TSFS09 1B

October 31, 2025

1 Introduction

In project 2A, we selected our component models and investigated what measurement signals and parameters were needed to characterize them. In this project, we will use measured turbocharger data to find appropriate values for our unknown parameters.

In project 1B, we worked with models linear in the parameter vector x, that is, with some effort they could be written on the form

$$A\ x = b.$$

For these models, we found that x can be estimated using the Matlab backslash operator.

As you might have noticed when compiling your models for the turbocharger, they are not linearly dependent on x. Thus, we cannot shuffle them into our desired form, and cannot use our previous tool to solve this problem. We must learn to use another.

1.1 Non-linear Regression

Consider the model

$$y = (x - v)^2,$$

where y is our desired output, v some measured signal (perhaps velocity) and x a scalar parameter. Note that the model is non-linear (square) in x.

Like before, we have several measured values for y and v, and just like before,

we get one equation per measurement:

$$\begin{aligned} y_1 &= (x - v_1)^2 \\ y_2 &= (x - v_2)^2 \\ &\vdots \\ y_N &= (x - v_N)^2 \end{aligned}$$

The objective has not changed: we still want to find an x to minimize the least-squares criterion (difference between what our model predicts and the measured value)

$$J(x) = \sum_{k=1}^{N} [y_k - (x - v_k)^2]^2$$

Note that the first square is from our model, the second from the criterion.

At this point we could be clever and recognize that our model contains a square, which might lead us into quadratic programming. However, for general non-linear models there are no guarantees of success. Most, if not all, minimize J by iteration: we assume an initial guess for x which we call x_0, and then try to find another value x_1 for which $J(x_1) < J(x_0)$ holds. When we are satisfied with the smallness of J, or if J stops shrinking, we stop iterating.

Most methods of finding x_1 from x_0 use information about the gradient (slope) of J. Some also use second-order information (Hessians). These techniques are built into the solvers we will apply, so we won't delve any further.

1.2 `lsqnonlin()`

Matlab conveniently provides us with the command `lsqnonlin()`, which we should think of as an alternative to the backslash operator when our models are non-linear in the parameters.

Compared to using the backslash operator, non-linear regression is actually easier: we don't need to shuffle our models into the $Ax = b$ form. Using the model above as an example, we can simply run the code

```
x = lsqnonlin(J, x0),
```

and Matlab does the rest. Let's try it on our turbine flow example.

2 Turbine Flow Example

Here is the Matlab function for turbine flow (or rather, the turbine flow parameter (TFP)):

```
function TFP = turbineFlow(p03, p04, k0, k1)

% turbine pressure ration
Pi_t = p04./p03;

% Turbine Flow Parameter (TFP)
TFP = k0 .* sqrt(1 - Pi_t.^k1);

end
```

In project 1, we had to find the parameters first, and then create the function. Now, we create the function first, and use that function to find the parameters.

To parameterize k_0 and k_1, we begin by loading the turbocharger map data:

```
% load turbo data
load_turbo_map;
y  = TFP;
```

As discussed above, we then need to supply Matlab with an initial guess for x, which we call x_0:

```
% initial guess
x0 = [1;1];
```

The command `lsqnonlin` has multiple options. Here, we ask `lsqnonlin` to output status reports on every iteration.

```
% options
opts = optimoptions('lsqnonlin', 'Display', 'iter');
```

We then use a convenient tool available in Matlab, the anonymous function, also known as a function handle.

```
% LSQ
J = @(x) turbineFlow(p_03, p_04, x(1), x(2)) - y;
```

Note that `turbineFlow()` is a function that depends on four inputs. Our newly created function handle `J` depends only on x, that is, our parameter.

Now we are ready to find our parameters:

```
% run optimization
x = lsqnonlin(J, x0, [], [], opts);
```

In the fourth and fifth inputs to `lsqnonlin` we have the option to include upper and lower bounds on x. In this case, no bounds are necessary so we can submit

empty vectors instead.

We now have a vector x with $x_1 = k_0 = 0.0054$, and $x_2 = k_1 = 1.4468$. Plotting the model response against the turbocharger map data gives the results shown in Figure 1.

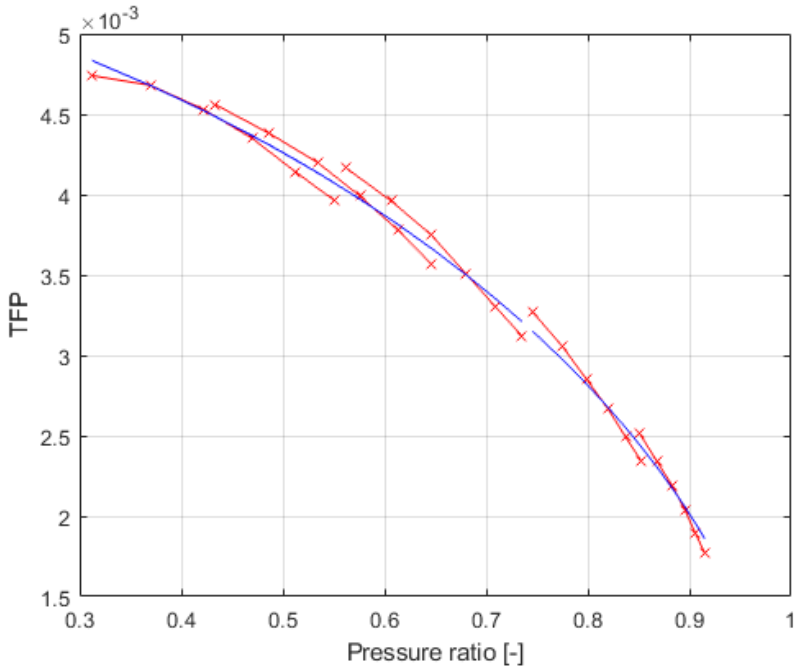

Figure 1: Model response (blue) versus map data (red).

- Complete the parameterization and validation of the remaining turbocharger models (turbine efficiency, compressor flow, and compressor efficiency). There is no parameterization or validation needed for the shaft dynamics.