v1.2

A brief introduction to optimisation

Dr. Mudassir Lone
Dynamics, Simulation & Control Group
Centre for Aeronautics
School of Aerospace, Transport and Manufacturing
Cranfield University

March 6, 2017

Contents

1 Introduction

Today's engineering problems and the systems being designed as potential solutions are highly complex, nonlinear in nature and consist of many subsystems that require knowledge of multiple engineering disciples. The modern approach to design such systems relies on what is known as multi-disciplinary optimisation (MDO). This couples advances made in mathematical modelling of complex systems with the availability of computational power. However, design processes that utilise computational fluid dynamics (CFD) and/or computational structural mechanics (CSM) can need large amounts of computational resources; both in terms of integration of modelling codes through careful parameterisation and the execution of an automated optimisation process. Consequently the desire to reduce the computational cost of MDO processes and thus, allow the exploration of an increased number of design parameters has also led to the development of techniques where the dynamics of complex models can be captured either through the development of neural networks or surrogate models.

The fidelity of MDO processes vary depending on the design stage being considered. Low fidelity models are used for purposes such as overall aircraft design where the optimisation process may need to consider not only aircraft performance but also fleet level operations. High fidelity MDO has primarily focused on the detailed design of components such as high lift devices using CFD for aerodynamic prediction and finite-element methods for assessing structural performance. Today, significant work is being carried out to link low fidelity and high fidelity tools to develop a multi-fidelity MDO processes.

Optimisation methods have not only been applied in engineering design. Work done in fields such as optimal control, optimal trajectory generation and optimal sensing clearly demonstrate the use of optimisation techniques for control, guidance and prediction applied to dynamic systems. This report aims to provide early researchers a basic introduction to optimisation using MATLAB/Simulink. The first section provides a basic introduction to optimisation processes and is followed by a brief discussion on the choice of cost functions and the quality of optimal solutions. The report provides the reader with three examples, that demonstrate the implementation of optimisation methods in MATLAB/Simulink for engineering design and parameter estimation problems.

2 Optimisation process

Any optimisation process requires a clear definition of the problem. This is typically done by writing down a mathematical problem statement that comprises of the following:

- the system being considered,
- the parameters being varied,
- the cost function being minimised and,
- any parameter constraints.

A typical multiobjective optimisation problem statement would take the following form[1]:

$$\min_{p \in U} \{\mathbf{W} J_i(p), \quad \text{for} \quad i = 1, 2, ...n\} \tag{1}$$

subject to:

$$f_i(p) \leq \varepsilon_i \qquad \text{for} \quad i = 1, 2, ...n \tag{2}$$

Here, p represents the variable parameters in the universe U and J is the set of scalar cost functions that correspond to a particular objectives. This problem involves n objectives, each of which is constrained through the function f. Therefore, the final optimal solution must provide a minima of the cost function whilst keeping the function f lower than or equal to ε. $\mathbf{W}$ is the weighting matrix that can be used to prioritise the objectives.

The solution to the above problem can be a local or a global minima. The latter can be defined as follows:

$$\forall p \in \mathcal{R}^n : f(p^*) \leq f(p) \tag{3}$$

3 Defining a cost function

The cost function basically defines the topology over which the optimisation algorithm has to travel and search for the optimal solution. Therefore, its definition will significantly impact the computational cost required to find the optimal parameters or provide insight for possible trade-offs. The desirable characteristics of a cost function are the following[2]:

- scalar,
- clearly defined (preferably unique) maximum or minimum (local or global) and,
- preferably positive-definite.

Functions that are convex in nature can help guarantee the existence of a unique global minima. However, a discussion on convex functions is beyond the scope of this report and the reader is referred to Reference [3].

Moreover, a simple cost function that is differentiable can significantly assist in speeding up the optimisation process. For simple models, if the cost function can be differentiated an optimal solution can be found analytically. This is the case for the ordinary linear least squares regression problem used in areas such as aircraft system identification[4], where the cost function is defined as follows:

$$J(p) = \frac{1}{2}(z-y)^T(z-y) \tag{4}$$

where z and y are the measurement and model output vectors respectively. Now assuming a linear model such as $y = \mathbf{X}p$ and a similar measurement model, $z = \mathbf{X}p + \nu$ where ν is effectively error, the cost function can be written as:

$$J(p) = \frac{1}{2}(z-\mathbf{X}p)^T(z-\mathbf{X}p) \tag{5}$$

The optimisation process therefore needs to find parameters p such that the difference between the model output and the measurements is minimised. Now $J(p)$ can differentiated analytically and set to zero to identify the parameter vector $\hat{p}$ such that:

$$\min_{p\in\mathcal{R}}\{J(p)\} \tag{6}$$

subject to the model and measurement equations. In this case the solution can be found to be:

$$\hat{p} = \mathbf{X}^+ z \tag{7}$$

Consequently, the optimisation process only requires the calculation of the Moore-Penrose pseudo-inverse defined as:

$$\mathbf{X}^+ = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T \tag{8}$$

4 Quality of optimal solution

The problem of finding the parameters that minimise the cost function is the same as finding the minima of any particular function such as $y = f(x)$, albeit in multiple dimensions. Using $y = f(x)$ as an example, the value of x that gives the minima must satisfy the following conditions[1]:

- the gradient at the minima must be zero and,
- the second differential of y must be positive.

For example, consider the following function:

$$y = x^3 - 7x^2 + 2x + 50 \qquad \text{for} \quad -1 \leq x \leq 7 \tag{9}$$

Figure 1 shows how the function and its derivatives vary with respect to the independent parameter x. A minima as well as an inflexion point exist for this function at $x = 4.5$ and $x = 0.1$ respectively. These two points are characterised by the fact that $dy/dx = 0$, that is the crossing points for the first differential of

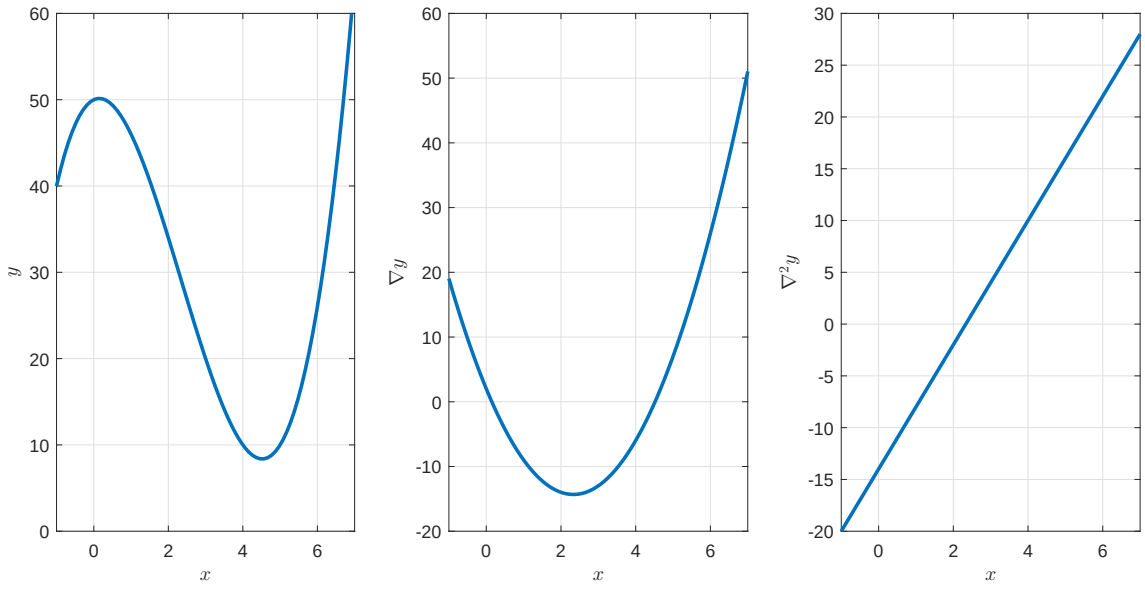

Figure 1: Properties of $y = x^3 - 7x^2 + 2x + 50$

y. However, the distinction between a minima and an inflexion point can be made only by considering the second differential which is positive for the minima and negative for the inflexion point.

The concept applied to the above example can now be generalised to a function of multiple variables. The properties of the first and second differential of such a function again determine the necessary conditions for a minima. In this case the first differential, also known as the Jacobian matrix is defined as follows:

$$\mathbb{J} = \nabla J(p) = \frac{\partial J(p)}{\partial p_i} \tag{10}$$

whilst the second differential, known as the Hessian is defined as:

$$\mathbb{H} = \nabla^2 J(p) = \frac{\partial^2 J(p)}{\partial p_i \partial p_j} \tag{11}$$

Given the above definitions, the necessary conditions for a minima p^* can be stated as:

- the Jacobian, $\mathbb{J}$ is zero, i.e. $\nabla J(p^*) = 0$.
- the Hessian matrix, $\mathbb{H}$ is positive semi-definite, i.e. :

$$\forall y \in \mathcal{R}^n : \qquad y^T \nabla^2 J(p^*) y \geq 0 \tag{12}$$

It should be noted that for a given cost function, the above conditions can be satisfied by a number of different combinations of parameter values. According to these conditions, each combination would be a valid minima and can be called a local minima. Therefore, these conditions are said to be sufficient for *local* optimality. The ideal solution to an optimisation problem is a *global* minima, which is a combination of parameters that lead to the minimum cost over the search domain of interest. However, searching the entire parameter space is often not feasible or cost effective. Consequently a number of algorithms exist that search for a minima given an initial starting set of parameters. The reader is referred to References [1], [2] and [3] for further details of such algorithms.

When carrying out multi-objective optimisation, often the objectives can be in conflict with each other and the optimisation problem ends up becoming a trade-off study. It also implies that the problem does not have a single solution that minimises the cost function and satisfies all constraints. Therefore, the optimisation process is in fact not a search for a single solution, but a search for a set of efficient solutions. An efficient or non-dominated solution, is a parameter set that provides the optimal trade-off between objectives, while at

the same time being away from any other feasible solution. Such solutions are also known as *pareto-optimal* solutions and a set of efficient solutions to a multi-objective problem define a *pareto-optimal front.*

The nature of engineering problems such as high level aircraft design or detailed component design, can be solved using multi-objective optimisation. Typically this will provide the designer with a set of efficient solutions that form pareto-optimal fronts in the objective space. However, often the solution itself is not of interest. Instead the value of conducting such a study is in understanding the dynamics between the set of conflicting objectives, the types of trade-off required to generate a feasible solution and finally, the physics governing this trade-off. Once these are understood the engineer can explore the role of other parameters and/or improve the design further.

At this point it is important to not forget the role of the Hessian matrix in assessing the quality of an optimal result. It is an indicator of the curvature of the cost function and, its variation to changes in parameter values provides insight into the sensitivity of the obtained minima. Small values in the Hessian matrix imply that the cost function is insensitive to changes in parameters and therefore, deviations from the optimal do not impact the cost function significantly. Therefore, it can be said that the obtained parameter set is not of high quality. Similarly, large values in the Hessian matrix imply strong sensitivity to variations in parameter values and thus, the obtained parameter can be said to be of high quality. This is especially the case in specific parameter estimation problems where the Hessian matrix can be directly linked to the Fisher information matrix and is used to calculate parameter confidence intervals.[5][6]

5 Examples

5.1 Example 1 : Linear mass-spring-damper

The basic application of an optimisation process is applied here to a simple mass-spring-damper system that has the following governing equations of motion:

$$m\frac{d^2x}{dt^2} + \lambda\frac{dx}{dt} + kx = u(t) \tag{13}$$

where the input is defined as $u(t) \in \mathbb{N}(0, \sigma^2)$, which represents random zero-mean Gaussian excitation with a standard deviation of σ . This is implemented in state-space form as a MATLAB function where stiffness, k is set to 1. It is desired to minimise the variance in the system's response to random noise by selecting a damping and mass for the system. The constraints are $0 \leq \lambda \leq 1$ and $0.1 \leq m \leq 1$ and variations in these parameters adds to the cost of the system. Therefore, a suitable cost function to minimise is as follows:

$$J = E[\dot{x} - \hat{\dot{x}}] + E[x - \hat{x}] + g(\lambda) + g(m) \tag{14}$$

where $\hat{x}$ and $\hat{\dot{x}}$ are the mean position and velocity respectively, and the cost of varying mass and damping takes the following form (illustrated in Figure 2):

$$g(y) = y^2 + e^{1-0.1y} \tag{15}$$

The analysis carried out here consists of two steps. The first step is a computationally expensive approach and aims to map the cost function topology and locate the minima. The second step effectively demonstrates the use of the constrained optimisation function **fmincon** in MATLAB to find the minima given an initial parameter guess.

The contour plot in Figure 2 is obtained by systematically varying parameters λ and m and for each combination running a linear simulation of the system model. At the end of each simulation the cost function value is calculated. It is evident that a single minima exists within the parameter space of interest and hence, the problem can be said to have a global minima. This occurs at $m = 0.26$ and $\lambda = 0.24$.

Once the cost function topology has been mapped like this, parameters that yield the minima can be identified via simple indexing of the data set generated in the process. However, this process is expensive and the cost function topology was obtained after almost 2000 simulations. A more efficient approach is to carry out the mapping not over the entire parameter space, but only near an initial guess and iterate based on gradient information. This can be achieved using the **fmincon** function in MATLAB. This function depends on the

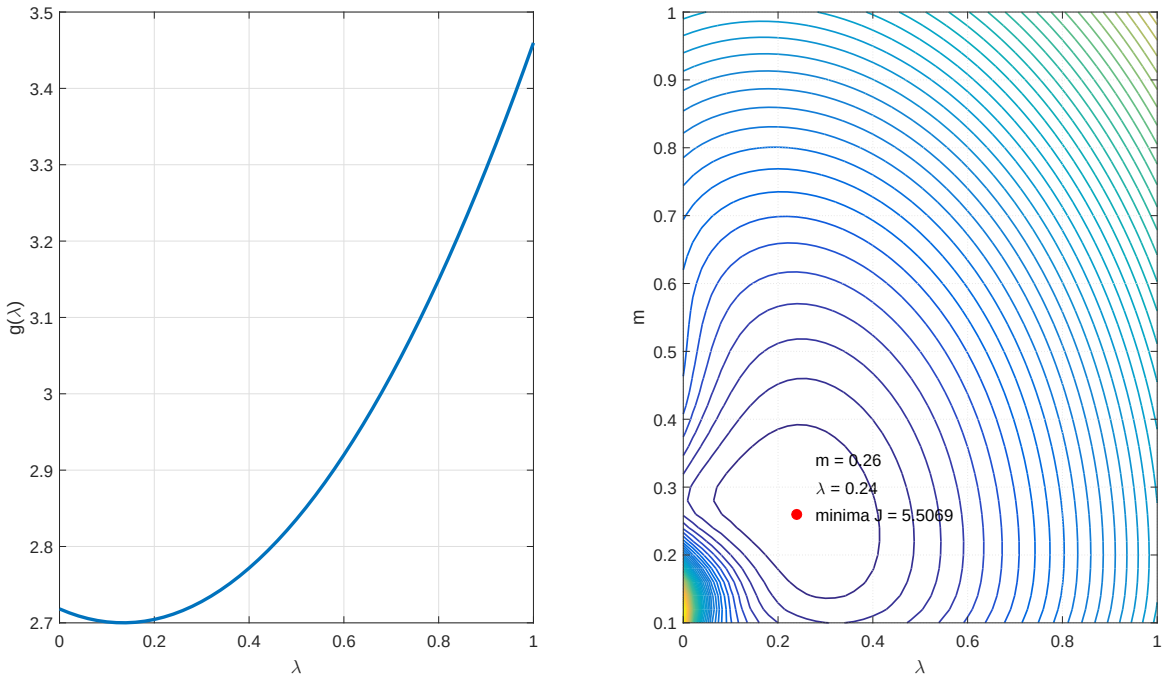

Figure 2: Penalty function for m and λ (left) and cost function topology (right)

user supplying it with (1) a MATLAB function that calculates the cost (that is where the cost function is defined) and, (2) a model of the for which parameters can be varied that is accessible to the optimisation algorithm.

For this example, the optimisation process is initiated in a script called *fmincon_approach.m*. Within this script, the optimiser settings are defined and the optimiser is initialised with arbitrarily chosen parameter values. The **fmincon** function is then supplied with a function handle for the cost function (named as *cost_fun.m*, within which the model outputs as a result of changes in m and λ are obtained and the cost function is evaluated for each instance. Note that the actual model is programmed as a function called *mass_spring_damper.m* that has the main parameters as input variables and it outputs the response (x and $\dot{x}$) together with the respective variances. The MATLAB codes used for this example can be found in Appendix **??**. The results of this process are shown below:

```
                                          First-order      Norm of
 Iter F-count            f(x)  Feasibility   optimality         step
    0       3    6.001029e+00    0.000e+00    1.325e+00
    1       7    5.623401e+00    0.000e+00    7.244e-01    4.186e-01
    2      11    5.507757e+00    0.000e+00    6.647e-02    2.967e-01
    3      15    5.511511e+00    0.000e+00    1.128e-01    5.490e-02
    4      18    5.508876e+00    0.000e+00    1.001e-01    2.423e-02
    5      21    5.507160e+00    0.000e+00    2.184e-02    1.928e-02
    6      24    5.506821e+00    0.000e+00    8.206e-03    1.216e-02
    :       :               :            :            :            :
   10      36    5.506815e+00    0.000e+00    4.008e-07    1.660e-05

Optimization completed: The relative first-order optimality measure, 4.008292e-07,
is less than options.OptimalityTolerance = 1.000000e-06, and the relative maximum constraint
violation, 0.000000e+00, is less than options.ConstraintTolerance = 1.000000e-06.

Optimization Metric                                      Options
relative first-order optimality =   4.01e-07       OptimalityTolerance =   1e-06 (selected)
relative max(constraint violation) =   0.00e+00    ConstraintTolerance =   1e-06 (selected)
```

The initial parameter guess provided to the optimiser was 0.5 and 0.8 for the mass and damping coefficient

respectively. Options for **fmincon** can be set such that a number of optimisation parameters are shown in the command window as the process is executed. The number of iterations and function counts are the first two columns of the output. This is followed by the cost function value at the end of each iteration. The feasibility of the parameters and the first order optimality are the next two columns. The latter as defined in MATLAB represents the first necessary condition for a minimum, $\nabla J = 0$. The final column is the step size taken at each iteration. The optimal parameters obtained were $m = 0.2654$ and $\lambda = 0.2392$. The results show that the optimal parameters were obtained after 36 evaluations, and in fact the optimisation algorithm almost found the minima after only 10 evaluations. In this case, an algorithm known as interior-point was selected to carry out the search. The Jacobian and Hessian matrices for the optimal solution were found to be:

$$\mathbb{J} = \begin{bmatrix} -0.5960 \times 10^{-6} & 0 \\ 0 & -0.6557 \times 10^{-6} \end{bmatrix} \quad \text{and} \quad \mathbb{H} = \begin{bmatrix} 6.2344 & 1.4093 \\ 1.4093 & 2.7572 \end{bmatrix} \tag{16}$$

The Jacobian is practically zero and the Hessian is clearly positive definite and therefore, both conditions required for a minima have been satisfied.

5.2 Example 2 : Nonlinear mass-spring-damper

The objective of this example is to demonstrate the integration of MATLAB's optimisation tools with nonlinear models programmed in Simulink. In this case the mass-spring damper model is modified such that the governing equations in state-space form are as follows:

$$\dot{x} = \mathbf{A}x + \mathbf{B}u + \mathbf{E}x^2 \tag{17}$$

$$y = \mathbf{C}x + \mathbf{D}u \tag{18}$$

The above equation can be modelled in Simulink quite simply using the following matrices:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -k/m & -\lambda/m \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 \\ 1/m \end{bmatrix} \quad \mathbf{E} = \begin{bmatrix} 1 & \lambda \\ m & -1 \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \mathbf{D} = 0 \tag{19}$$

The stiffness parameter k is again kept constant at a value of 1. For this example, the cost function has been defined as follows:

$$J = \frac{1}{N}\sum_{i}^{N}(x_i - u_i)^2 + \frac{1}{m} + \lambda \tag{20}$$

where N is the total number of samples obtained from a 10 second simulation. The first term is effectively a representation of error whereas the second and third terms are penalties for variations in mass and damping coefficient. Since $0.1 \leq m \leq 1$ and $0 \leq \lambda \leq 1$, this formulation means that the addition of mass is more expensive than increase in damping.

The same two step approach as that used for Example 1 is implemented here. First the cost function topology is obtained by evaluating all combinations of parameters (see Figure 3). Again, this required over 2000 simulations. The minima can also be identified via simple indexing of the data set generated in the process. In this example, the minima is achieved by setting $m = 0.54$ and $\lambda = 0.2$.

Figure 3 clearly shows the existence of a global minima and the topologies for the corresponding Jacobians also point to a unique optimal solution.

Implementing the connection between the Simulink model and the **fmincon** function requires the programmer to understand that the optimiser must be able to exchange values for the design variables between the MATLAB workspace (the script) and the cost function space (the function). This implies that Simulink can no longer take variables from the workspace, but instead needs to operate in the function space. The exchange of variables between MATLAB workspace and function space is simple and straightforward. However, the execution of Simulink models in function space is not done so regularly. This is achieved by applying the following setting:

```
opt = simset('SrcWorkspace','current');
```

and passing the options variable to the command for simulation as follows:

```
[T,x,Y] = sim(model,10,opt,[t; u]');
```

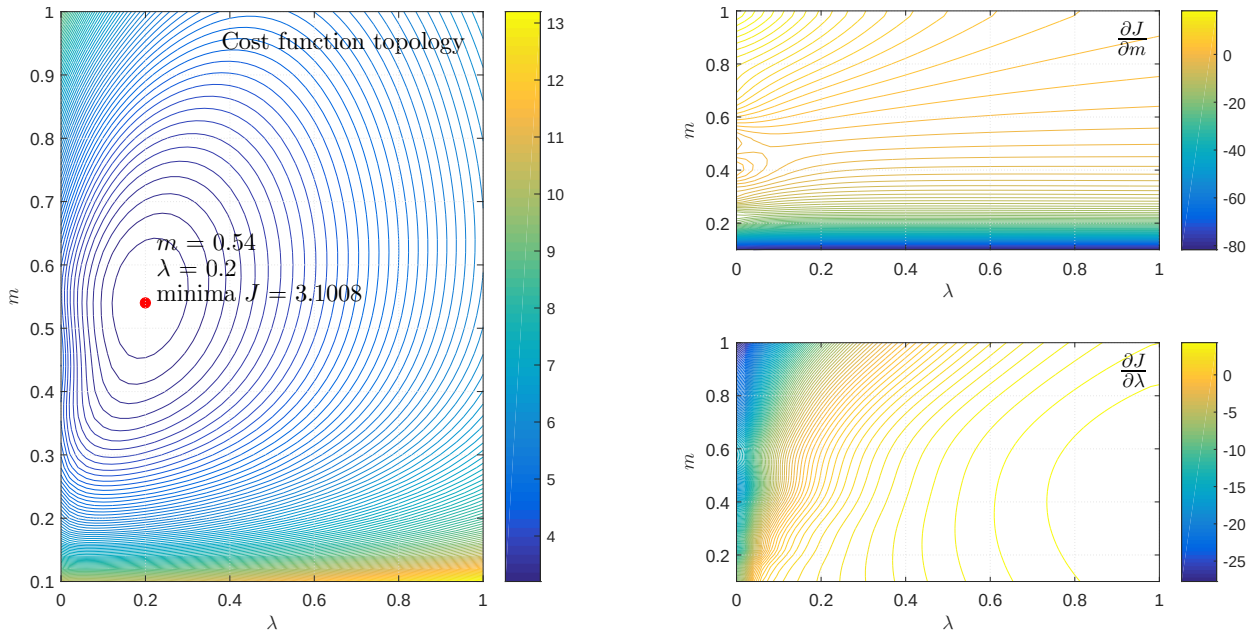

Figure 3: Cost function (left) and Jacobian (right) topologies

Note that the above is the backwards compatible code for the execution of Simulink models from command line. The scripts and functions used in this Example can be found in Appendix **??**. The optimisation results obtained using the **fmincon** function for this example are as follows:

Iter	F-count	f(x)	Feasibility	First-order optimality	Norm of step
0	3	5.083846e+00	0.000e+00	3.823e+00	
1	6	4.268814e+00	0.000e+00	1.823e+01	8.910e-01
2	11	4.071622e+00	0.000e+00	6.843e+00	1.052e-01
3	15	3.250316e+00	0.000e+00	1.719e+00	4.264e-01
4	18	3.127332e+00	0.000e+00	5.663e-01	7.900e-02
5	21	3.103495e+00	0.000e+00	2.058e-01	4.939e-02
6	24	3.103822e+00	0.000e+00	1.030e-01	7.413e-03
7	27	3.103518e+00	0.000e+00	1.000e-01	8.440e-04
8	30	3.100006e+00	0.000e+00	2.444e-02	1.310e-02
9	33	3.099783e+00	0.000e+00	7.356e-03	4.426e-03
10	36	3.099783e+00	0.000e+00	2.000e-04	1.830e-04
:	:	:	:	:	:
13	45	3.099783e+00	0.000e+00	2.305e-07	2.944e-07

```
Optimization completed: The relative first-order optimality measure, 2.304966e-07,
is less than options.OptimalityTolerance = 1.000000e-06, and the relative maximum constraint
violation, 0.000000e+00, is less than options.ConstraintTolerance = 1.000000e-06.

Optimization Metric                                   Options
relative first-order optimality =   2.30e-07      OptimalityTolerance =   1e-06 (selected)
relative max(constraint violation) =   0.00e+00   ConstraintTolerance =   1e-06 (selected)
```

The initial guess for this process was 0.8 for both λ and m, and the final parameters output by **fmincon** were 0.5490 and 0.1997 for mass and damping coefficient respectively. It should be noted that the optimal parameters are approached within 4 iterations that constitute only 18 simulations. The Jacobian and Hessian matrices output at these parameters were as follows:

$$\mathbb{J} = \begin{bmatrix} -0.2384 \times 10^{-6} & 0 \\ 0 & -0.0894 \times 10^{-6} \end{bmatrix} \quad \text{and} \quad \mathbb{H} = \begin{bmatrix} 23.8639 & -3.4403 \\ -3.4403 & 26.3993 \end{bmatrix} \tag{21}$$

The Jacobian in this case is practically zero and the Hessian is positive definite since $y^T \mathbb{H} y > 0$ for every non-zero element of y.

Care should be taken when interpreting the output of **fmincon**. The Hessian output is produced via an algorithm that generates an approximation of the real Hessian and the algorithm has numerous limitations. Therefore, it is recommended that when possible, include the calculation of the analytical Hessian and if this is not possible, at least check the eigenvalues and condition number, γ of the Hessian matrix. The condition number can be obtained as follows:

$$\gamma = \log_{10}\left(\frac{\xi_{\max}}{\xi_{\min}}\right) \tag{22}$$

where $\xi_{\max}$ and ξ_{min} are the largest and smallest eigenvalues of $\mathbb{H}$. Large condition numbers imply problems with the final result. Negative or near zero eigenvalues also indicate problems. In this case the eigenvalues and condition number were found to be:

$$\xi = \begin{bmatrix} 21.4652 \\ 28.7980 \end{bmatrix} \qquad \text{and} \qquad \gamma = 0.1276 \tag{23}$$

Since the cost function topology was mapped first and it was established that a clear global minima existed and could be found, the satisfactory values for ξ and γ are not surprising.

5.3 Example 3 : Parameter estimation for pilot models

Parameter estimation is another area where optimisation processes are used extensively. This example demonstrates the use of the MATLAB **fmincon** function to quantify human control behaviour by estimating parameters for a mathematical pilot model. In this example, the human subject is aiming to minimise the pitch attitude deviations of the aircraft due to random turbulence. It is assumed that the stick deflection, δ is related to the observed error e via the following relationship:

$$\delta(t) = Y_p(t)e(t) + \nu \qquad \text{where} \quad \nu \in \mathbb{N}(0, \sigma) \tag{24}$$

where ν captures any nonlinear or random behaviour. At this stage it is important to note that ν and the turbulence itself are effectively zero mean Gaussian stochastic processes. The model postulated for the pilot is the following transfer function:

$$Y_p(s) = K_p \frac{\tau_I s + 1}{\tau_L s + 1} e^{-\tau s} \tag{25}$$

which includes a delay term that makes the model nonlinear. This requires the identification of four parameters from the measured error (pitch attitude deviation) and stick deflections. The overall experimental and modelling setup are the same and illustrated in Figure 4. The reader is referred to Reference [7] for a detailed discussion on the experimental method.

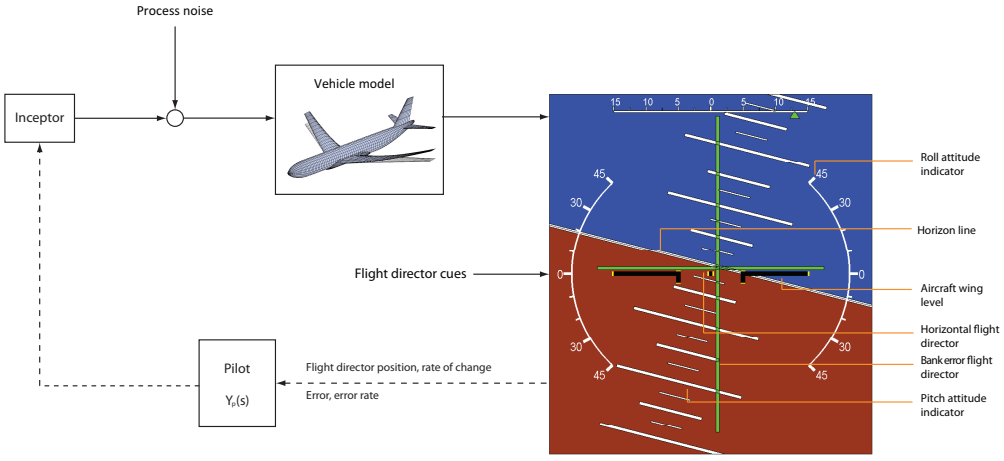

Figure 4: Experimental and modelling setup to identify manual control dynamics

Majority of parameter estimation algorithms in aerospace engineering are based on the Maximum Likehood Estimation (MLE) approach. The reader is referred to References [8] and [5] for a detailed discussion on MLE methods, but the cost function (assuming no state noise) is as follows:

$$J(\theta) = \sum_{i=1}^{N} [z(i) - y(i)]\mathbf{R}^{-1}[z(i) - y(i)]^T + \frac{1}{N} \ln \mathbf{R} \tag{26}$$

where z and y are the measurements and model predictions respectively. The matrix $\mathbf{R}$ is the measurement noise covariance and in this case assuming $\mathbf{R} = \mathbf{I}$ simplifies the cost function to:

$$J(\theta) = \sum_{i=1}^{N} [z(i) - y(i)][z(i) - y(i)]^T \tag{27}$$

For this example, the pilot model and the cost function can be programmed as MATLAB functions and so the optimisation process that minimises J does not rely on a Simulink model. Now **fmincon** can be used with the following a priori known limits:

$$-1 \leq K_p \leq -0.01 \qquad 0 \leq \tau_I \leq 5 \qquad 0 \leq \tau_L \leq 5 \qquad 0.2 \leq \tau \leq 0.5 \tag{28}$$

and the initial guess $\theta_0 = \begin{bmatrix} K_p & \tau_I & \tau_L & \tau \end{bmatrix} = \begin{bmatrix} -0.1 & 2 & 4 & 0.3 \end{bmatrix}$. The **fmincon** optimiser has been programmed in a similar way to that shown for Examples 1 and 2, and the output for this example is as follows:

```
                                        First-order      Norm of
 Iter F-count            f(x)  Feasibility   optimality         step
    0       5    7.898953e+02    0.000e+00    6.912e+03
    1      13    5.408482e+02    0.000e+00    3.282e+03    2.429e-01
    2      20    5.118095e+02    0.000e+00    1.869e+03    5.615e-01
    3      26    5.089632e+02    0.000e+00    2.403e+03    7.170e-01
    4      31    5.023465e+02    0.000e+00    1.090e+03    6.384e+00
    5      36    4.930608e+02    0.000e+00    6.731e+02    6.405e-02
    6      41    4.780949e+02    0.000e+00    2.323e+02    1.821e-01
    7      46    4.660711e+02    0.000e+00    2.100e+02    6.424e+00
    8      52    4.635074e+02    0.000e+00    2.047e+02    9.647e-01
    9      57    4.630863e+02    0.000e+00    1.165e+02    1.056e+00
   10      62    4.626005e+02    0.000e+00    1.219e+01    7.183e-01
   11      67    4.625509e+02    0.000e+00    6.848e+00    1.773e-02
   12      72    4.625335e+02    0.000e+00    1.238e+01    4.484e-02
   13      77    4.625323e+02    0.000e+00    6.790e+00    2.409e-02
   14      82    4.625321e+02    0.000e+00    9.570e-01    1.473e-02
    :       :               :           :            :            :
   22     122    4.625320e+02    0.000e+00    2.000e-04    2.803e-05
   23     127    4.625320e+02    0.000e+00    2.087e-04    1.906e-05
   24     132    4.625320e+02    0.000e+00    6.158e-05    2.063e-06
   25     138    4.625320e+02    0.000e+00    1.912e-04    8.805e-07
   26     150    4.625320e+02    0.000e+00    1.683e-04    2.215e-07
   27     167    4.625320e+02    0.000e+00    1.980e-04    6.831e-10

Optimization stopped because the relative changes in all elements of x are
less than options.StepTolerance = 1.000000e-10, and the relative maximum constraint
violation, 0.000000e+00, is less than options.ConstraintTolerance = 1.000000e-06.

Optimization Metric                                      Options
max(abs(delta_x./x)) =   6.98e-11                  StepTolerance =   1e-10 (default)
relative max(constraint violation) =   0.00e+00   ConstraintTolerance =   1e-06 (selected)
```

The parameter estimates obtained from this process are shown in Figure 5. The optimisation process required 27 iterations consisting of 167 simulations.

Note that in this case the cost function topology is five dimensional (cost and the four parameters) and there is little benefit in attempting to visualise it. However, if parameters such as τ_I and τ_L are fixed, the cost function can be mapped and explored in terms of variations in K_p and τ. The final parameter set obtained in this example was as follows:

$$\hat{\theta} = \begin{bmatrix} -0.1935 & 2.5456 & 4.9495 & 0.3718 \end{bmatrix} \tag{29}$$

and the corresponding Jacobian matrix was computed to be as follows:

$$\mathbb{J} = \begin{bmatrix} -0.0012 & 0 & 0 & 0 \\ 0 & 0.0001 & 0 & 0 \\ 0 & 0 & 0.0001 & 0 \\ 0 & 0 & 0 & -0.0001 \end{bmatrix} \tag{30}$$

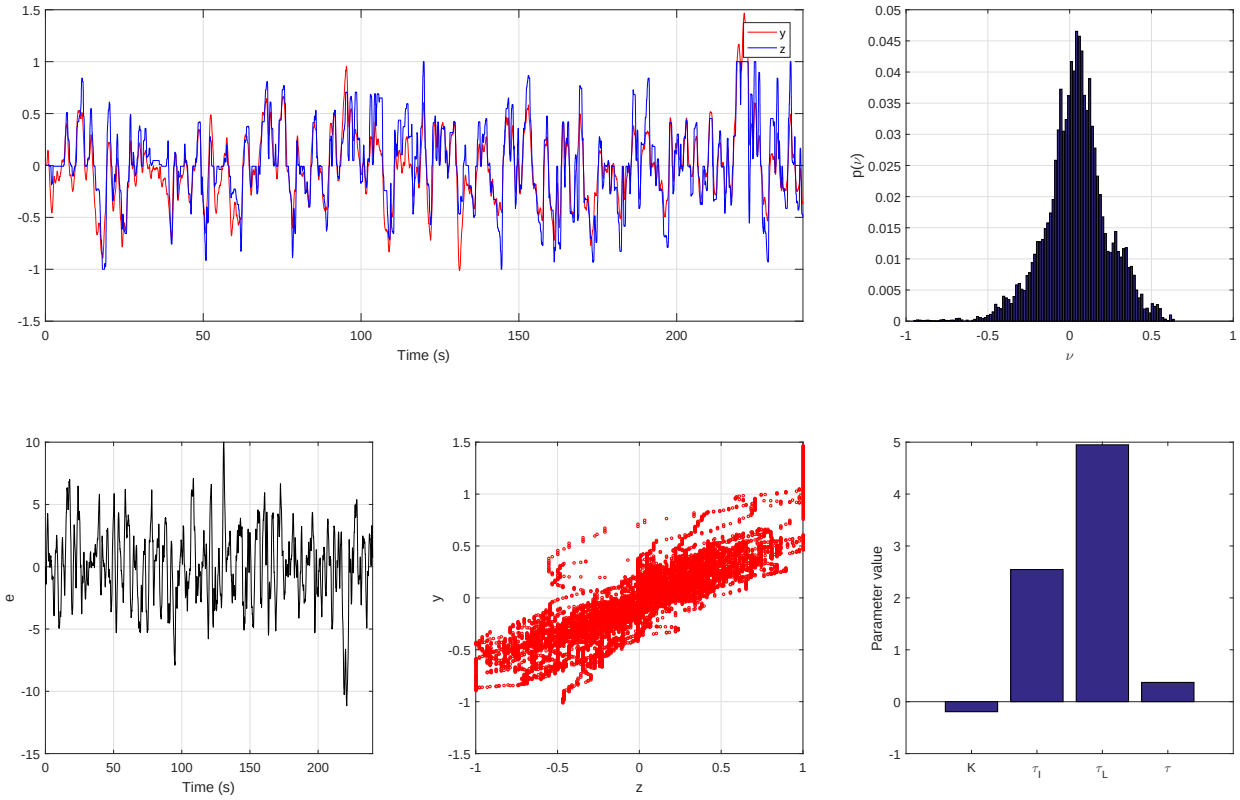

Figure 5: Results of applying **fmincon** to pilot model parameter estimation problem

Once the parameters have been estimated, the key question that remains is: what are the corresponding confidence intervals? The process of obtaining these is simplified by the fact that the MLE approach is adopted here. This means that the standard error for each parameter is a function of the Hessian matrix as follows:

$$\sigma = \sqrt{\text{diag}(\mathbb{H}^{-1})} \geq \text{CRB} \tag{31}$$

where σ is the vector of standard errors for each parameter and CRB is the matrix of Cramér-Rao bounds. The reader is referred to References [4], [6] and [5] for the derivation of the inequality linking the Hessian to the CRB and explanations for interpreting standard errors in aircraft system identification. Here, the Hessian matrix was found to be as follows:

$$\mathbb{H} = \begin{bmatrix} 3.8117 & 1.1718 & -0.4353 & 1.9878 \\ 1.1718 & 0.3603 & -0.1338 & 0.6111 \\ -0.4353 & -0.1338 & 0.0497 & -0.2268 \\ 1.9878 & 0.6111 & -0.2268 & 1.0436 \end{bmatrix} \times 10^6 \tag{32}$$

The 95% confidence intervals are therefore obtained as follows:

$$\theta = \hat{\theta} \pm 1.96\sqrt{\frac{\sigma^2}{N}} \qquad \text{where} \quad \hat{\theta} \in \mathbb{N}(\theta, \sigma) \tag{33}$$

where $\hat{\theta}$ is the parameter estimate vector obtained from **fmincon** and N is the number of samples in the measurement. In this case, the test was conducted for 3 minutes and the data sampling rate was 100Hz. As a result the total number of samples collected is very large ($N = 24001$) and consequently the confidence intervals around the estimated parameters are quite tight. The MATLAB functions used for this example can be found in Appendix **??**.

References

[1] G.P. Liu, J.B. Yang, and J.F. Whidborne. *Multiobjective optimisation and control*. Research Studies Press Ltd., 2001.

[2] S. Boyd and L. Vandenberghe. *Convex Optimisation*. Cambridge University Press, 2004.

[3] R. Stengel. *Optimal Control and Estimation*. Dover Publications, 1994.

[4] V. Klein and E. Morelli. *Aircraft system identification: Theory and practice*. American Institute of Aeronautics and Astronautics, 2006.

[5] K.W. Illif and R.E. Maine. More than you may want to know about maximum likelihood estimation. Technical Report NASA-TM-85905, National Aerionautics and Space Administration, 1985.

[6] M.B. Tischler and R.K. Remple. *Aircraft and rotorcraft system identification*. American Institute of Aeronautics and Astronautics, 2006.

[7] M.M. Lone and A.K. Cooke. Pilot-model-in-the-loop simulation environment to study large aircraft dynamics. *Proc IMechE Part G: J Aerospace Engineering*, (227(3) 555–568), 2012.

[8] Lennart Ljung. *System identification: Theory for the user*. Prentice Hall, 1999.