_{0-}^+(u)$ to get

$$\mathcal{L}_{0-}^+(y)(s) = \mathbf{C}(s\mathbf{I}_n - \mathbf{A})^{-1}\mathbf{B}\mathcal{L}_{0-}^+(u)(s) + D\mathcal{L}_{0-}^+(u)(s)$$

Note that we may write

$$T_{\Sigma}(s) = \frac{\mathcal{L}_{0-}^+(y)(s)}{\mathcal{L}_{0-}^+(u)(s)} = \mathbf{C}(s\mathbf{I}_n - \mathbf{A})^{-1}\mathbf{B} + D.$$

This defines our **transfer function** T_{Σ} for the linear system $\Sigma = (\mathbf{A}, \mathbf{B}, \mathbf{C}, D)$.

Frequency Response

When studying signals, we often think of how they behave over time. One of the issues with analyzing systems in the *time domain* is that it can be difficult in high order systems. Instead, these systems are transferred to the *frequency domain* using the Fourier transform or the *s domain* using the Laplace transform. In the frequency- or *s*-domain, the system can be easily manipulated and analyzed. (Note: Fourier and Laplace are equivalent when $s = i\omega$).

Let us say we have the input signal $\sin(\omega t)$, where ω is the frequency. A linear system will produce the output $A\sin(\omega t + \theta)$. The output will be at the same frequency, ω , but, the magnitude, A , and phase, θ , are not the same as the input. So, how do the input and output relate?

When we take a system into the frequency domain, we can look at its *frequency response*. This function takes the input signal's frequency and returns a complex number that allows us to determine the output signal's magnitude and phase.

We will start off by defining the frequency response, and then give its interpretation. For a SISO linear control system $\Sigma=(\mathbf{A},\mathbf{B},\mathbf{C},D)$ we let $\Omega_\Sigma \subset \mathbb{R}$ be defined by

$$\Omega_\Sigma = \{\omega \in \mathbb{R} \mid i\omega \text{ is a pole of } T_\Sigma\}.$$

The **frequency response** for Σ is the function $H_\Sigma : \mathbb{R} \setminus \Omega_\Sigma \rightarrow \mathbb{C}$ defined by $H_\Sigma(\omega) = T_\Sigma(i\omega)$. Note that we *do* wish to think of the frequency response as a \mathbb{C} -valued function, and not a rational function, because we will want to graph it. Thus when we write $T_\Sigma(i\omega)$, we intend to evaluate the transfer function at $s = i\omega$. In order to do this, we suppose that all the poles and zeroes of $T_\Sigma(i\omega)$ have been cancelled.

System Analysis.

Stability

Control theory is the study of the behaviour of systems. One of the main goals of control theory is to have stability. In simplest terms, the output should not approach infinity. This is an important property for real world systems; an unstable system can cause problems such as explosions.

We must determine how a system “reacts” to given input and/or initial conditions. Depending on how the system behaves, it can be classified into different categories of stability. Using certain theorems, stability can be useful in determining the properties of a system. Some more rigorous definitions of stability are below.

A SISO linear system $\Sigma=(\mathbf{A},\mathbf{B},\mathbf{C},D)$ is

(i) **internally stable** if

$$\limsup_{t \rightarrow \infty} \|\mathbf{x}(t)\| < \infty$$

for every solution $\mathbf{x}(t)$ of $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}(t)$;

(ii) **internally asymptotically stable** if

$$\lim_{t \rightarrow \infty} \| \mathbf{x}(t) \| = 0$$

for every solution $\mathbf{x}(t)$ of $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}(t)$;

(iii) **internally unstable** if it is not internally stable.

Internal stability has nothing to do with any part of Σ other than the matrix \mathbf{A} , and stability can almost be determined from the spectrum of \mathbf{A} . Although internal stability is important, we are mostly concerned with input/output stability. That is we want nice inputs to produce nice outputs. We now provide another notion of stability called BIBO stability.

A SISO linear system Σ is **bounded input, bounded output stable (BIBO stable)** if there exists a constant $K > 0$ such that, $\mathbf{x}(0)=\mathbf{0}$, and $|u(t)| \leq 1$ for $t \geq 0$ imply that $y(t) \leq K$.

Bode and Nyquist Plots

A Bode plot is a graphical representation of a system's *frequency response*. It depicts the gain and phase of an LTI system at different frequencies. A Bode plot can give a lot of information about the system if you know what to look for.

The Nyquist Criterion is a test to determine if a closed-loop system is stable. This is accomplished by using a Nyquist plot, which is closely related to the Bode plot. A Nyquist plot is the polar representation of a Bode plot - the phase is the angle θ and the magnitude is the radius r . The outcome of the Nyquist Criterion has several implications.

Feedback

Although a control systems can have any number of loops, SISO control typically deals with the simple case where we have an interconnection with one loop. We first look at the typical single loop control problem with a plant transfer function that is to be modified by a controller transfer function. In particular, we say what we mean by open-loop and closed-loop control.

We start with a rational function $R_P \in \mathbb{R}(s)$ that describes the **plant**, and also a rational function $R_C \in \mathbb{R}(s)$ which is our controller. The plant rational function should be thought of as unchangeable, which we are allowed to vary our controller rational function. One could simply use an **open-loop** controller and design R_C so that the **open-loop transfer function** $R_P R_C$ has the desired properties. This corresponds to the situation in Figure 2 below.

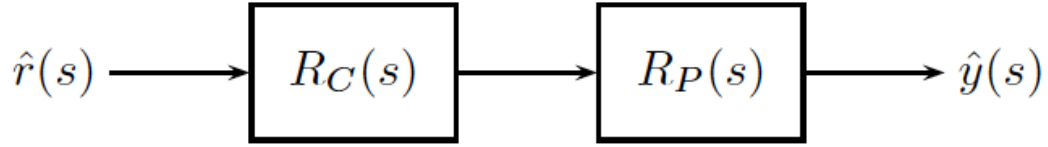


Figure 2: An open-loop control system

There are some issues that arise from open-loop control designs, to get around this we design a controller to act not on the reference signal \hat{r} , but on the **error signal** $\hat{e} = \hat{r} - \hat{y}$. One may place the controller in other places in the block diagram, and one may have other rational functions in the block diagram. However, for such systems, the essential methodology is the same, and so let us concentrate on one type of system interconnection for the moment for simplicity. The block diagram configuration for the so-called **closed-loop** system we consider is depicted in Figure 3 below.

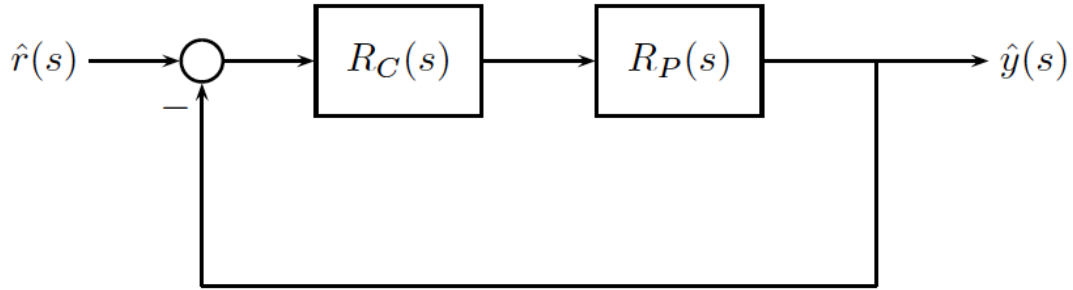


Figure 3: An closed-loop control system

The **closed-loop transfer function** from \hat{r} to \hat{y} is readily computed as

$$\frac{\hat{y}}{\hat{r}} = \frac{R_P R_C}{1 + R_P R_C}.$$

White Noise and Filters

A continuous time white noise process $w(t)$, $t \in \mathbb{R}$ is a random process which has the property that $\mathbb{E}(w) = 0$. White noise is encountered in practice almost everywhere in engineering. Luckily, in most situations the noise is small and observations can be deduced without having to worry about removing it. In other applications, the noise is so strong, that it is imperative to remove the noise in order to accurately process the data. This is where the idea of filters comes in. Filters are used in practice to

remove some unwanted component of feature from a signal, e.g. white noise. Different filters have different applications, and there is not one filter that will work well for every signal. It is up to the engineer to know the system, and define what time of disturbance the signal has, in order to properly construct a filter which eliminates the disturbance.

How to Use the GUI

IMPORTANT: Before getting started with the GUI (Graphical User Interface) your group should slowly and carefully read through this user guide in order to save both yourself, and your teaching assistant some trouble.

Getting Started. Unzip the zip file and you will get these files in a folder called BlackBox:

- blackBox.fig
- blackBox.p
- stateSpace.mdl
- csfunc.p
- inputFunc.m
- inputManual.m

The GUI

The GUI (graphical user interface) is a program written in Matlab which will be provided to you by your teaching assistant. It will comprise of several different files, one being "blackBox.p", a Matlab file, and the second being "blackBox.fig", a figure file. Although you have access to any ".m" file, the ".p" means it is protected and cannot be viewed or edited; it can only be run in Matlab. Another file given to you is a ".mdl" file, a model file. These files can be opened in Simulink and edited (see below for what you should and should not edit).

The main focus should be spent testing various solutions via the .fig and .mdl files. There are also some additional ".m" files. Their purpose will be explained later. If, for whatever reason, your group needs to make a change to any of the .m or .mdl files to accommodate your solution, you can talk to your teaching assistant for help. If your group decides that you want to make changes to files by yourselves, it is suggested that a duplicate of the original is saved under a different name so that an original copy is always available.

Matlab and Simulink

1. Open Matlab
2. Run Simulink by typing *Simulink* into the Command Window
3. Open the *stateSpace.mdl* and you will get Figure 4. **You need to have this file open for the GUI to work.**

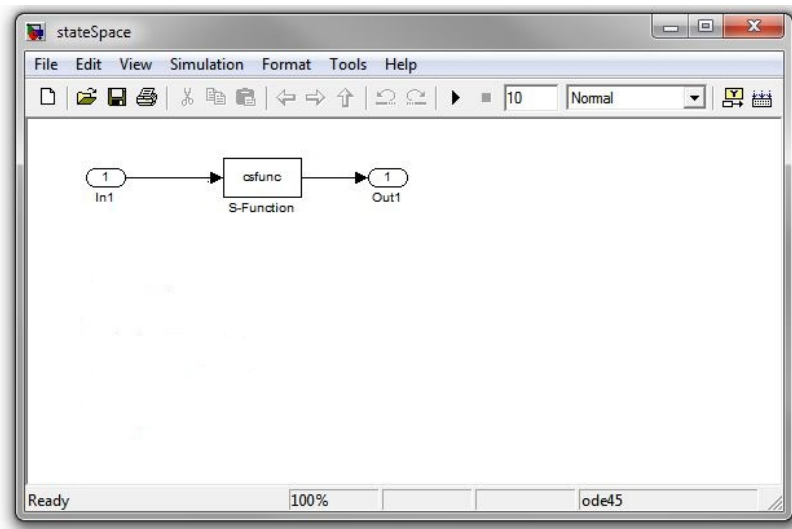


Figure 4: Simulink model of the black box.

4. **In1**, **Out1** and **csfunc** blocks must remain the way they are, with the settings they have. Also, do not modify the **Configuration Parameters** as seen in Figure 5 without talking to the teaching assistant.

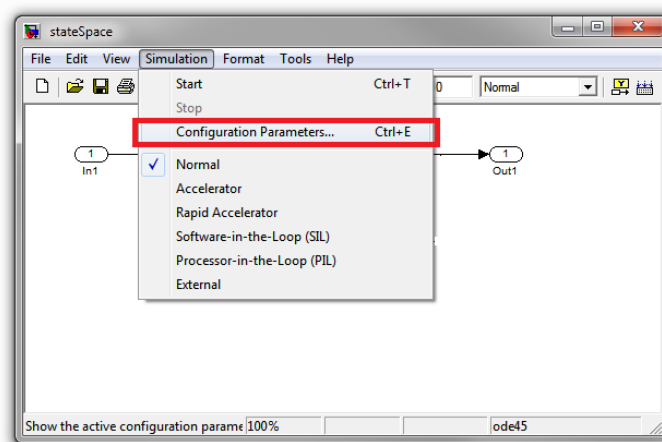


Figure 5: Simulink configuration parameters that you should not modify

5. At any time, you can add blocks to the model that may help with identifying the system. Just remember to save before you run the GUI. For example, you can add a controller and make it a closed-loop system as seen in Figure 6.

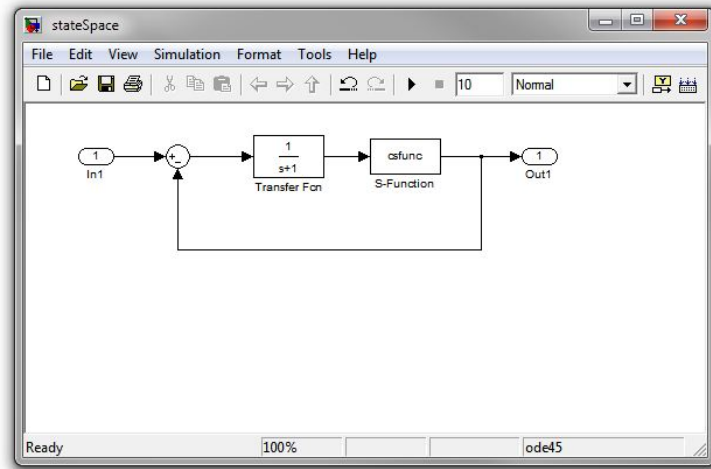


Figure 6: Modified Simulink model of the black box.

6. Open up the GUI by typing *blackBox* into the Command Window. The GUI will appear as seen in Figure 7.

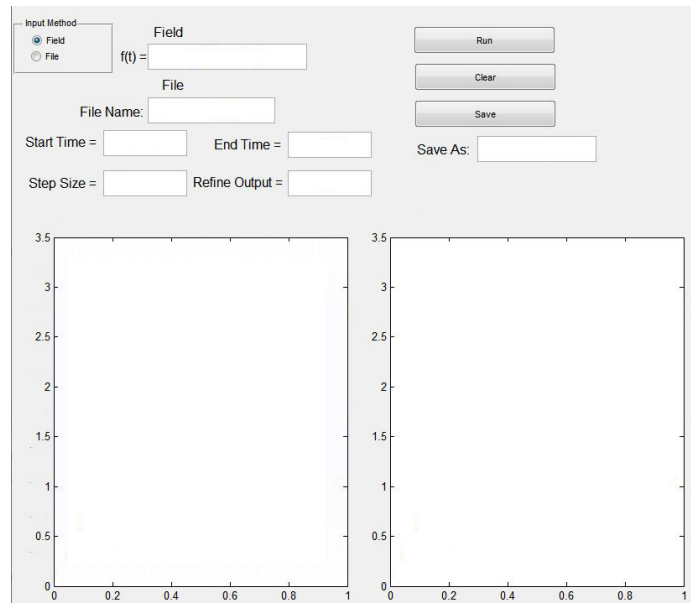


Figure 7: GUI for the black box

Settings for the GUI

There are several options for the GUI. We will go over each option now.

Input Method - $f(t)$ - this allows you to write a function for the input signal. More details can be found in *Input Functions* below.

Input Method - File Name - this allows you to write a function for the input signal. More details can be found in *Input Functions* below.

Start Time - the time the simulation starts at

End Time - the time the simulation ends at

Step Size - signals are made of discrete points spaced at this step size apart

Refine Output - output signals can be made to have a smaller step size

Run - this will cause the simulation to run and the input will be graphed on the left and the output will be graphed on the right. More details can be found in *Running the Simulations* below.

Clear - this will clear the graphs and text fields

Save - this will save the input and output values as *file_name.mat*. More details can be found in *Saving the Results* below.

Input Functions. There are two ways to create the input function used by the GUI. To change between these methods, look at the GUI in the top left for the *Input Method* box. There are two radio buttons. Click on the button for the method you want.

(i) *GUI Method - the Field button*

For this method, you input the function into the specified text field.

1. Look for the text field with $f(t)$ beside it, as seen in Figure 8.

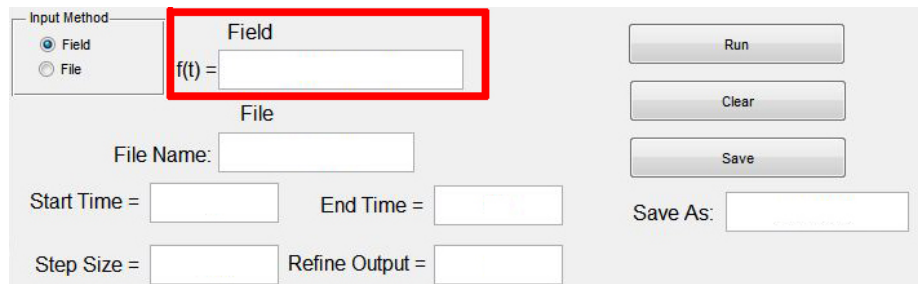
The image shows a software interface for setting simulation parameters. At the top left, under 'Input Method', there are two radio buttons: 'Field' (which is selected) and 'File'. To the right of these is a text field labeled 'f(t) ='. This text field is enclosed in a red rectangular box. Below the 'Field' section, there is a 'File' section with a 'File Name:' label and a text field. Further down, there are labels for 'Start Time =', 'End Time =', 'Step Size =', and 'Refine Output =', each followed by a text field. On the right side of the interface, there are three buttons: 'Run', 'Clear', and 'Save'. Below the 'Save' button is a 'Save As:' label followed by a text field.

Figure 8: Type in the function into the text field that is circled.

2. Write the function you want into this field. Note: the variable in the function will be with respect to t .

- You can write any polynomial

$$f(t) = \boxed{t^3 + 5 * t^2 + 1}$$

- You can use any Matlab function

$$f(t) = \boxed{\sin(t)}, f(t) = \boxed{\exp(t)}, f(t) = \boxed{\text{heaviside}(t)}$$

- You can use irrational numbers

$$f(t) = \boxed{2 * \pi * t + \text{sqrt}(2)}$$

- The possibilities are endless so play around and if you run into a problem, ask the teaching assistant for help.

Note that the multiplication between each of the terms is denoted by an asterisk(*).

3. When you are done typing in the function, press *ENTER*.
4. Make sure the radio button in the top left is set to *Field* or else this method won't work.

(ii) Matlab File Method - the File button

For this method, you need to have a separate Matlab file with your own function.

1. Open *inputManual.m*. This file gives the basic layout for what the function file has to be. **It also gives an example of how you can write the function in the file.**
2. It is suggested that you either copy and paste this into a new file or save this file under a new name.
3. Write the function you want into *your new file*. Note: the variable in the file will be t .
 - You can put *if statements* - this can help you to create discontinuous functions
 - You can use any Matlab function - $\sin(t)$, $\text{heaviside}(t)$, $\exp(t)$
 - You can use irrational numbers - π , $\sqrt{2}$
 - The possibilities are endless so play around and if you run into a problem, ask the teaching assistant for help.
4. Remember to save the file.
5. Make sure the radio button in the top left is set to *File* or else this method won't work.

6. Now, enter in the file name into the text field seen in Figure 9.

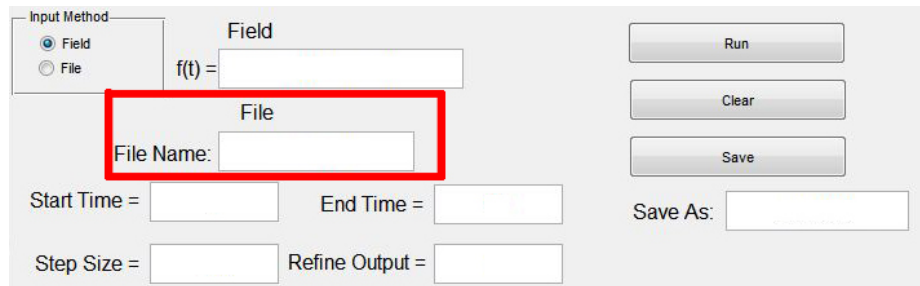


Figure 9: Type in the file name to the text field that is circled.

Let's say the file you want to use is called "*inputManual.m*". You would type into the text field:

FileName :

Running the Simulation.

- Here's a checklist of what you should have done by now:
 - Simulink is open with *stateSpace.mdl*. Any blocks you wanted to add are there.
 - The GUI is open.
 - You have an input function set up.
- Set the start and stop times by filling in the corresponding text fields as seen in Figure 10.

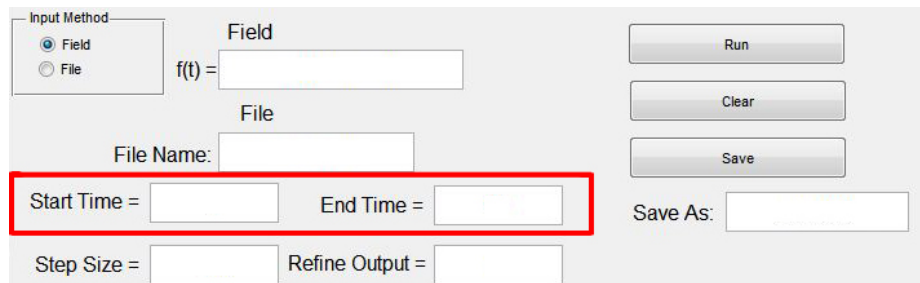


Figure 10: Type in the start and end times in the text field that is circled.

These values should be reasonable. For example:

- the start time must be 0 or above (no negative time)
- the stop time cannot be less than the start time
- if you are having problems with the times, go talk to the teaching assistant for help.

3. You can set the step size for the **input and output signal** by changing the value in the text field as seen in Figure 11.

Figure 11: Type in the step size in the text field that is circled.

The step size should be chosen accordingly. If your function is changing rapidly, you may need a smaller step size. To show how the step size works, let the start time be 0 and the end time be 1 with a step size of 0.01. The array of time values will be:

$$Time = [0 \ 0.01 \ 0.02 \ 0.03 \ \dots \ 0.98 \ 0.99 \ 1.0]$$

4. The refine factor will give a smoother output more efficiently than just modifying the step size. It does this by generating more points between steps for **only the output signal**. This value can be changed in the text field as seen in Figure 12.

Figure 12: Type in a refine factor in the text field that is circled.

The refine factor must be an integer greater than zero. To show how the refine factor works, let's have a step size of 0.01 and just look at one time step.

With a refine factor of 1,

$$Time = [0 \ 0.01]$$

With a refine factor of 4,

$$Time = [0 \ 0.0025 \ 0.0050 \ 0.0075 \ 0.01]$$

5. Click the *Run* button.

- Depending on the magnitude of the time interval you choose, it could take a couple of seconds.
- When the simulation has completed, they will be automatically graphed. The input will be graphed on the left and the output will be graphed on the right.
- If you changed the input function, but the graphs remained the same, **try pressing Run again.**
- If errors show up in the Matlab Command Window, make sure everything was done correctly.
- If there are still problems, talk to the teaching assistant.

Saving the Results.

- Enter a name into the text field beside the Save button as seen in Figure 13.

The figure shows a MATLAB GUI with the following elements:

- Input Method:** Radio buttons for 'Field' (selected) and 'File'.
- Field:** A text field labeled 'f(t) ='.
- File:** A text field labeled 'File Name:'.
- Time and Step Settings:** Text fields for 'Start Time =', 'End Time =', 'Step Size =', and 'Refine Output ='.
- Buttons:** 'Run', 'Clear', and 'Save' buttons.
- Save As:** A text field next to the 'Save' button, which is highlighted with a red rectangle.

Figure 13: Type in the file name to the text field that is circled.

Let's say you want the file you save to be called "*results*". You would type into the text field:

Save as :

- Press the save button to save the input and output signals with the file name given.
- The signals will be saved in a ".mat" file in the directory that the GUI and other files are located in (a variable will also appear in the Matlab Workspace).
- The signals themselves are saved as a struct. You can explore the struct by double clicking on the variable in the Matlab Workspace. To access the arrays in the struct you write:
 - variable_name.input.time*
 - variable_name.input.signal*
 - variable_name.output.time*
 - variable_name.output.signal*
- Now you can analyze the data that has been generated.