

MatDeck

MATLAB

.m support

Open, Evaluate, and Import MATLAB files in MatDeck · User manual

Contents

Introduction 3

Two ways to use a MATLAB file 3

New, Open, and Save 3

Evaluate with MATLAB or Octave 4

Stop and Step Evaluate 5

Colour, completion, and Functions Help 5

MATLAB names inside MatDeck SCRIPT 6

Import as a MatDeck document 6

What the converter changes 7

Vector indexes 8

Graphs on the canvas 9

Scripts Import will not open 10

Examples shipped with MatDeck 11

Building the Windows installer 12

Introduction

MatDeck opens, edits, saves, and runs MATLAB .m files. The same file can stay a MATLAB script, evaluated by MATLAB or Octave, or it can be imported onto a MatDeck canvas, where Evaluate draws native MatDeck graphs and runs MatDeck SCRIPT.

This manual describes both paths, the editor help that sits behind them, and the exact MATLAB subset Import will turn into a canvas document. It matches the MATLAB support on the current MatDeck build. There is no MATLAB App Designer in this version, and MatDeck does not read binary .mat data files.

Please note MatDeck does not register *.m with Windows or the desktop. Double-clicking a .m file outside MatDeck still opens whatever program already owns that extension. Inside MatDeck, use File → Open.

Two ways to use a MATLAB file

Every .m file has two honest uses. They share the source text and the examples folder. They do not share the engine that runs the file.

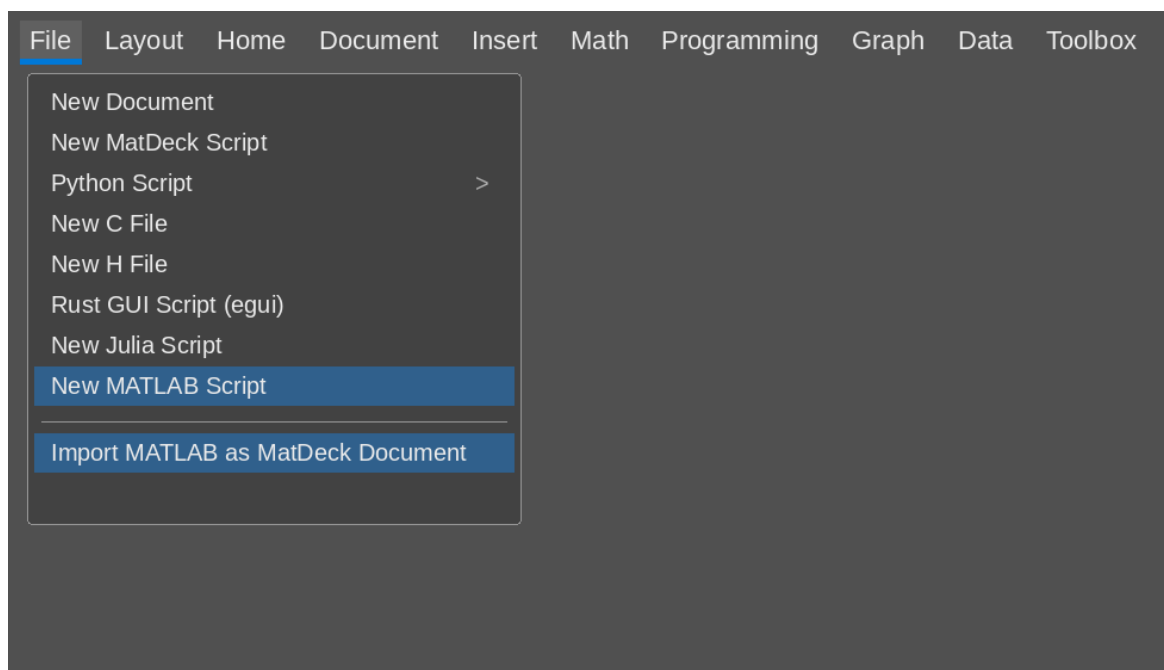
Path	Menu	What Evaluate runs
Open	File → Open, or File → New → New MATLAB Script	MATLAB, or Octave if MATLAB is not installed. Figures are MATLAB or Octave figures.
Import	File → New → Import MATLAB as MatDeck Document. The same command is on File → Open.	MatDeck SCRIPT on a canvas. plot() fills MatDeck graphs. MATLAB is not required.

Open keeps MATLAB syntax: square-bracket matrices, end to close a block, 1-based parentheses, and classes. Import rewrites a supported subset into SCRIPT (:=, braces, double-quoted strings) and refuses the rest before it creates a canvas. A file that Import refuses is still a normal Open file.

Please note The original .m tab is left unchanged when you Import. Save the canvas with Save As .mdd. Saving the canvas back to .m would discard the graphs and the SCRIPT.

New, Open, and Save

File → New → **New MATLAB Script** opens an empty MATLAB editor. It is a script editor, not App Designer and not a GUI form.



File → New. New MATLAB Script opens the editor. Import MATLAB as MatDeck Document builds a canvas.

File → Open lists *.m among the supported types. The match is case-insensitive, including drag-and-drop onto the window. The tab stays in the MATLAB language: comments use %, and Evaluate still means MATLAB or Octave.

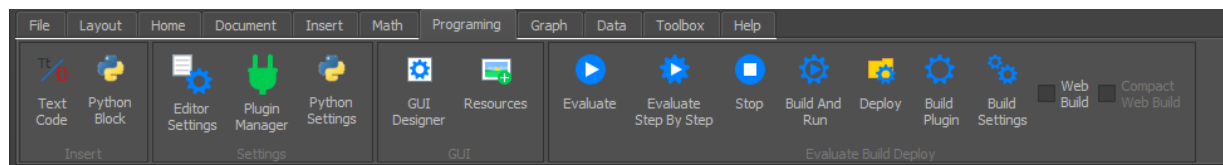
Save and Save As write UTF-8 .m source. MatDeck keeps its own editor metadata in a sidecar file with the extension .mmat, next to the script. That sidecar is not a MATLAB binary .mat file. An older .mm sidecar is still read once; after the next successful .mmat save, the .mm file is removed.

When both the .m file and the sidecar exist, MatDeck loads editor metadata from the sidecar and loads the script text from the .m file. An edit made in MATLAB or Octave on disk is what you see when you reopen the file in MatDeck. A corrupt sidecar is ignored and the plain .m text is opened. An empty .m file is kept as empty; it is not replaced by stale sidecar text.

Please note Evaluate of a MATLAB tab that has never been saved writes a private temporary .m file for that editor. It does not reuse one shared script.m for every unsaved tab.

Evaluate with MATLAB or Octave

On a MATLAB tab, Evaluate and the Evaluate buttons on the Math and Programming ribbons launch an interpreter. MatDeck looks for MATLAB first, then Octave. The working directory is the folder that contains the script, so relative load and addpath paths resolve beside the file.



Programming ribbon. Evaluate, Evaluate Step By Step, and Stop are in the Evaluate Build Deploy group. The Math ribbon has the same three commands.

MATLAB. MatDeck prefers a real matlab.exe under bin\win64 over the MATLAB launcher. The run uses batch mode (R2019a or newer). A diary file keeps disp output while a figure

window is open. The run waits on the figure window. Do not add pause at the end of the script to hold the window.

Octave. MatDeck prefers `octave.exe` over `octave-cli.exe` on Windows, and starts Octave with `--no-gui`. It selects the Qt toolkit when that toolkit is installed, forces figures to be visible, and flushes the figure before waiting. On Linux and macOS the launcher is a shell script with its own name, so two Evaluates do not share one script file.

The console. Windows opens its own console window and types the captured interpreter output into it. If the usual launch path fails, MatDeck falls back to a shell execute. The console stays up at the end so you can read `disp` output. If neither MATLAB nor Octave can be found, MatDeck says so. It does not pretend the script ran.

A small script to try is `examples/matlab/hello_matdeck.m`:

```
disp('Hello from MatDeck MATLAB support');
x = linspace(0, 2*pi, 200);
y = sin(x) .* cos(x);
disp(['max = ' num2str(max(y))]);
```

The plot lines in that file are comments. Uncomment `figure` and `plot` when you want an Octave or MATLAB figure. Close the figure window to let Evaluate finish.

Stop and Step Evaluate

Stop is on the same Math and Programming ribbons as Evaluate. It kills the MATLAB or Octave process tree that this Evaluate started, using the process id MatDeck stored at launch. On Windows that is a tree kill, so the console and the interpreter both stop. Stop recognises a .m file by its extension, including when the tab was not already marked as a MATLAB editor. Stop also applies to a canvas that was imported from MATLAB.

Step Evaluate uses Octave's keyboard prompt, so you can pause inside the script. It is available only when Octave is the interpreter. If only MATLAB is installed, MatDeck explains that and runs a normal Evaluate instead. If the document is not a MATLAB file, or it cannot be saved, Step Evaluate says so rather than launching a debugger on the wrong tab.

Colour, completion, and Functions Help

A MATLAB tab is coloured by the MATLAB highlighter: % comments, keywords, strings, and the core function names. The same highlighter is used when that editor is printed or exported to PDF, and when a block is folded.

Typing in the editor offers completions from the MATLAB catalog. Hovering a catalog name shows the same summary Functions Help uses. Toggle Comment on a MATLAB tab adds or removes %, not //.

Help → Functions Help lists thirteen MATLAB groups at the top of the tree:

Group	What it covers
MATLAB Language	Scripts, functions, and control-flow names

Group	What it covers
MATLAB Constants	pi, inf, NaN, true, false, and related names
MATLAB Math	sin, exp, log, and the elementary maths set
MATLAB Arrays	zeros, ones, linspace, size, numel, and indexing helpers
MATLAB Strings	String and character helpers
MATLAB I/O	disp, fprintf names, file parts, and path helpers
MATLAB Plotting	plot, figure, hold, title, xlabel, legend, grid
MATLAB Programming	Programming helpers used from SCRIPT
MATLAB Linear Algebra	inv, det, rank, and related names
MATLAB Signal and Polynomials	Signal and polynomial names
MATLAB Logic	Logical tests and set-style names
MATLAB Types	class, isa, and conversion names
MATLAB Time	Clock and date names

Each catalog entry has a summary, a description, the result it returns, and an argument list. The highlighter, the completer, and Functions Help read one catalog, so a name that colours in the editor is the same name Help describes.

Please note The Functions Groups ribbon on a canvas stays a list of native MatDeck nodes. It does not insert MATLAB catalog items onto the canvas. Use Functions Help, or type the name in a SCRIPT or MATLAB editor.

MATLAB names inside MatDeck SCRIPT

The catalog names are also callable from an ordinary MatDeck script (File → New → New Script), with no MATLAB or Octave process. `zeros`, `linspace`, `disp`, and `plot` run inside MatDeck's maths engine.

Where a name already exists in MatDeck, MatDeck's meaning wins. `sin(x)` is the MatDeck sine. `log(x)` is log base 10. Natural log is `ln(x)`. Extra MATLAB shapes, such as `rand(3, 3)` or `size(A, 1)`, go through the MATLAB catalog layer.

SCRIPT is not MATLAB syntax. Assignments use `:=`, blocks use braces, and strings use double quotes. A short script that already speaks SCRIPT is `examples/matlab_in_matdeck_script.txt`. Use that when you want the catalog without converting a .m file.

Import as a MatDeck document

Import is for a script you want to Evaluate inside MatDeck, on a canvas, next to other MatDeck objects and other .mdd files. Choose **Import MATLAB as MatDeck Document** from File → New (under New MATLAB Script) or from File → Open (its own item, above the recent-file list). Pick a .m file, or import the MATLAB tab that is already open.

MatDeck then does three things, in order.

- bullet** It converts the MATLAB text to SCRIPT and checks that SCRIPT can actually run.
- bullet** If the check fails, it lists the reasons and does not open a canvas.
- bullet** If the check passes, it opens a canvas document: a title, one MatDeck graph per figure, then the full SCRIPT in a code box under the last graph.

The code box is a real script editor on the canvas. Graphs are separate graph objects. They are not painted inside the line numbers of the script. Each graph is 480 by 200 points, inset from the page edge so it is accepted onto the page. When the script has two or more figures, each extra graph goes on the following page. The script starts after the last graph. After layout, the page indicator stays on page 1, which is the first graph.

Evaluate on that canvas compiles the SCRIPT with MatDeck's own compiler and fills the graphs from `plot()`. It also evaluates any other MatDeck formulas you later drop on the same canvas. A syntax error is printed and Evaluate stops. The graphs are left unfilled, and the Play indicator on the tab is cleared. You do not get a half-drawn figure from a script that did not compile.

When the canvas opens, an information window lists what Import changed: a MATLAB function turned into a SCRIPT function, a commented plot turned on, or a missing closing brace inserted. The canvas heading is the document title. Known examples use a fixed title: Trading Desk, Aurora Triad, Harmonic Prism, Two-Tone Beat (`signal_lab.m`), and Damped Harmonic Oscillator (`damped_oscillation.m` and `oscillator_lab.m`).

Please note Commented `plot`, `title`, `xlabel`, and `ylabel` lines are turned on during Import, so Evaluate has a curve to draw. If a comment was only a note, delete that line in the SCRIPT box before you Evaluate.

What the converter changes

Import is a translation, not a second MATLAB. The SCRIPT you see in the code box is the program Evaluate will compile. The table is the translation for the subset MatDeck runs.

MATLAB	SCRIPT on the canvas
% comment	// comment
%{ block %}	/* block */
x = 1	x := 1
'text' or "text"	"text"
[1 2; 3 4]	[1, 2; 3, 4]
['n = ' num2str(x)]	strcat("n = ", num2str(x))
if / for / while ... end	if(...) {} for(...) {} while(...) {}
elseif	else if
else on its own line	} else, then a brace. Evaluate splits that so the block closes.
for i = 1:N	for(i := 1; i <= N; i = i + 1)

MATLAB	SCRIPT on the canvas
for i = 1:2:N	for(i := 1; i <= N; i = i + 2)
pi	pi()
x .* y on two vectors	x * y (element-wise for two row vectors)
close all	close("all")
clc / clear	clc() / clear()
[a, b] = size(M)	a := size(M) and b is dropped
function ... end	A SCRIPT function, with return of the first output

Command form without parentheses is a common MATLAB habit and a SCRIPT syntax error. `close all` contains a space, which SCRIPT reads as a broken string. Import writes `close("all")`, `clc()`, and `clear()` so the canvas compiles.

`.*` becomes `*` because, on two row vectors, SCRIPT `*` is already element-wise. That is what makes an imported parabola `x .* x` draw as a curve. A call such as `sin(x)`, `zeros(1, N)`, or `plot(t, price)` stays a call.

A fragment of `trading_desk.m` after Import:

```
N := 360
t := linspace(1, N, N)
price := zeros(1, N)
price[(1) - 1] = 100
for(i := 2; i <= N; i = i + 1)
{
    price[(i) - 1] = price[(i-1) - 1] * (1 + 0.0004 + wave + shock)
}
```

Vector indexes

MATLAB indexes with parentheses and starts at 1. SCRIPT indexes with square brackets and starts at 0. Import rewrites an index only when the name is a variable you already assigned, and only when the index is a single value: no colon, no end, and no comma.

The 1-based shift is written into the expression. Existing MatDeck documents keep the subscript operator they already use. A MATLAB index of 1 lands on SCRIPT index 0.

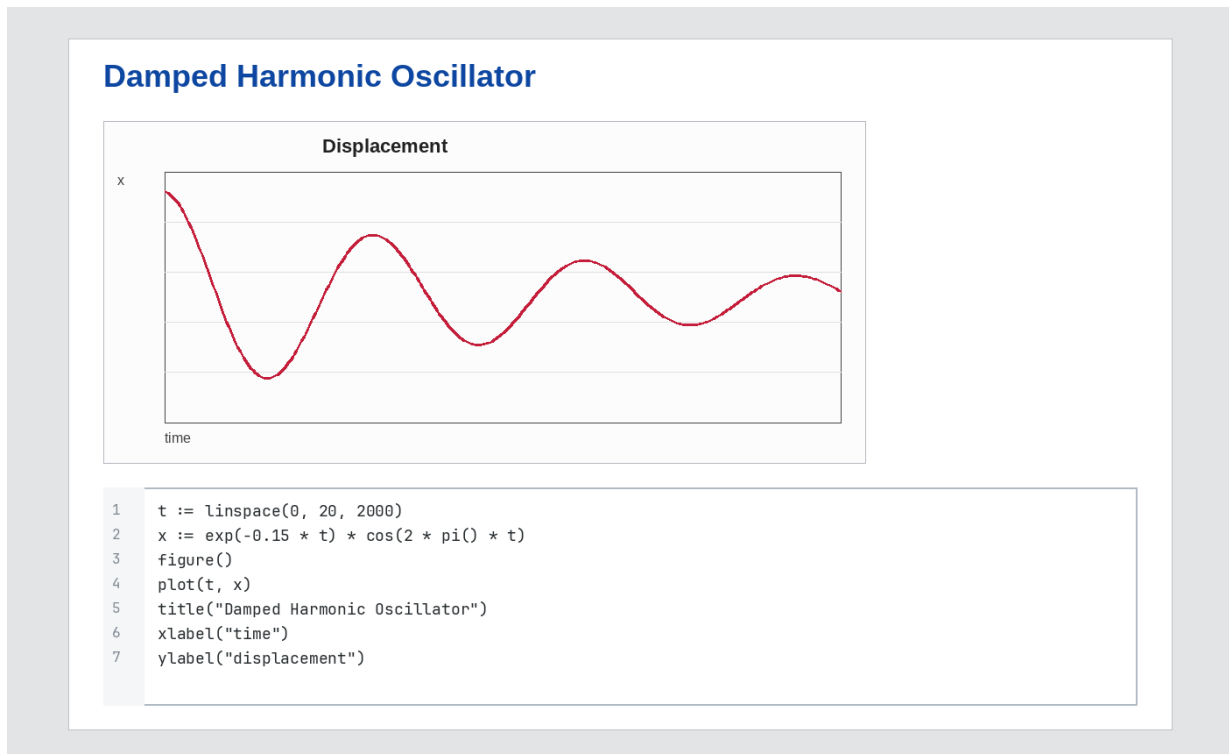
MATLAB	SCRIPT
price(1) = 100	price[(1) - 1] = 100
price(i) = price(i-1) * gain	price[(i) - 1] = price[(i-1) - 1] * gain
y = price(i) + sin(x)	y := price[(i) - 1] + sin(x)
sin(x), zeros(1, N), plot(t, price)	unchanged calls

The loop variable from `for i = 2:N` is a declared name, so `price(i)` later in that loop is an index. `zeros(1, N)` creates a row vector; the rewritten subscript writes that element. Nested indexes are rewritten from the inside out, so `price(i-1)` becomes `price[(i-1) - 1]`.

Please note A slice such as `price(i-19:i)`, a use of end inside parentheses, or a two-dimensional index `A(i, j)` is not rewritten. Those files are refused at Import. Open them and Evaluate with MATLAB or Octave.

Graphs on the canvas

Import reads `figure`, `hold`, `plot`, and the label commands, and places one graph object per figure before you Evaluate. Evaluate then fills those graphs. The labels you see are the ones the script sets, not a separate MATLAB figure window.



An imported canvas. The document title and the graph come first. The SCRIPT box is after the graph. Evaluate fills the curve.

- bullet figure** starts a new graph and turns hold off.
- bullet hold on** keeps the next curve on the current graph. **hold off** starts a new graph at the next plot. **hold** with no argument toggles.
- bullet plot** adds one curve. **scatter**, **stem**, **stairs**, **area**, **semilogx**, **semilogy**, and **loglog** draw on that same canvas graph.
- bullet title**, **xlabel**, **ylabel**, **zlabel**, and **grid** attach to the current graph.
- bullet legend** names the curves on the current graph, in plot order. Words such as Location or northeast are skipped; they are legend placement, not series names. Curve colours are assigned for you.
- bullet subplot**, **axis**, **xlim**, **ylim**, **box**, **colorbar**, and **colormap** are accepted and do not change the canvas graph.

Trading Desk's first figure is one graph with five curves:

```
figure; plot(t, price); hold on;
plot(t, fast); plot(t, slow); plot(t, upper); plot(t, lower);
title('Price, EMAs and bands');
legend('Price', 'Fast EMA', 'Slow EMA', 'Upper band', 'Lower band');
```

After Import that is one canvas graph. The SCRIPT keeps the calls in order:

```
figure()
plot(t, price)
hold("on")
plot(t, fast)
title("Price, EMAs and bands")
legend("Price", "Fast EMA", "Slow EMA", "Upper band", "Lower band")
```

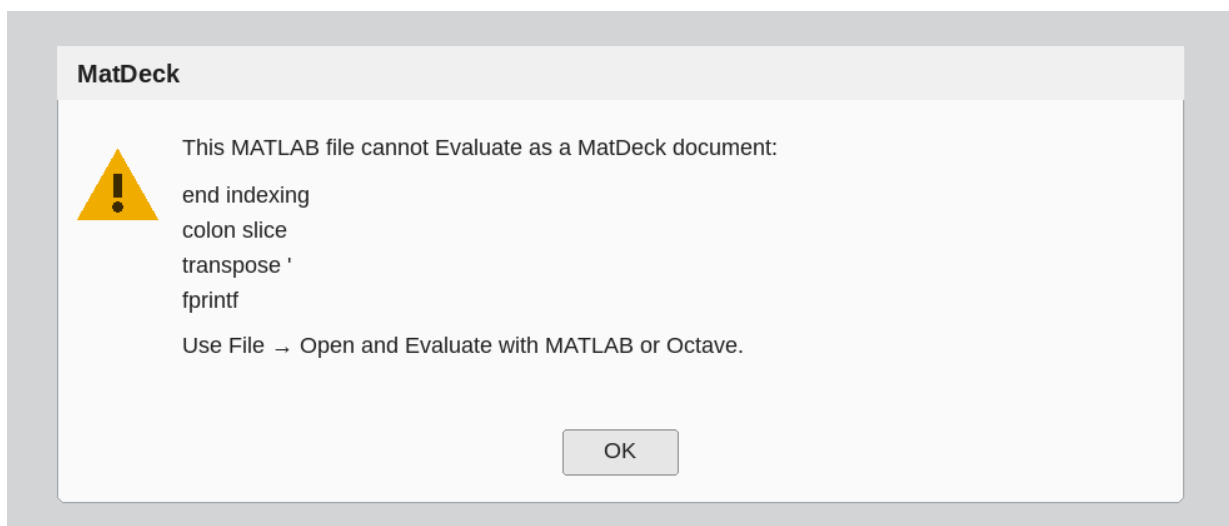
A second figure starts the next graph. `oscillator_lab.m` and `signal_lab.m` each produce three graphs, on successive pages, with the SCRIPT after the last one. `aurora_triad.m` and `harmonic_prism.m` keep three curves on a single graph because they use `hold on`. A short file such as `sine_wave.m` is one curve on one graph.

`plot(x)` with one argument uses 1, 2, 3, ... as the horizontal axis. `plot(x, y)` with an empty `y` draws nothing. It does not invent a flat line at zero.

Please note Changing the SCRIPT later does not add or remove graph objects by itself. The graphs were created from the script at Import. Evaluate fills the graphs that are already on the canvas, and can add an extra curve to a graph when the figure has more `plot` calls than the graph had series.

Scripts Import will not open

Before creating a canvas, Import scans the converted SCRIPT. If any of the following remain, a dialog lists them and Import stops. The file is not opened as a document that looks ready to Evaluate. Use File → Open and Evaluate with MATLAB or Octave.



Import refuses the file and lists the lines SCRIPT cannot run. Open the same .m file and Evaluate with MATLAB or Octave.

Listed reason	MATLAB you still have
end indexing	A(end), price(end)
colon slice	price(1:n), price(i-19:i)
multi-index	A(i, j) on a variable
transpose '	A' (a quote used as transpose)
power operator ^	x^2 or .^ SCRIPT ^ is bitwise xor. Use pow(a, b) if you rewrite it.
fprintf	fprintf(...)
switch/case	switch blocks
try/catch	try blocks
classdef	MATLAB classes
parfor	parallel for
unclosed {}	a block the translator could not close
parentheses index assignment	a variable index that could not be rewritten

quant_trading.m is the Open showcase because it uses several of these at once: transpose, colon slices, end, power, and fprintf. Import of that file stops and lists those reasons. The shorter trading_desk.m is the Import showcase: the same kind of desk (price, EMAs, bands, RSI, MACD, equity), written so the indexes rewrite and the four figures become four MatDeck graphs.

global and persistent are MATLAB keywords the canvas does not run. A script that depends on them belongs on File → Open. They are separate from the list above: that list is what the refusal dialog prints.

Please note A file can mix supported and unsupported lines. One unsupported line is enough for Import to stop. Removing that line, or using the trading-desk script instead, is what makes Import succeed.

Examples shipped with MatDeck

The .m examples live in examples/matlab. After installation they are in examples\matlab under the MatDeck folder. Each one has a matching .imported.txt. That text file is the SCRIPT Import places in the code box. Do not open it as a .m file. This manual is MATLABSupport.pdf, next to MATLAB_SUPPORT.md in the parent examples folder.

File	Open	Import
quant_trading.m	Seven figures: price and bands, signals, RSI, MACD, volatility, equity, drawdown	Stops. Lists end, slices, fprintf, transpose, and power.
trading_desk.m	Shorter desk: price, EMAs, bands, RSI, MACD, equity	Four graphs, three or more curves each. Title: Trading Desk.
aurora_triad.m	Three aurora curves on one figure	One graph, three curves. Title: Aurora Triad.

File	Open	Import
harmonic_prism.m	Three Fourier harmonics on one figure	One graph, three curves. Title: Harmonic Prism.
oscillator_lab.m	Displacement, envelope, and energy	Three graphs, then SCRIPT. Title: Damped Harmonic Oscillator.
signal_lab.m	Sine, beat, and power	Three graphs, then SCRIPT. Title: Two-Tone Beat.
damped_oscillation.m	$\exp(-t/2) \cdot \sin(4t)$	One graph. Title: Damped Harmonic Oscillator.
sine_wave.m	$\sin(x)$ over one period	One graph.
cosine_wave.m	$\cos(x)$ over one period	One graph.
parabola.m	$x \cdot x - x - 2$	One graph.
circle.m	Unit circle	One graph.
exponential_decay.m	$\exp(-t)$	One graph.
gaussian.m	Bell curve	One graph.
lissajous.m	3:2 Lissajous curve	One graph.
array_basics.m	disp, for, zeros, and a sine plot	SCRIPT with := and braces, plus one graph.
hello_matdeck.m	Console disp. The plot line is a comment.	Import turns that plot on, so the canvas has a graph.

Two more Open scripts sit in the parent examples folder: `matlab_core_functions.m` exercises the catalog names, and `matlab_plot_functions.m` exercises the plot names. Both use `try` and `fprintf`, so Import refuses them. `matlab_in_matdeck_script.txt` is already SCRIPT; it is not a .m file.

A first session, after MatDeck is built from this support:

- bullet File → New → New MATLAB Script. Type a disp line. Evaluate. Read the console.
- bullet File → Open examples/matlab/quant_trading.m. Evaluate with MATLAB or Octave.
- bullet File → New → Import MATLAB as MatDeck Document. Choose examples/matlab/trading_desk.m. Evaluate. Four graphs, then the SCRIPT.
- bullet Import quant_trading.m once, to see the refusal dialog, then close it and keep using Open for that file.
- bullet Help → Functions Help. Open any MATLAB group.
- bullet Save the canvas as a .mdd. Leave the .m file as the MATLAB copy.

Building the Windows installer

The MATLAB menus, the examples, and Import are in the MatDeck build that includes this support. The Windows kit is Qt 6.8.3 with MinGW 64-bit, normally `C:\Qt\6.8.3\mingw_64`. Keep the clone at `C:\MatDeck`, outside OneDrive.

From that folder, one command transfers the vendor tree, builds MatDeck, and compiles the installer:

```
cd /d C:\MatDeck  
MAKE-MATLAB-INSTALLER.bat
```

Install the new MatDeckSetup.exe that this command produces. An older copy under Program Files does not gain the MATLAB menus. The rebuilt program is also at C:\MatDeck\2Dgraph01\trunk\build_release\MatDeck.exe.

If you already have local edits, stash them, pull this branch, then run the installer command again. Leave the stash where it is. Do not register *.m in the installer script. The association header shipped with this branch registers MatDeck's own document types and deliberately leaves .m alone.

The installer copies the .m examples to \$INSTDIR\examples\matlab, and copies MATLABSupport.pdf and MATLAB_SUPPORT.md to \$INSTDIR\examples. A rebuild without the installer also leaves the examples next to MatDeck.exe under build_release\examples. It also places the compiler header the canvas Evaluate needs beside MatDeck.exe, together with the Qt 6 runtime. Canvas Evaluate of an imported script does not need a Qt software development kit on the machine that only runs MatDeck.

Please note Longer build notes, including a missing NSIS header or a missing module DLL, are in MATLAB_SUPPORT.md in the same examples folder. This manual is the description of the feature. That note is the build log.