MATLAB Interface for PowerBI

MathWorks, Inc.

Apr 10, 2024

OVERVIEW

CHAPTER

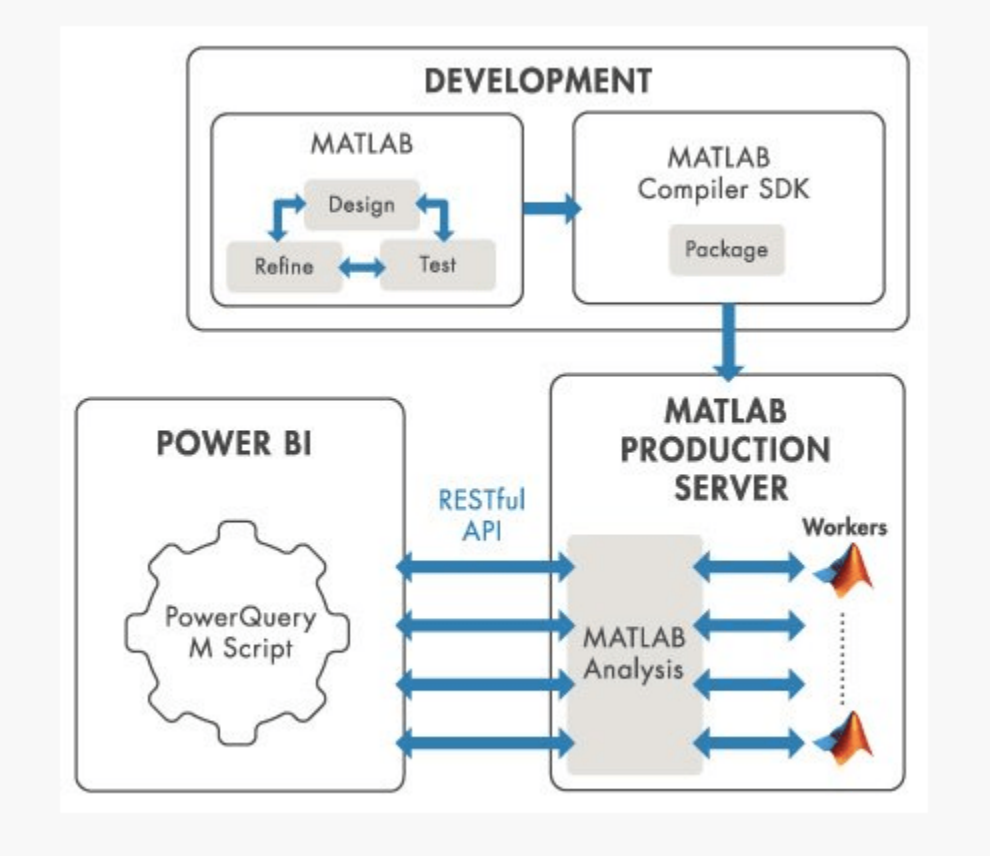
ONE

OVERVIEW

1.1 Introduction

This document provides an overview of the configuration and use of the MATLAB Production Server Interface for Power BI software. The interface is a lightweight Power BI custom data connector that enables the integration of MATLAB algorithms with Power BI visualizations. The custom data connector is authored in M Query language.

1.2 Architecture Diagram



The MATLAB Production Server Interface for Power BI enables Power BI users to directly access MATLAB algorithms using the custom data connector feature of Power BI (https://docs.microsoft.com/en-us/power-bi/connect-data/desktop-connector-extensibility).

The custom data connector communicates with MATLAB functions using a RESTful API. HTTP requests send data from tables in Power BI to MATLAB and receive the results from MATLAB computations in JSON format. There are options to drill down into the result set, apply transformations, and perform other modifications as required using the query editor in Power BI.

MATLAB algorithms can run either:

1. In a centralized scalable environment using MATLAB Production Server. This allows for multiple Power BI users to call the MATLAB algorithms concurrently, or
2. On a desktop machine with MATLAB Compiler SDK installed using the 'Test Client' feature. This allows the Power BI desktop application to make calls to MATLAB function on the same machine.

1.3 System Requirements

MathWorks Products

1. MATLAB (R2016b or later)
2. MATLAB Compiler SDK (R2016b or later)
3. MATLAB Production Server (R2016b or later) – *required if deploying MATLAB applications to an enterprise environment*

Microsoft Products

1. Microsoft Power BI Desktop (version 2.47 or later)

CHAPTER

TWO

INSTALLATION

2.1 Installation

The custom data connector is provided as `Software\PowerBI\MATLABProductionServer.mez`. This file will have to be added to the local custom connectors directory *and* as the provided connector is not officially validated nor signed, some additional configuration is needed.

Further, the package includes a few MATLAB functions which help with data marshalling between MATLAB and PowerBI (also see *MATLAB Code Interface*), these functions will have to be added to the MATLABPATH.

2.1.1 Enable the custom connector to load

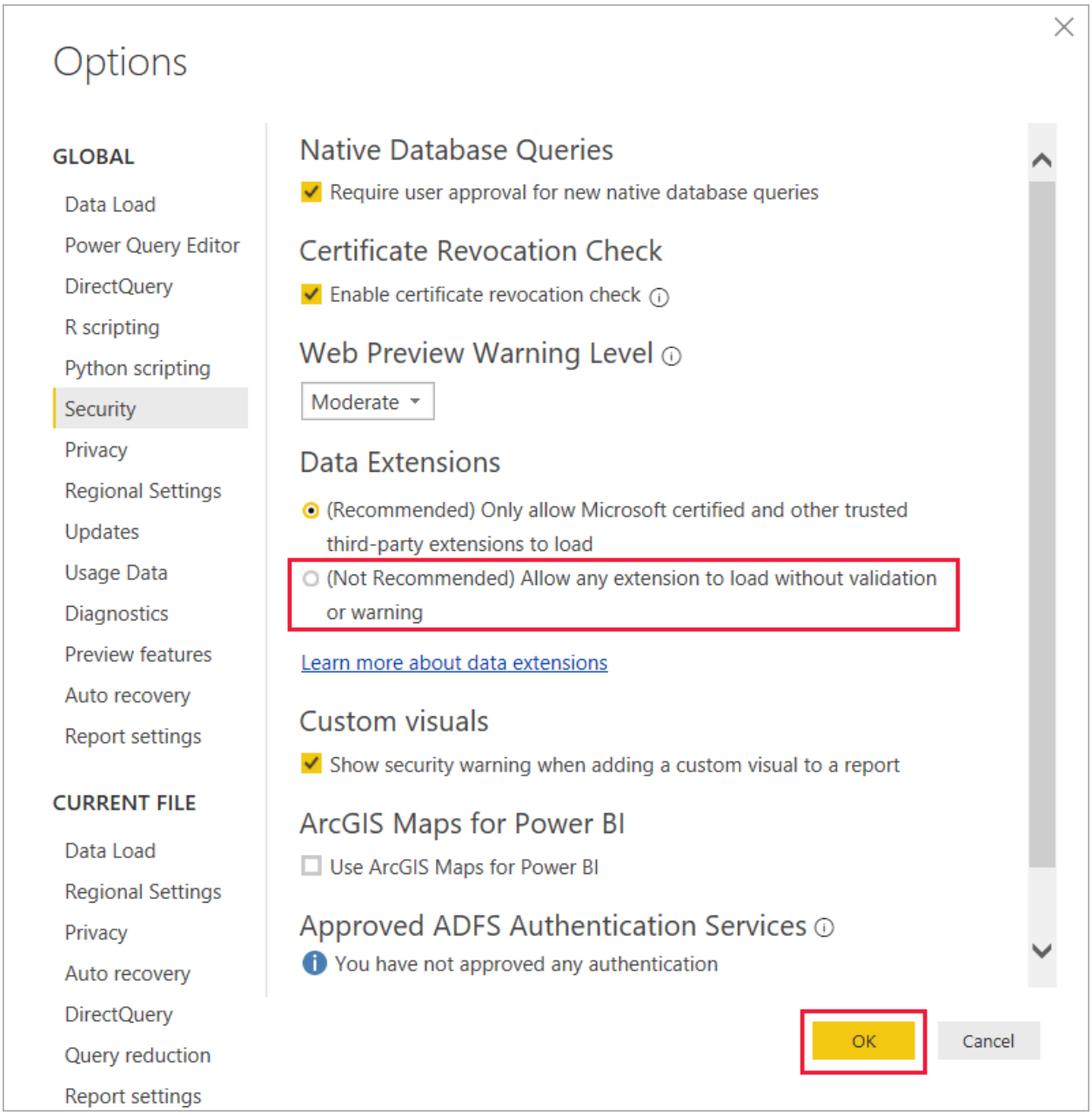
As discussed in Microsoft's "Connector extensibility in Power BI" documentation, in order to be able to use a custom connector, it either needs to be signed and trusted or Power BI must be configured to allow loading *any* extension. The custom connector as provided is *not* signed by MathWorks but it should be possible to first sign it by yourself and then trust your own signature. If this is the approach you want to follow see *Sign the custom connector*, alternatively see *Configure Power BI Security settings*.

Sign the custom connector

To learn more about repacking the provided MEZ-file as signed PQX-file and how to trust the signature see Handling Power Query Connector Signing in the Microsoft documentation.

Configure Power BI Security Settings

As an alternative to signing and trusting the custom connector, it is possible to configure Power BI to allow *any* extension to be used. To enable this option, in Power BI Desktop under `File` → `Options and settings` → `Options` → `Security` → `Data Extensions` enable the `(Not Recommended) Allow any extension to load without validation or warning` option:



2.1.2 Install the custom connector

To actually install the custom connector, copy the MEZ-file (or PQX-file if you or your company signed it) into the `Documents\Power BI Desktop\Custom Connectors` directory.

Note: This directory may not exist yet in which case it will first have to be created.

2.1.3 MATLAB Setup

In order to add the MATLAB helper functions to the MATLABPATH in MATLAB run `Software\MATLAB\startup.m`. The MATLABPATH can then be saved using `savepath` or `startup.m` can be run again in new MATLAB sessions to add the functions to the path again.

CHAPTER

THREE

USAGE

3.1 MATLAB Code Interface

In Power BI you mainly work with data in *Tables*, hence, when interacting between MATLAB and Power BI it also makes sense to write the MATLAB code in such a way that it can take one Power BI *Table* as input (which can then be turned into a MATLAB *Table*) and have it produce its outputs in MATLAB *Tables* (which can then be turned into Power BI *Tables*). As a matter of fact, this is the recommended approach when working with this package. Both the *"Data Source" approaches*, as well as the *data conversion helper function*, assume *and require* that your MATLAB code is indeed in this format.

The *invoke function approach* and *entirely manual approach* do not have this requirement but may require (significantly) more complex Power Queries in order to process in- and outputs.

In order to implement a MATLAB function which follows this "one (or no) input table, output tables" approach, write your MATLAB code according to the following template:

```
function [out1,out2] = myFunction(varargin)
    %% PowerBI Input/Output Handling
    inputTable = PowerBI.InputToTable(varargin);

    % Start new tables for the outputs
    outTable1 = table;
    outTable2 = table;

    %% Actual Algorithm
    % Write the actual algorithm to work with MATLAB tables

    % Just some simple example - replace with your actual algorithm
    outTable1.D = inputTable.A + inputTable.B;
    outTable1.E = upper(inputTable.C);

    outTable2.D = inputTable.A - inputTable.B;
    outTable1.E = lower(inputTable.C);

    %% PowerBI Input/Output Handling
    out1 = PowerBI.TableToOutput(outTable1);
    out2 = PowerBI.TableToOutput(outTable2);
```

Where `PowerBI.InputToTable` and `PowerBI.TableToOutput` are functions provided with this package.

For more background information on how this in- and output handling works, see *Data Marshalling*.

If your function does not require any input at all, a "no input, output tables" approach is supported as well:

```
function out = myFunction()
    % Start a new table for the output
    outTable = table;

    %% Actual Algorithm
    % Write the actual algorithm to work with MATLAB tables

    % Just some simple example - replace with your actual algorithm
    outTable.R = rand(10,1);

    %% PowerBI Input/Output Handling
    out = PowerBI.TableToOutput(outTable);
```

Hint: When encountering issues/receiving errors related to data marshalling it can be very helpful to make use of the Test Client feature of MATLAB Compiler SDK, and configure Power BI to make calls against the test instance. You can then set a breakpoint at the start of the function and literally see what data in what format the MATLAB function received. Or set a breakpoint right before the end of the function such that you can double check whether the outputs are in the correct format and orientation.

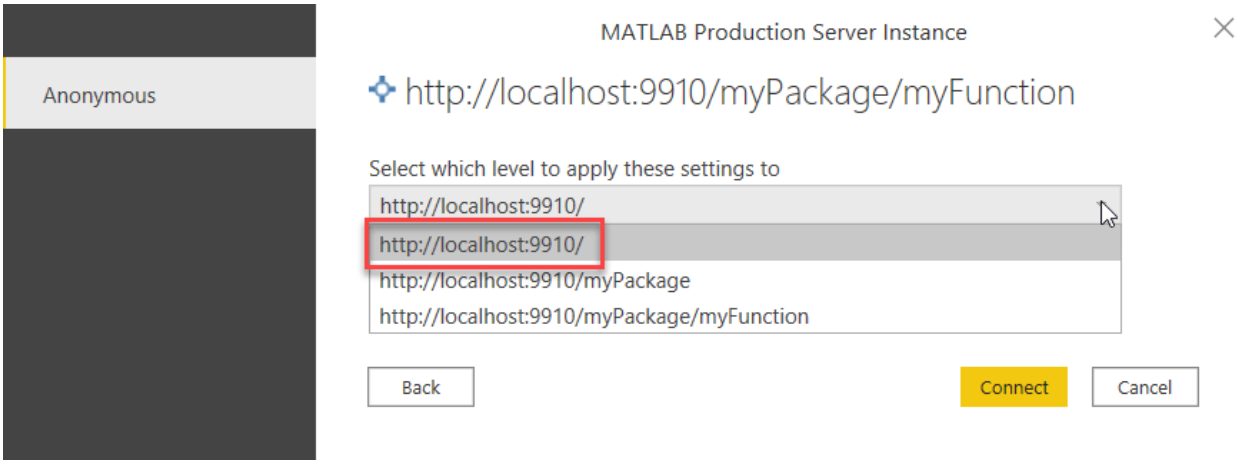
3.2 Usage in Power BI Desktop

The package offers three major workflows, one helper function and one alternative approach. Two out of three approaches, as well as the helper function have been designed to specifically work with MATLAB functions which have been specifically written to accept a Power BI Table as input (or no input at all) and produce a table as output, see *MATLAB Code Interface*

Hint: If none if these workflows work for your specific MATLAB function and modifying the MATLAB function is not an option, see the alternative *Manual Approach* which offers more flexibility (at the cost of having to write more Power Query code).

Note: The MATLAB Production Server custom connector *can* also be used in reports published to Power BI online. This will work out of the box with a static snapshot of the data, *however* refreshing data online requires on-premises data gateway with the custom connector installed/enabled, see also *Usage in Power BI Online*. If you wish to publish refreshable online reports *without* needing on-premises data gateway, refer to the alternative *Manual Approach*, which does not require on-premises data gateway (if the MATLAB Production Server instances are reachable by Power BI Online directly) instead of using the custom connector approaches described below.

Note: When connecting to a server/instance for the first time from Power BI, it may ask to configure/confirm Data Source permissions. That can then be done on various path levels (server/instance root, CTF, specific function). It is recommended to do this on the highest (server/instance root) level such that the custom connector can then work with *all* CTF archives and *all* functions deployed to this instance (without having to ask for permission again) *and* it can then also query the discovery end-point on `/api/discovery` as well as server status on `/api/health`:

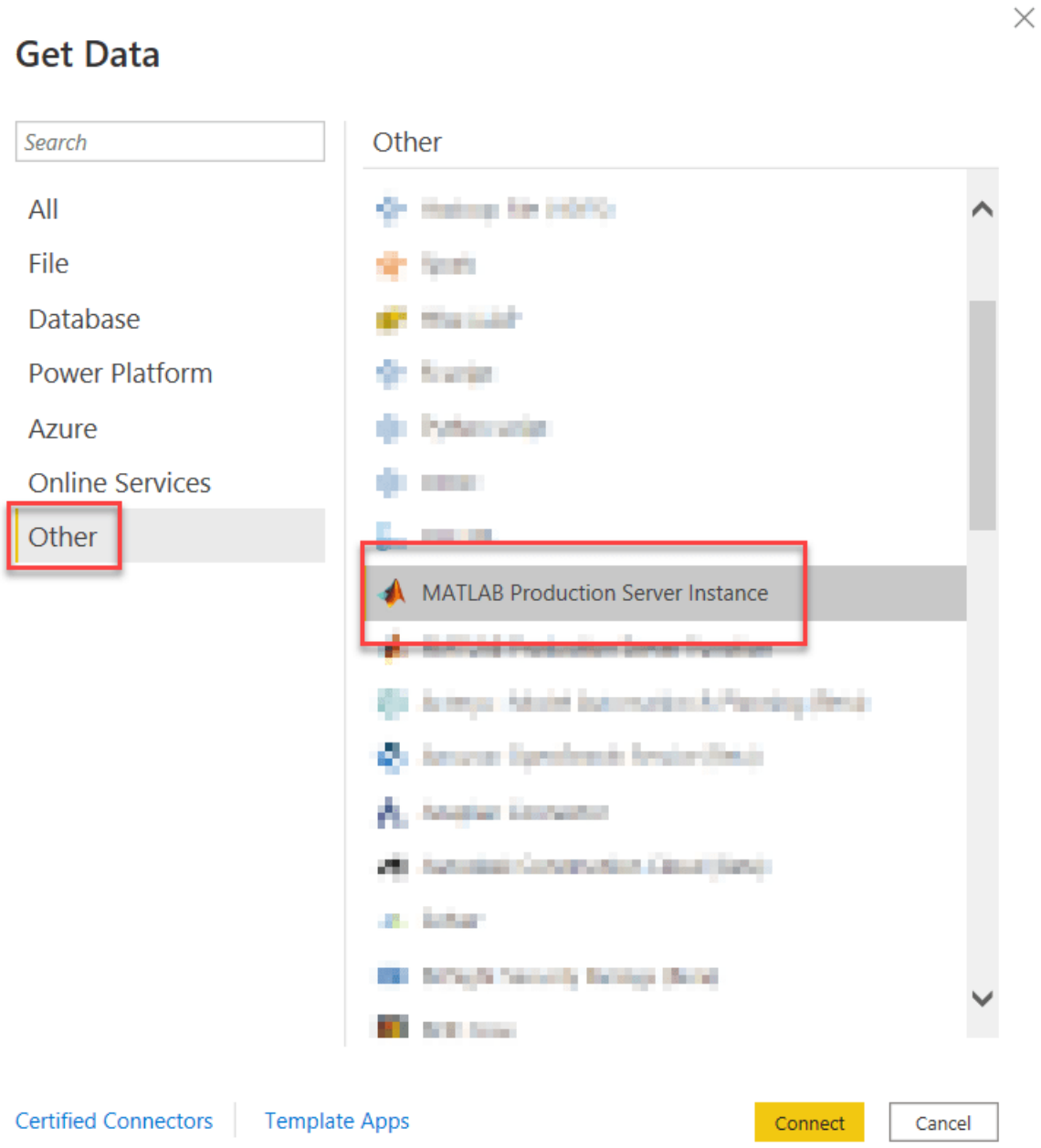


3.2.1 MATLAB Production Server Instance *Data Source*

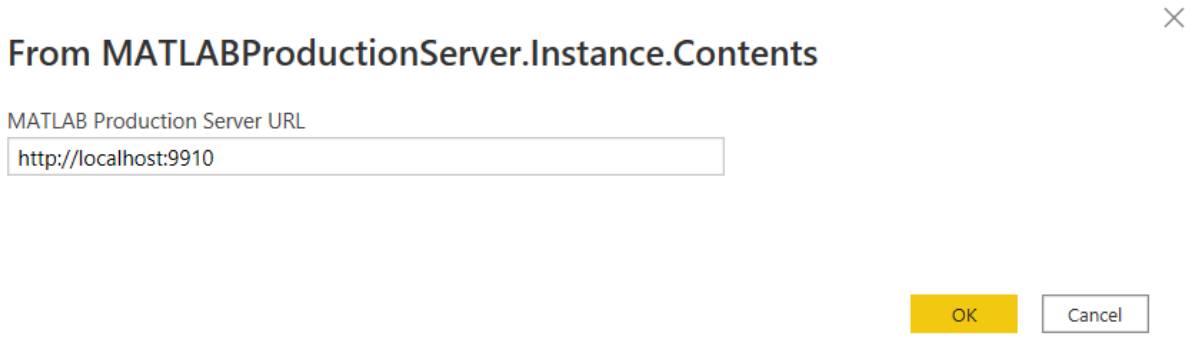
Important: This workflow is only valid when working with the recommended *one (or no) input table, output tables approach*

The "MATLAB Production Server *Instance*" Data Source allows you to connect to a MATLAB Production Server instance which has the discovery service *enabled*.

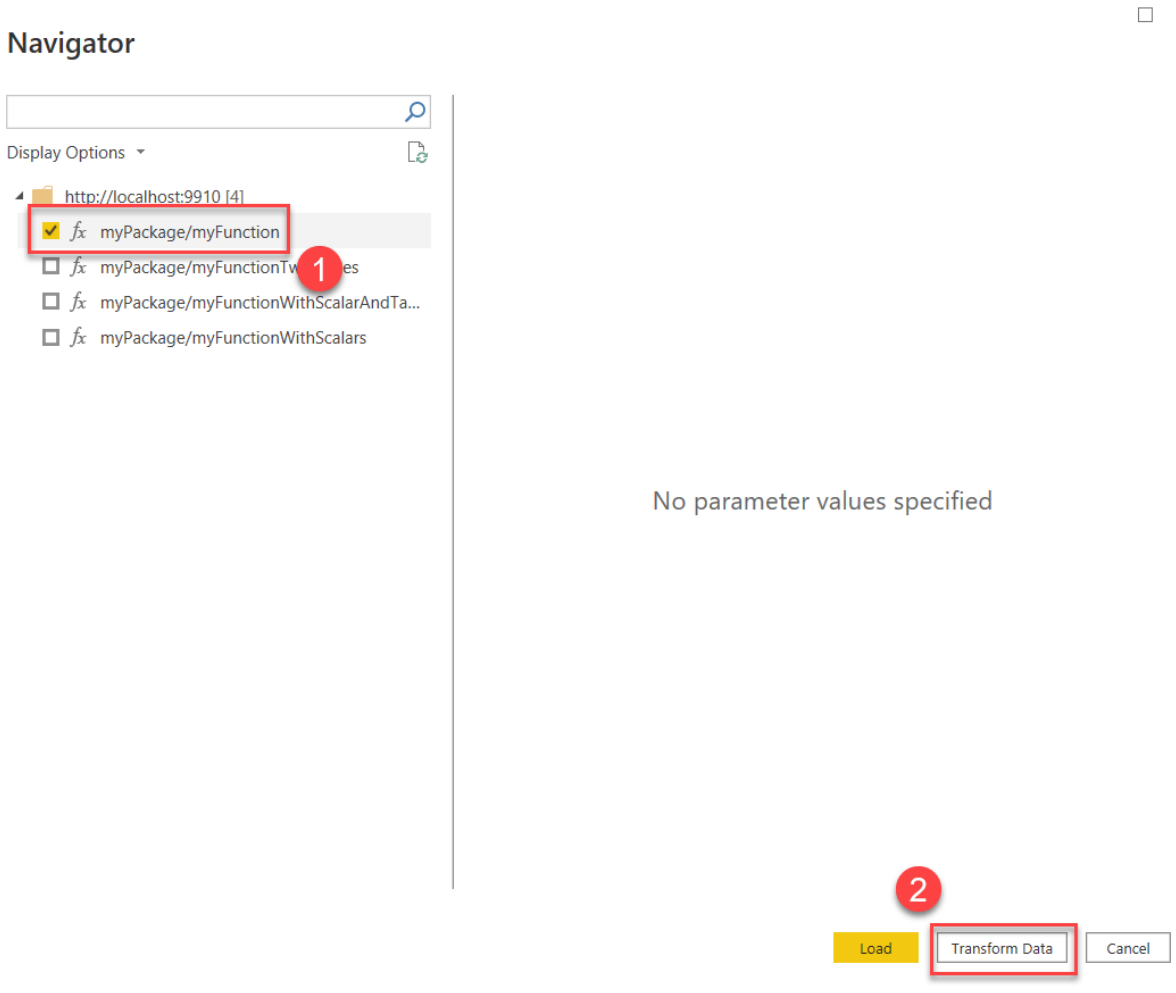
The data source is available under `Get Data` → `Other`



After entering the MATLAB Production Server instance URL:

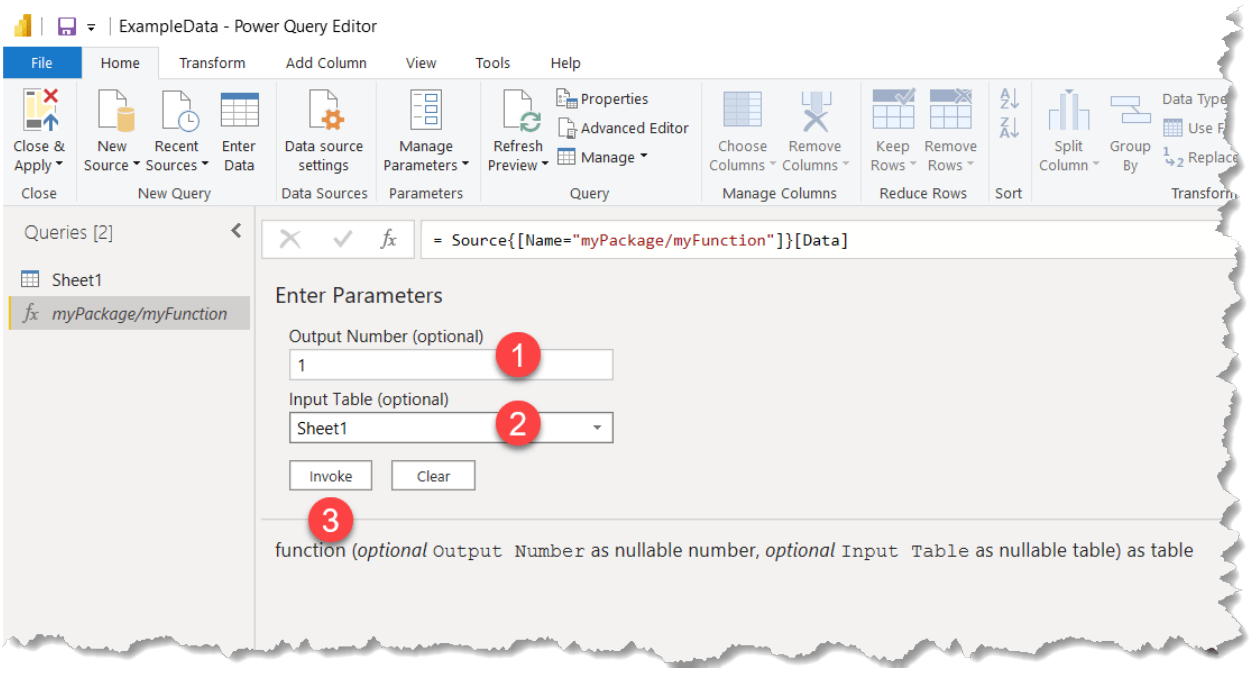


The functions, discovered through the discovery API, are listed. You can then select the function you want to call (1) and then hit (2) `Transform Data`.

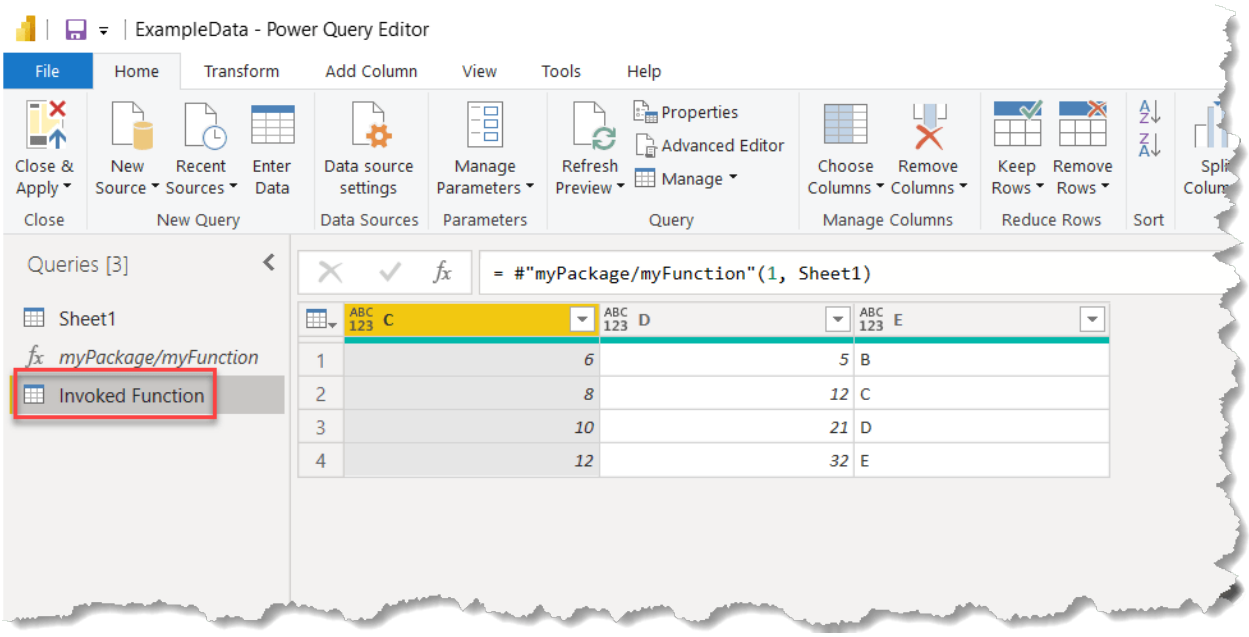


Which takes you to the Power Query Editor where then:

1. If the function outputs multiple tables, specify which of the outputs you want to get, and
2. Optionally an input table can be provided, and
3. The function can then be invoked using `Invoke`



This should create a new "Query" which shows the result:



To call the same function with a different input or requesting another output, you can select the function from the "Queries" list again and enter different parameters. When then hitting `Invoke` again this will add a *new* Query with a different input (preserving the old invocation/output as well).

If the function produces more outputs and you want to get them all, it can be beneficial to use a lower level approach where the function is called only once, requesting all outputs in one call, and then multiple other queries can be added which reference this result to produce multiple tables; the *Sunspot Example* uses such an approach.

To call other functions from the same- or another MATLAB Production Server instance, repeat the procedure from the start.

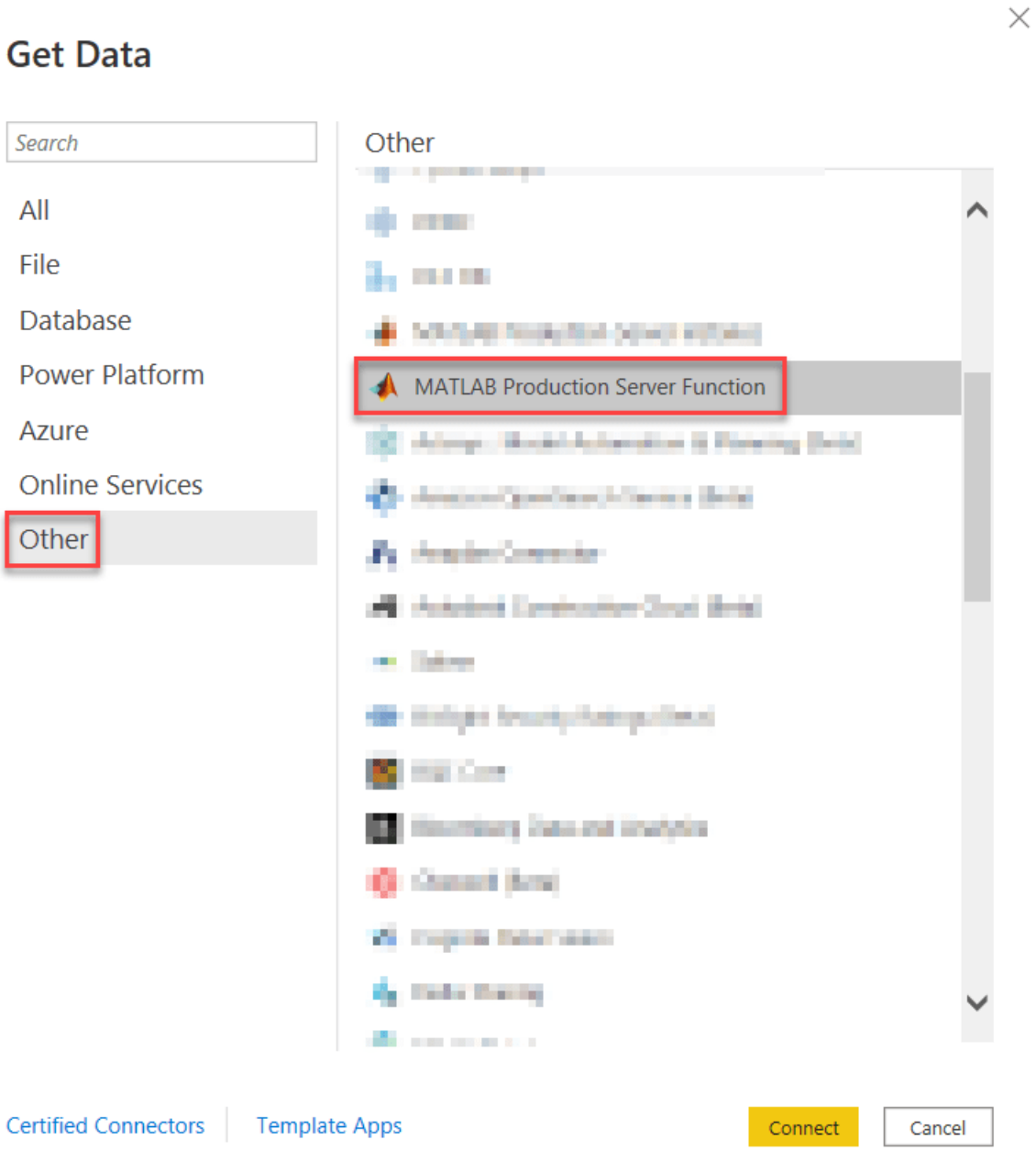
3.2.2 MATLAB Production Server Function

Important: This workflow is only valid when working with the recommended *one (or no) input table, output tables approach*

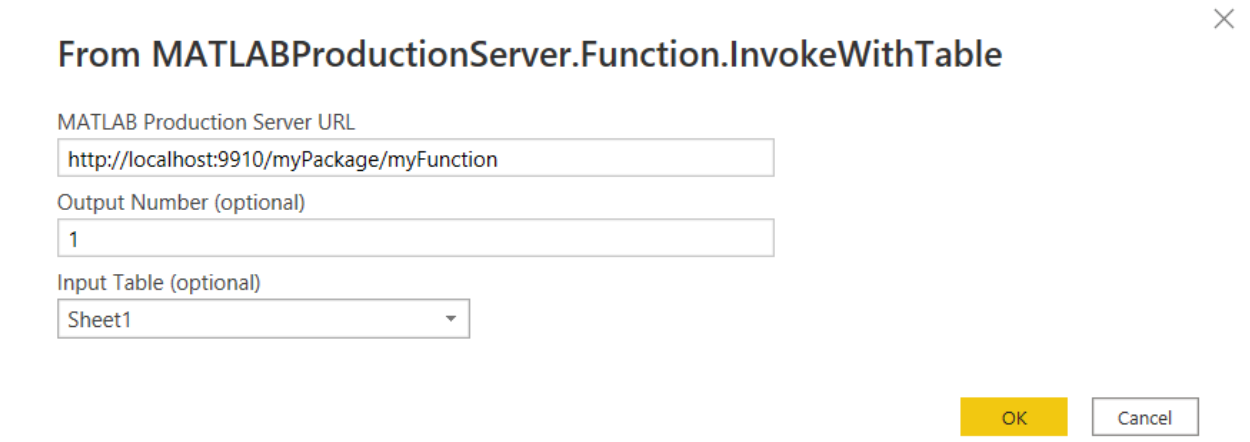
For MATLAB Production Server instances without discovery API enabled, or if you simply already know which exact function from which exact archive you want to call, it is also possible to *directly* invoke the function. This can be done as Data Source through `Get Data` or by directly invoking `MATLABProductionServer.Function.InvokeWithTable` from a Query.

MATLAB Production Server Function *Data Source*

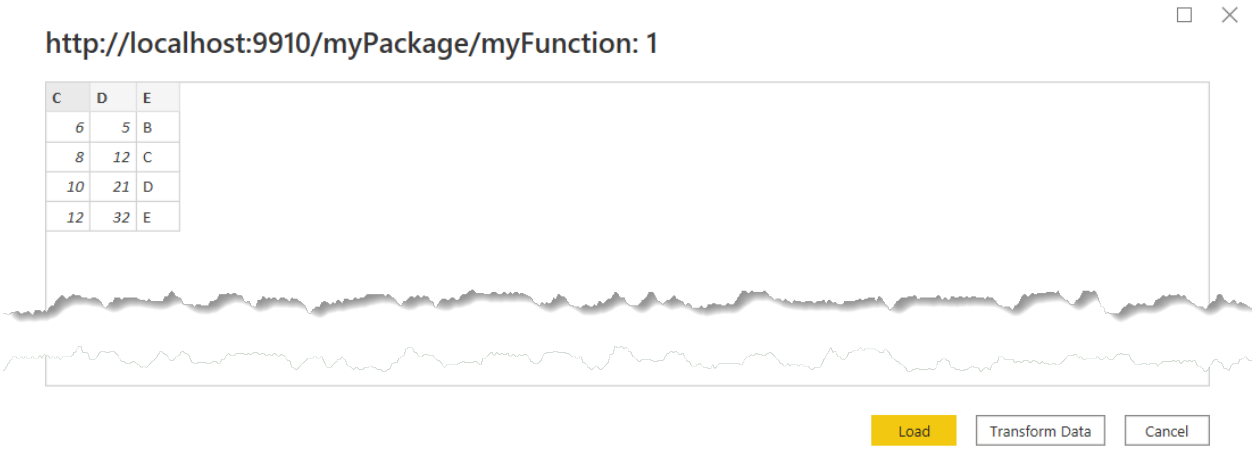
The data source is available under `Get Data` → `Other`



When connecting to this source, you directly fill out the full URL including the archive name and function name and you can also immediately choose which output you want if there are multiple outputs as well as an optional input:

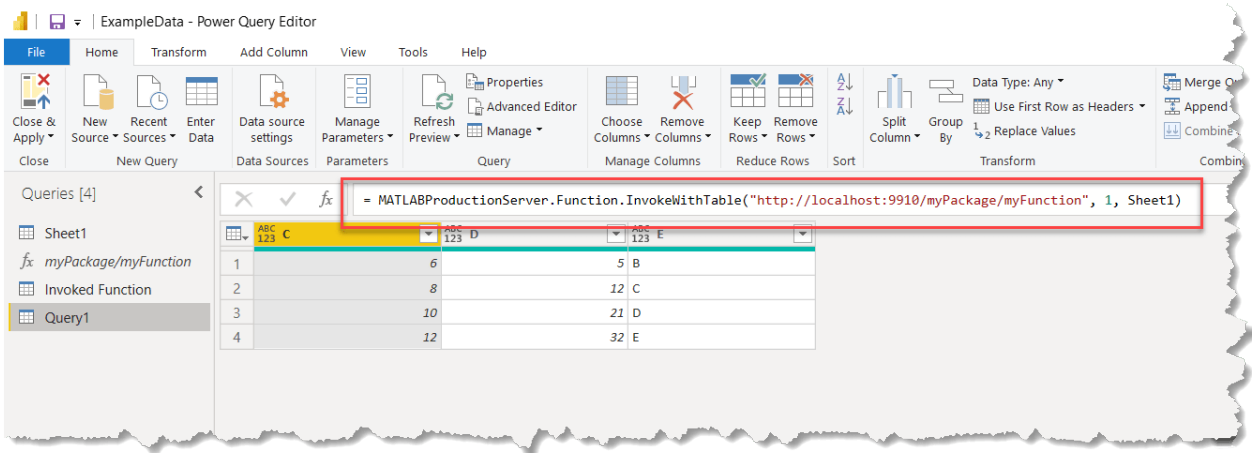


Which will then directly execute the function and show the data which can then be imported or further transformed:

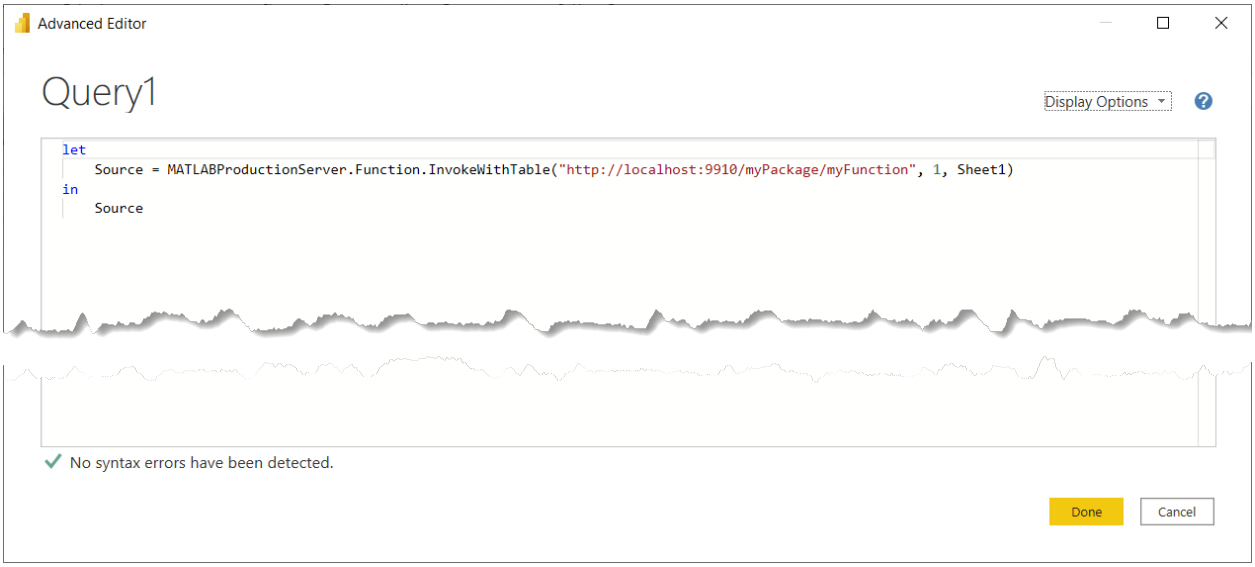


MATLABProductionServer.Function.InvokeWithTable *function*

The same functionality can also directly be invoked from a Query through `MATLABProductionServer.Function.InvokeWithTable`:



Or in the Advanced Query Editor:



3.2.3 MATLABProductionServer.Function.Invoke *function*

If the MATLAB function does not directly follow the "one (or no) input table" approach, the function can still be invoked through `MATLABProductionServer.Function.Invoke`. This function can only be called from a Query/in the Query Editor, there is *no* Data Source for this under `Get Data`.

The function has three **inputs**:

1. The full URL including archive and function names.
2. The number of outputs. `nargout` parameter in REST API.
3. (Optional) Inputs. `rhs` parameter in REST API. Inputs will be encoded using `Json.FromValue`. Refer to its documentation to learn how various Power BI types are encoded and ensure that the provided inputs will indeed be serialized into a valid `rhs` parameter for your function. This may involve using `{}` to create lists or `[]` to create records, or using functions like `List.Combine`, `Table.ToRecords`, etc. See the Power Query M function reference. A few examples are documented *below*.

The function **output** is a `Json.Document` representation of the RESTful response.

If the output(s) of the function are still tables (returned as described in *MATLAB Code Interface*), *see how `MATLABProductionServer.TableResponseToTable` can be used* to transform this response into a Power BI Table. If the outputs are not tables you will have to further process the result by yourself through the (Advanced) Query Editor.

Hint: Alternatively, functions like these can also be called entirely manually not using the custom connector at all, see *Manual Approach*.

Example: function with two scalar inputs

The following MATLAB Function which takes two scalars as input:

```
function out = myFunctionWithScalars(a,b)
    %% PowerBI Input/Output Handling
    % Actually with a simple scalar as input and no tables, no special input
    % processing is needed

    % Start a new table for the output
    outTable = table;

    %% Actual Algorithm
    % Write the actual algorithm to work with MATLAB tables

    % Just some simple example
    outTable.Result = a + b;

    %% PowerBI Input/Output Handling
    out = PowerBI.TableToOutput(outTable);
```

Can for example be called using:

```
response = MATLABProductionServer.Function.Invoke("http://localhost:9910/myPackage/
↪myFunctionWithScalars", 1, {11,31})
```

Example: function with a scalar input and a table

Another interesting specific situation is the following MATLAB function which has a scalar as input followed by a table:

```
function out = myFunctionWithScalarAndTable(myScalar, varargin)
    %% PowerBI Input/Output Handling
    inputTable = PowerBI.InputToTable(varargin);

    % Start a new table for the output
    outTable = table;

    %% Actual Algorithm
    % Write the actual algorithm to work with MATLAB tables

    % Just some simple example
    outTable.C = inputTable.A + myScalar;
    outTable.D = inputTable.B * myScalar;

    %% PowerBI Input/Output Handling
    out = PowerBI.TableToOutput(outTable);
```

Hint: In this function the scalar input is the first, *and not the last*, input on purpose. In this way it can be followed by `varargin` (if used at all, `varargin` must always the last input of a function) which as explained in *Data Marhshalling* helps with accepting "structures which can be converted to a table" as input. This example can also be extended to take

more than just one scalar inputs *followed by a table*.

This can be called using:

```
response = MATLABProductionServer.Function.Invoke("http://localhost:9910/myPackage/
↪myFunctionWithScalarAndTable", 1, List.Combine({{42},Table.ToRecords(Sheet1)}))
```

Here input table `MyInput` is first explicitly converted to a List of Records (which `Json.FormValue` would normally have done implicitly) such that *first* the scalar value (`42` in this example) can be prepend to it by using `List.Combine`. And *then* `Json.FromValue` can encode the entire input correctly.

3.2.4 Helper function `MATLABProductionServer.TableResponseToTable`

This function can be used if `MATLABProductionServer.Function.Invoke` is used to invoke a function with "alternative" inputs but the function *outputs* are still one or more tables (as in the two examples from the previous section). It takes the the table cell-array as input and produces a Power BI Table as output. This table cell-array will first have to be "indexed" from the result, i.e. typically you get the `lhs` field and then select which of the outputs you want.

So then the examples above could actually become Queries like:

```
let
    response = MATLABProductionServer.Function.Invoke("http://localhost:9910/myPackage/
↪myFunctionWithScalars", 1, {11,31}),
    outputTable = MATLABProductionServer.TableResponseToTable(response[lhs]{0})
in
    outputTable
```

or:

```
let
    response = MATLABProductionServer.Function.Invoke("http://localhost:9910/myPackage/
↪myFunctionWithScalarAndTable", 1, List.Combine({{42},Table.ToRecords(Sheet1)})),
    outputTable = MATLABProductionServer.TableResponseToTable(response[lhs]{0})
in
    outputTable
```

Or for a function which returns multiple output tables:

```
function [out1,out2] = myFunctionTwoTables(varargin)
    %% PowerBI Input/Output Handling
    inputTable = PowerBI.InputToTable(varargin);

    % Start a new table for the output
    outTable = table;

    %% Actual Algorithm
    % Write the actual algorithm to work with MATLAB tables

    % Just some simple example
    outTable.C = inputTable.A + inputTable.B;
    outTable.D = inputTable.A .* inputTable.B;
    outTable.E = char('A' + inputTable.A);
```

(continues on next page)

(continued from previous page)

```
    %% PowerBI Input/Output Handling
    out1 = PowerBI.TableToOutput(outTable);

    outTable.F = outTable.C + 1;
    out2 = PowerBI.TableToOutput(outTable);
```

This could be called using the following to return the first output:

```
let
    /* If we are only interested in the first output, we also only have to request the␣
↪first output */
    response = MATLABProductionServer.Function.Invoke("http://localhost:9910/myPackage/
↪myFunctionTwoTables", 1, Sheet1),
    /* Select item 0 (first output) from lhs and process into table */
    outputTable = MATLABProductionServer.TableResponseToTable(response[lhs]{0})
in
    outputTable
```

Or the following to return the second:

```
let
    /* To be able to get the second output we have to request both outputs */
    response = MATLABProductionServer.Function.Invoke("http://localhost:9910/myPackage/
↪myFunctionTwoTables", 2, Sheet1),
    /* Select item 1 (second output) from lhs and process into table */
    outputTable = MATLABProductionServer.TableResponseToTable(response[lhs]{1})
in
    outputTable
```

Note that since this specific function does in fact take a table as input, it can also be called directly with `MATLABProductionServer.Function.InvokeWithTable`, e.g. to obtain the second output table:

```
let
    outputTable = MATLABProductionServer.Function.InvokeWithTable("http://localhost:9910/
↪myPackage/myFunctionTwoTables", 2, Sheet1)
in
    outputTable
```

3.3 Usage in Power BI Online

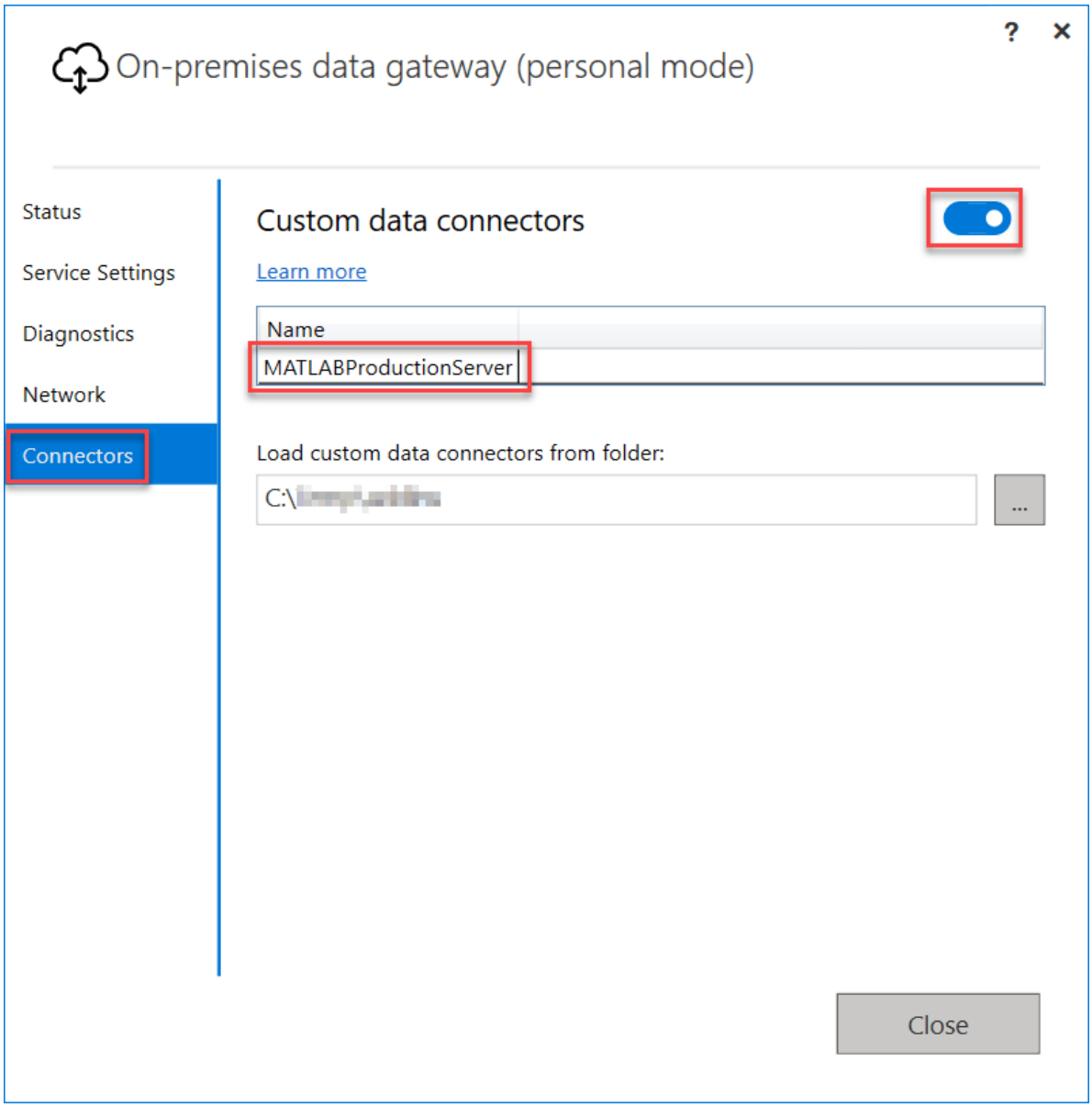
Power BI reports developed in Power BI Desktop using the MATLAB Production Server custom connector can also be published to Power BI Online. Out-of-the-box the report can be published with a *static snapshot* of the data produced by the MATLAB Production Server calls. In order for the report to be refreshable online (on-demand by an end-user or on a schedule) on-premises data gateway with the custom connector installed/enabled is required.

Hint: If you wish to publish refreshable online reports *without* needing on-premises data gateway, refer to the alternative *Manual Approach*, which does not require on-premises data gateway (if the MATLAB Production Server instances are reachable by Power BI Online directly) rather than using the custom connector.

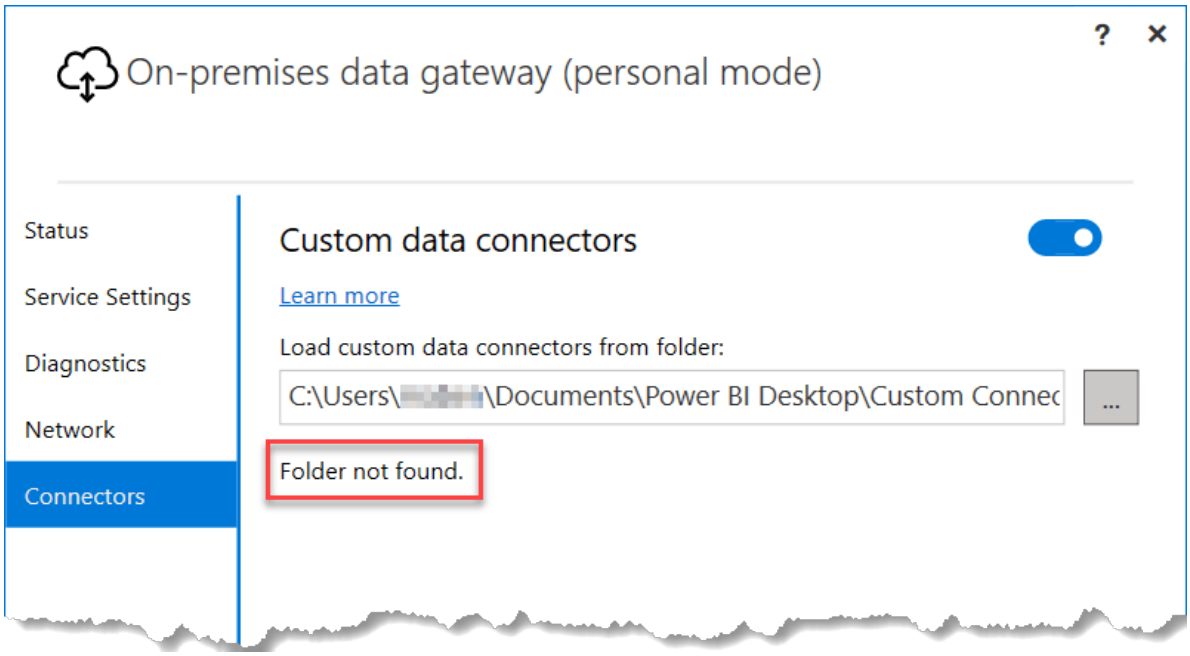
3.3.1 Example with on-premises data gateway (personal mode)

This example is written using "on-premises data gateway **(personal mode)**". Consult the Microsoft documentation to learn more about different gateway options and which option/mode is best for your needs.

1. Install on-premises data gateway (personal mode).
2. During/after installation, under `Connectors` verify that `Custom data connectors` is enabled and the MATLAB Production Server custom connector is listed:

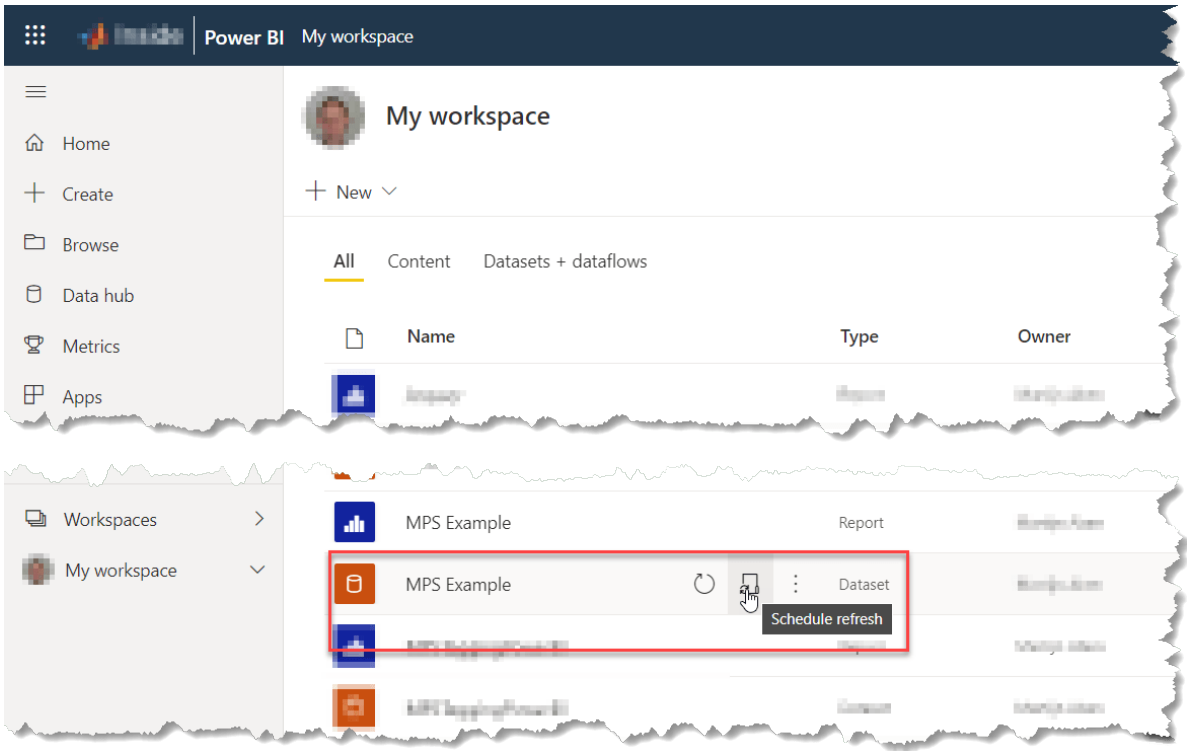


Hint: If you see a "Folder not found." error

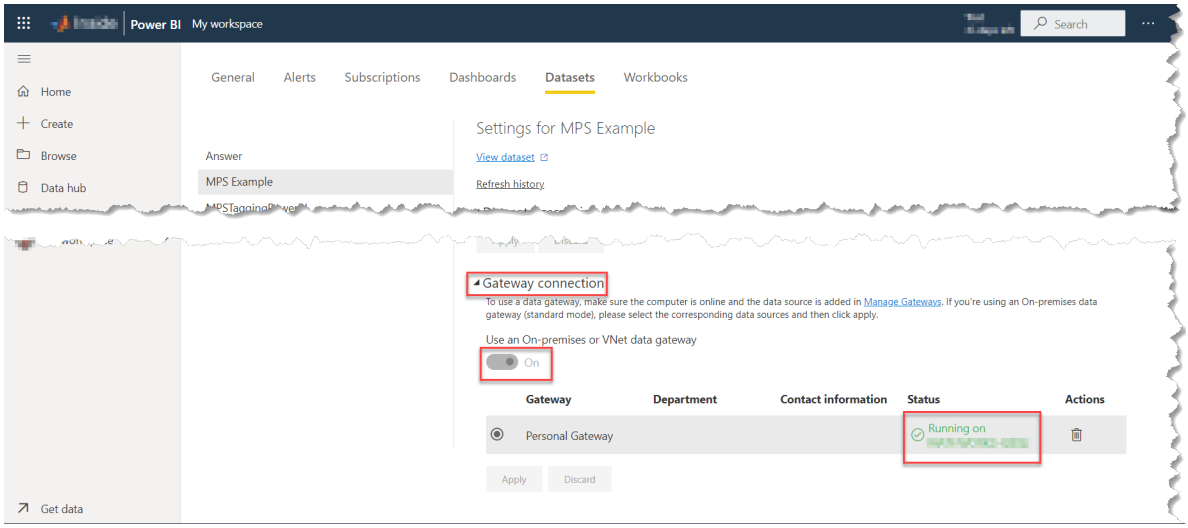


This is likely a permissions issue and you may have to grant the service account under which the on-premises data gateway (personal mode) runs access to the specified folder or alternatively choose a different location which this account can access and copy the MEZ-file (or PQX-file) into this location as well.

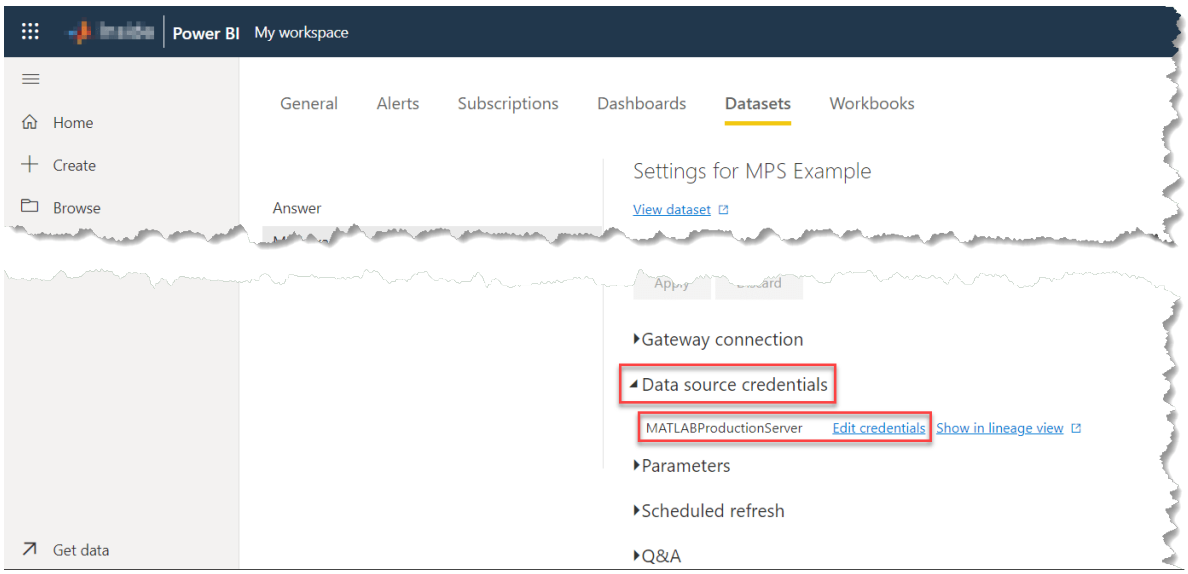
3. In your Power BI Online workspace find the Dataset for the published report and click `Schedule refresh` (even if you do not want to actually *schedule* refreshes and want on-demand refresh only):



4. Under `Gateway connection` ensure that `User an On-premises or VNet data gateway` is enabled and the correct gateway is selected and that its status is indeed running:

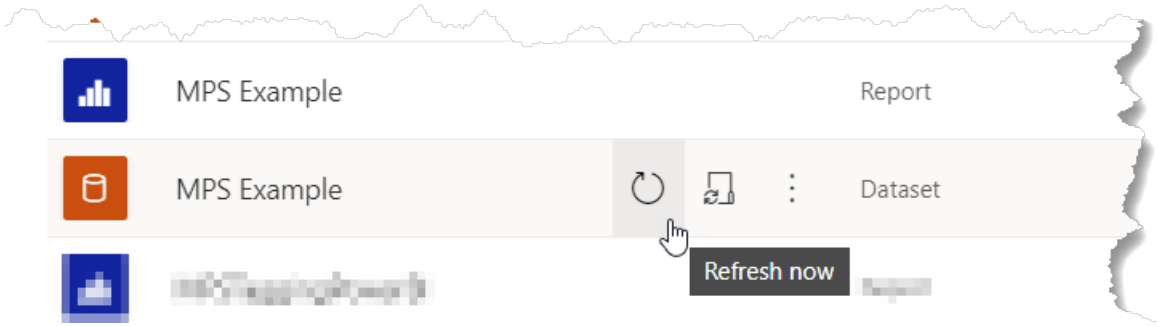


5. Under `Data source credentials` find the credentials for the MATLAB Production Server instance(s) the report works with and click `Edit credentials`:



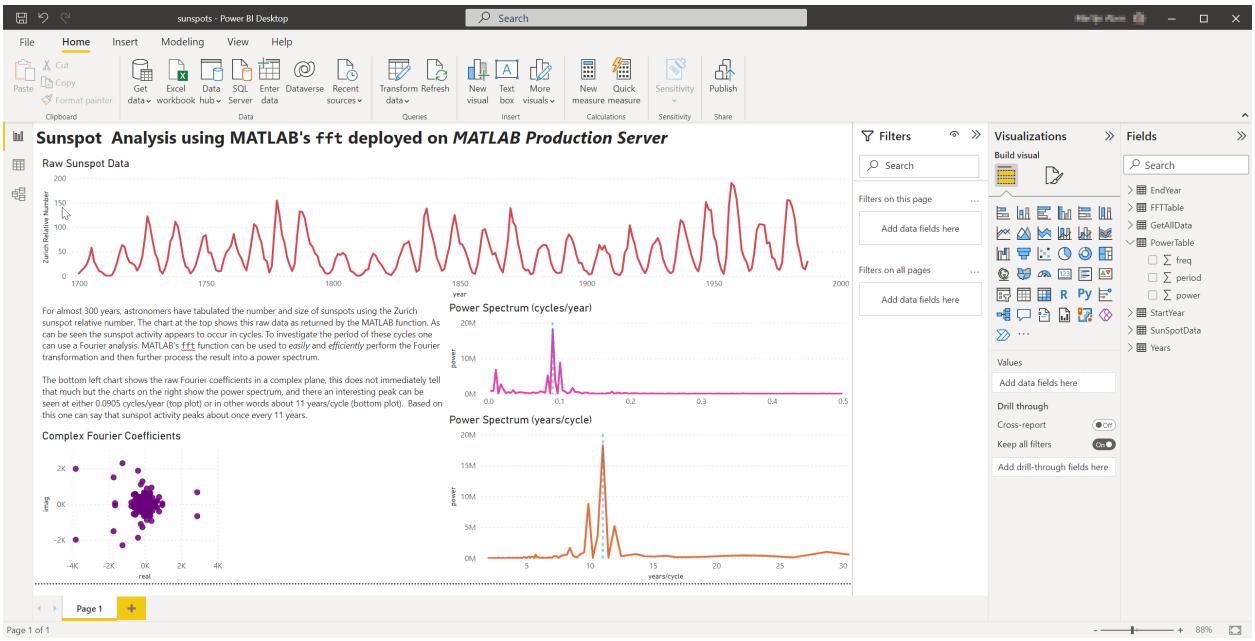
6. Enter the credentials and click `Sign In`. Note, this will *not* actually test the connection to the server and will always complete successfully. If something is wrong in the connection between on-premises data gateway (personal mode) and the MATLAB Production Server instance errors will occur during an *actual* refresh.

7. If you want to schedule automatic refreshes you can do so under `Scheduled refresh` or you can now go back to the Workspace view and try refreshing using `Refresh now`:



3.4 Examples

3.4.1 Sunspot Example



The example will allow Power BI users to call a MATLAB application and analyze cyclical data using a fast Fourier transform algorithm. Fourier transformations allow users to analyze variations in data, such as an event in nature over a period of time. The data retrieved here represents the number and size of sunspots for the last 300 years, using the Zurich sunspot relative number. The data retrieved from MATLAB can be plotted in Power BI to answer questions such as the frequency of peak sunspot activity, the power variation over the years etc.

The function used in this MATLAB application to perform Fourier transformation is `fft`, which has a lower computational cost when compared to other direct implementations. By integrating the MATLAB analysis with Power BI, it is possible to provide Power BI users direct access to powerful analyzing capabilities in MATLAB.

The MATLAB code and example PowerBI report are available in `Software\MATLAB\examples\Sunspots`.

MATLAB Code

This MATLAB code in this is based on the Analyze Cyclical Data with FFT example from the MATLAB documentation but has been modified for in- and output handling, also see *MATLAB Code Interface*.

```
function [RawOut, FFTOut, PowerOut] = getsunspotdata(varargin)
    %% PowerBI Input/Output Handling
    inputTable = PowerBI.InputToTable(varargin);

    % Start new tables for the outputs
    RawTable = table;
    FFTTable = table;
    PowerTable = table;

    % Load the data from the datafile included with MATLAB
    load sunspot.dat;
```

(continues on next page)

(continued from previous page)

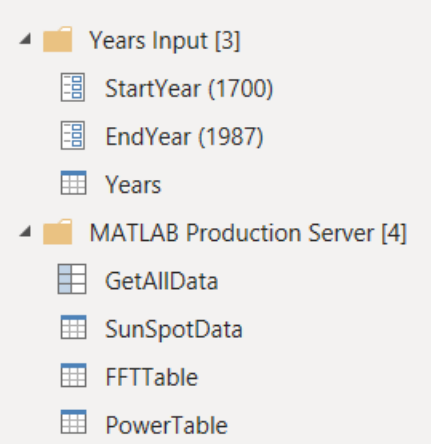
```
% The years are in the first column on of the loaded data
year = sunspot(:,1);

% Determine which years to use based on the input from PowerBI
idx = ismember(year,[inputTable.Years]);

% Output the raw data in RawTable
RawTable.year = year(idx);
RawTable.data = sunspot(idx,2);

% Compute the FFT
y = fft(sunspot(idx,2));
y(1) = [];

% Return the real and imaginary parts in two separate columns
FFTTable.realvalue = real(y);
FFTTable.imaginary = imag(y);

% Compute the power spectrum
n = length(y);
% Return this in a separate table
PowerTable.power = abs(y(1:floor(n/2))).^2;

maxfreq = 1/2;
PowerTable.freq = ((1:n/2)/(n/2)*maxfreq)';

PowerTable.period = 1./PowerTable.freq;

%% PowerBI Input/Output Handling
RawOut = PowerBI.TableToOutput(RawTable);
FFTOut = PowerBI.TableToOutput(FFTTable);
PowerOut = PowerBI.TableToOutput(PowerTable);
```

Deploy to MATLAB Production Server

Compile the function into a CTF archive named `SunSpots` and deploy to a MATLAB Production Server instance or use the Test Client feature inside MATLAB to host the component.

PowerBI

See `Software\MATLAB\examples\Sunspots\sunspots.pbix` and go to `Transform data` to see how the function can be called from PowerBI. A number of queries have been defined:



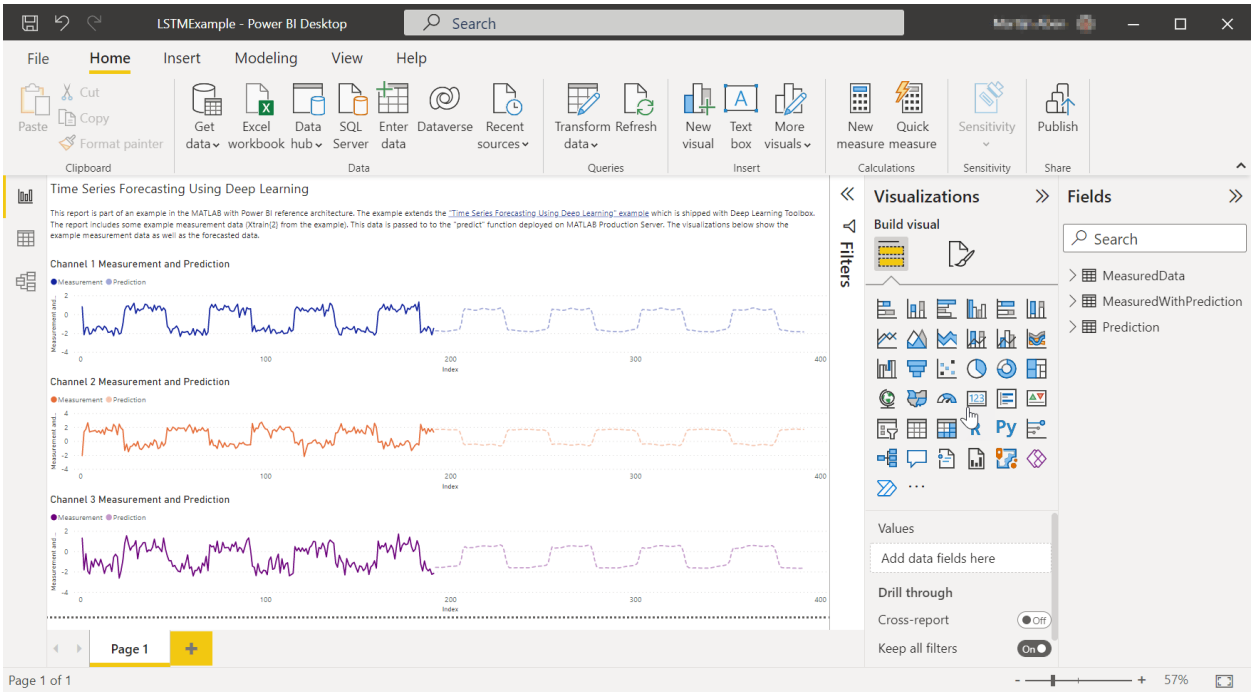
Years Input queries

As we can see by inspecting the MATLAB code, the MATLAB function expects a table with column `Years` as input, which specify which years from the included data to actually use in the analysis. This Table is created in PowerBI by taking a `StartYear` and and `EndYear` and then creating a range of values in between and then form the `Years` table.

MATLAB Production Server queries

In this example we choose to make use of `MATLABProductionServer.Function.Invoke` to call the function, such that we can retrieve all 3 output tables in one call. This is done in the `GetAllData` query. And then there are three queries which use `MATLABProductionServer.TableResponseToTable` and reference `GetAllData` to process the three outputs into PowerBI tables. These three queries also use some standard PowerBI functionality to convert the columns to the correct types.

3.4.2 Time Series Forecasting using Deep Learning Toolbox



Note: This example requires Deep Learning Toolbox.

This example is an extension of one of the examples shipped with Deep Learning Toolbox, see:

https://www.mathworks.com/help/releases/R2022a/deeplearning/ug/time-series-forecasting-using-deep-learning.html

It is recommended to fully work through this example inside MATLAB first to understand how this kind of forecasting and the network training works exactly; the example below only focusses on deploying the trained forecasting algorithm to MATLAB Production Server and calling it from Power BI.

The final MATLAB Code, example data, trained network and PowerBI report are included in the package in the `Software\MATLAB\examples\LSTMExample` directory.

Save the trained network

Having worked through the example shipped with Deep Learning Toolbox, the trained network should still be available as the `net` variable and it can be saved to a MAT-file:

```
>> net = resetState(net);
>> save net net
```

MATLAB Code

A MATLAB function can then be written which loads this trained network and then performs a closed loop forecast on input data. The code from the original example is slightly modified to accommodate the *one table input, one table output interface* which can be easily invoked from Power BI:

```
function [out] = predict(varargin)
    %% PowerBI Input/Output Handling
    inputTable = PowerBI.InputToTable(varargin);

    X = inputTable{:,["Channel1","Channel2","Channel3"]}';
    numChannels = 3;

    %% Prediction
    % Ensure the previously trained net is loaded
    persistent net
    if isempty(net)
        load('net','net');
    end
    % Use closed loop forecasting to predict the next 200 values
    net = resetState(net);
    [net,Z] = predictAndUpdateState(net,X);

    numPredictionTimeSteps = 200;
    Xt = Z(:,end);
    Y = zeros(numChannels,numPredictionTimeSteps);

    for t = 1:numPredictionTimeSteps
        [net,Y(:,t)] = predictAndUpdateState(net,Xt);
        Xt = Y(:,t);
    end

    %% PowerBI Input/Output Handling
    outputTable = table(Y(1,:)',Y(2,:)',Y(3,:)', ...
        'VariableNames',["Channel1", "Channel2", "Channel3"]);

    out = PowerBI.TableToOutput(outputTable);
```

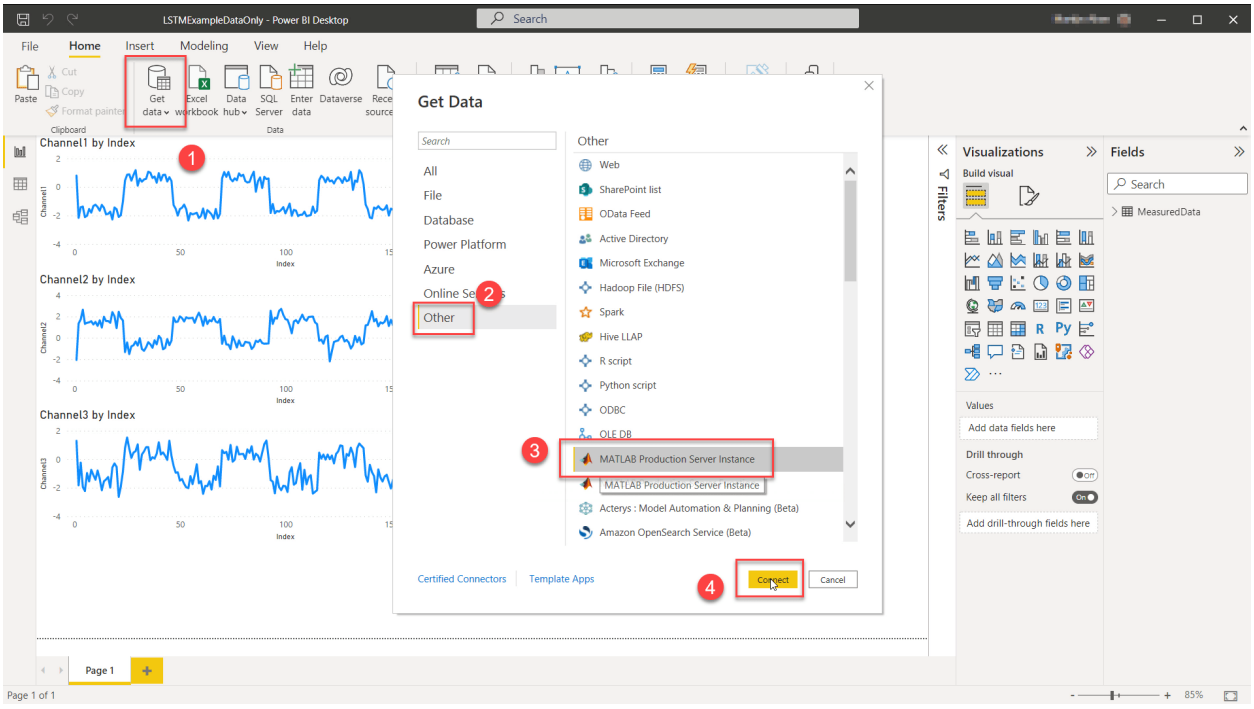
Deploy to MATLAB Production Server

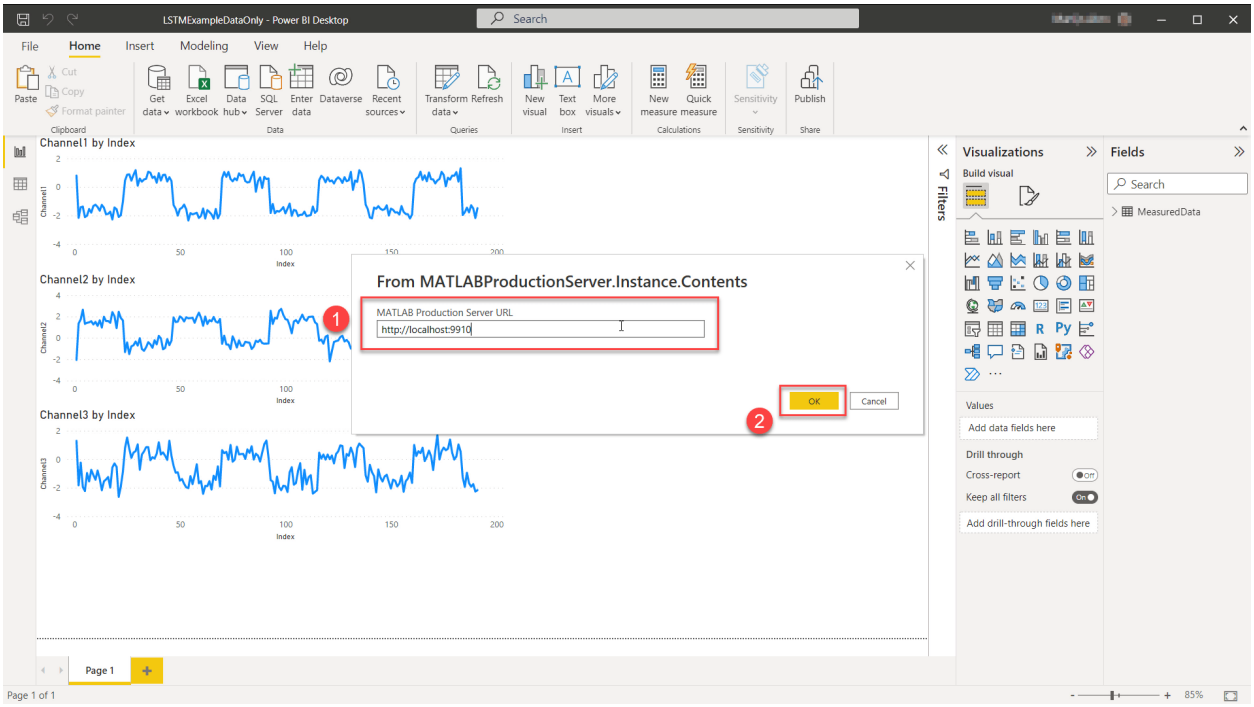
Compile the function into a CTF archive named `predict` and deploy to a MATLAB Production Server instance or use the Test Client feature inside MATLAB to host the component.

PowerBI

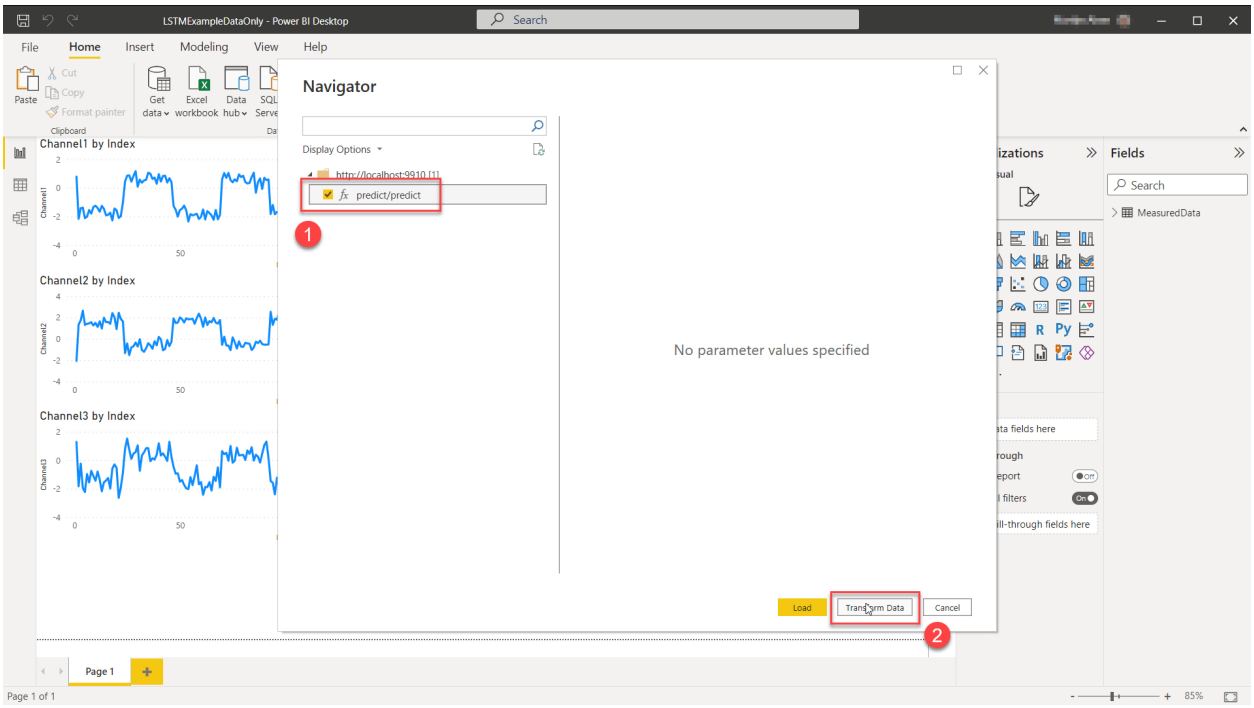
The PowerBI report included in this example `Software\MATLAB\examples\LSTMExample\LSTMExample.pbix` contains one of the example datasets (`XTest{2}`) from the Deep Learning Toolbox example in a table named `MeasuredData`.

Then using `MATLAB Production Server Instance`:

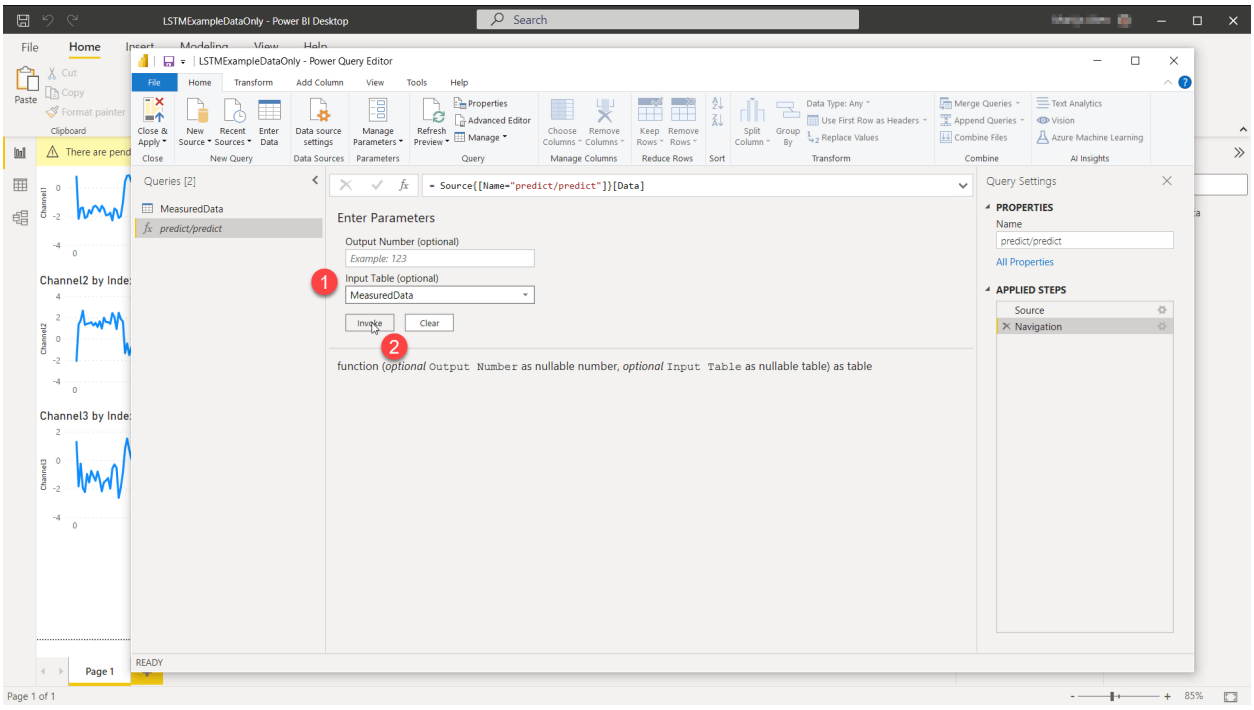




The `predict/predict` function was imported:



And it was invoked with `MeasuredData` as input:



An `Index` column was added and this query was then saved as `Prediction`. Further, a `MeasuredWithPrediction` query was created which concatenates the original `MeasuredData` and the forecast in `Prediction` together into one big table. Finally, all this data was then used in the visualizations to show the original data as well as the forecasted future data.

A video showing the full workflow in Power BI can be found on the website:

https://www.mathworks.com/products/reference-architectures/power-bi.html

CHAPTER

FOUR

REFERENCES

4.1 Data Marshalling

Note: This section contains additional *background information* on how the data marshalling works exactly when working with the *recommended "one (or no) input table, output tables" approach*. It is not absolutely necessary to understand this in full detail in order to be able to work with this approach.

When calling MATLAB Production Server functions from Power BI, this is done through MATLAB Production Server's RESTful interface. This RESTful interface works with requests in JSON format where the input data to the function is then encoded in JSON format as well.

To encode data into JSON format in Power BI `Json.FromValue` can be used.

Looking at these documentation pages, there is no straightforward one-to-one mapping between MATLAB and Power BI types. It will be difficult to generate the correct input using Power BI if the MATLAB function has not been designed for Power BI specific input, also it will be difficult to parse the result back into Power BI if the MATLAB function did not return the data in a particular convenient format. The following approach seems to work quite well though.

From the Power BI end provide a Table to `Json.FromValue` which will encode it as "an array of objects". Now an JSON "object" can map to a MATLAB struct *in short notation*, but unfortunately *short notation* does not support "*array of* objects"; so this cannot be used to directly pass an entire Power BI Table to MATLAB as *a single* input. However, since the `rhs` field in the request is an array as well, this *can* be used to pass in those objects as separate inputs. So a Table can be passed to MATLAB as a number of separate inputs, where each input structure then represents a row of the Power BI Table. Further, in MATLAB it is easy to define a function which takes a variable number of input. Also, concatenating a cell-array of structures into one big structure array is in fact a very simple operation in MATLAB (if all the structures have the same fields, which *will* be the case here). And, this structure array can then also easily be turned into a MATLAB table.

Similarly if we make the MATLAB function returns its outputs as a cell-array of structs and request the function to return the output *in short notation*, these outputs can relatively easily be transformed back into Tables on the Power BI end. Where again it is easy to get to this cell-array of structs format from a MATLAB Table.

The code template provided in *MATLAB Code Interface* makes use of these guidelines.

Hint: Apart from the workflow offered by the MATLAB Production Server Interface *for Power BI* as documented above, it is also possible to use manually written Power Queries to call MATLAB Production Server through its REST interface. If you need more flexibility than the package offers and you are comfortable with writing Power Queries, see *Manual Approach*

CHAPTER

FIVE

ALTERNATIVE

5.1 Manual Power Query Approach

Instead of using the MathWorks provided custom connector, it is also possible to directly interact with the REST Interface of MATLAB Production Server. This offers more flexibility at the cost of having to write your own Power Query in Power BI in order to make the call.

This for example allows you to make calls to functions which do not take a single table as input and produce a single table as output, as is expected by the custom data connector. Nevertheless, in many cases you may still want to follow the advice from the *Data Marshalling* section, as this in- and especially the *output* format does allow you to write relatively simple queries for processing the in- and outputs.

Another benefit of this approach is that it can be used in online reports without needing an on-premises gateway, also see *Usage in Power BI Online*

5.1.1 MATLAB Function with one table as input and one table as output

When calling a MATLAB function which does indeed follow the advice from the *Data Marshalling* section.

For example:

```
function out = myFunction(varargin)
    %% PowerBI Input/Output Handling
    % Concatenate the separate input structures into one big structure
    % array and convert it to a table
    inputTable = struct2table([varargin{:}]);

    % Start a new table for the output
    outTable = table;

    %% Actual Algorithm
    % Write the actual algorithm to work with MATLAB tables

    % Just some simple example
    outTable.C = inputTable.A + inputTable.B;
    outTable.D = inputTable.A .* inputTable.B;
    outTable.E = char('A' + inputTable.A);

    %% PowerBI Input/Output Handling
    % Convert the table to a struct array and then cell array
    out = num2cell(table2struct(outTable));
```

Then, the following Power Query can be used (which is in fact very similar to what the custom connector uses internally as well):

```
/* Call the Web Service*/
res = Json.Document(
Web.Contents("http://localhost:9910/myPackage/myFunction", /* Update with your server,␣
↪ctf and function name */
    [
        Headers=[#"Content-Type"="application/json"],
        IsRetry = true,
        Content=Json.FromValue([
            nargout = 1,
            outputFormat = [mode = "small"],
            rhs = MyInput /* MyInput is a Power BI Table here, update with your table */
        ])
    ])
),
/* Process the response into a Power BI table named "result" */
mwdata = res[lhs]{0}[mwdata],
resTable = Table.FromList(mwdata, Splitter.SplitByNothing(), null, null, ExtraValues.
↪Error),
result = Table.ExpandRecordColumn(resTable, "Column1", Record.FieldNames(mwdata{0}))
```

Hint: `IsRetry = true` is used to enforce PowerBI to always really re-execute the request and not return cached data.

5.1.2 MATLAB Function with scalars as inputs and table output

Consider the following MATLAB function which instead of a table takes two scalars as input and returns a table with just one column and one row as output:

```
function out = myFunctionWithScalars(a,b)
    %% PowerBI Input/Output Handling
    % Actually with a simple scalar as input and no tables, no special input
    % processing is needed

    % Start a new table for the output
    outTable = table;

    %% Actual Algorithm
    % Write the actual algorithm to work with MATLAB tables

    % Just some simple example
    outTable.Result = a + b;

    %% PowerBI Input/Output Handling
    % Convert the table to a struct array and then cell array
    out = num2cell(table2struct(outTable));
```

This can then be called using:

```
/* Call the Web Service*/
res = Json.Document(
Web.Contents("http://localhost:9910/myPackage/myFunctionWithScalars", /* Update with␣
↪your server, ctf and function name */
    [
        Headers=[#"Content-Type"="application/json"],
        IsRetry = true,
        Content=Json.FromValue([
            nargout = 1,
            outputFormat = [mode = "small"],
            rhs = {31,11}
        ])
    ])
),
/* Process the response into a Power BI table named "result" */
mwdata = res[lhs]{0}[mwdata],
resTable = Table.FromList(mwdata, Splitter.SplitByNothing(), null, null, ExtraValues.
↪Error),
result = Table.ExpandRecordColumn(resTable, "Column1", Record.FieldNames(mwdata{0}))
```

Where really only the `rhs = ...` statement had to be updated (as well as the function name in the URL) and it was updated to:

```
rhs = {31,11}
```

To quite simply provide two scalars `31` and `11` as inputs here. Since the output here is still a single table, processing the output has not changed.

5.1.3 MATLAB Function with a scalar *and table* as input and table output

Consider the following function which takes a scalar as first input and as second now a table, also the output is still a table:

```
function out = myFunctionWithScalarAndTable(myScalar, varargin)
    %% PowerBI Input/Output Handling
    % Concatenate the separate input structures into one big structure
    % array and convert it to a table
    inputTable = struct2table([varargin{:}]);

    % Start a new table for the output
    outTable = table;

    %% Actual Algorithm
    % Write the actual algorithm to work with MATLAB tables

    % Just some simple example
    outTable.C = inputTable.A + myScalar;
    outTable.D = inputTable.B * myScalar;

    %% PowerBI Input/Output Handling
    % Convert the table to a struct array and then cell array
    out = num2cell(table2struct(outTable));
```

In this function the scalar input is the first (and not the last) input on purpose. In this way it can still be followed by `varargin` which as explained in *Data Marhshalling* helps with accepting "structures which can be converted to a table" as second input.

This function can then be called using the following query:

```
/* Call the Web Service*/
res = Json.Document(
Web.Contents("http://localhost:9910/myPackage/myFunctionWithScalarAndTable", /* Update␣
↪with your server, ctf and function name */
    [
        Headers=[#"Content-Type"="application/json"],
        IsRetry = true,
        Content=Json.FromValue([
            nargout = 1,
            outputFormat = [mode = "small"],
            rhs = List.Combine({{42},Table.ToRecords(MyInput)})
        ])
    ])
),
/* Process the response into a Power BI table named "result" */
mwdata = res[lhs]{0}[mwdata],
resTable = Table.FromList(mwdata, Splitter.SplitByNothing(), null, null, ExtraValues.
↪Error),
result = Table.ExpandRecordColumn(resTable, "Column1", Record.FieldNames(mwdata{0}))
```

Where again, the only big change was to the `rhs = ...` statement:

```
rhs = List.Combine({{42},Table.ToRecords(MyInput)})
```

Input table `MyInput` is first explicitly converted to a List of Records (which `Json.FormValue` would normally have done implicitly) such that then first the scalar value (`42` in this example) can be prepend to it by using `List.Combine`. And then `Json.FromValue` can again be used to encode the entire request.

This example can easily be extended to accept additional other inputs.

Other in- and outputs

If needing to call functions with other inputs, consult the documentation of MATLAB Production Server's REST Interface and especially JSON Representation of MATLAB Data Types to learn more about how to specify the `rhs` parameter.

For outputs it is recommended to try to always return one or more tables, even if it is just one output with one column and/or one row such that you can consistently use the following relatively simple Power Query to parse the output to a Power BI table which can then easily be processed further:

```
mwdata = res[lhs]{0}[mwdata],
resTable = Table.FromList(mwdata, Splitter.SplitByNothing(), null, null, ExtraValues.
↪Error),
result = Table.ExpandRecordColumn(resTable, "Column1", Record.FieldNames(mwdata{0}))
```

This is not a hard requirement though and the MATLAB function is allowed to return any types- and numbers of outputs as supported by MATLAB Production Server, but further processing such outputs may require further customized (complicated) (Power) Queries in Power BI.