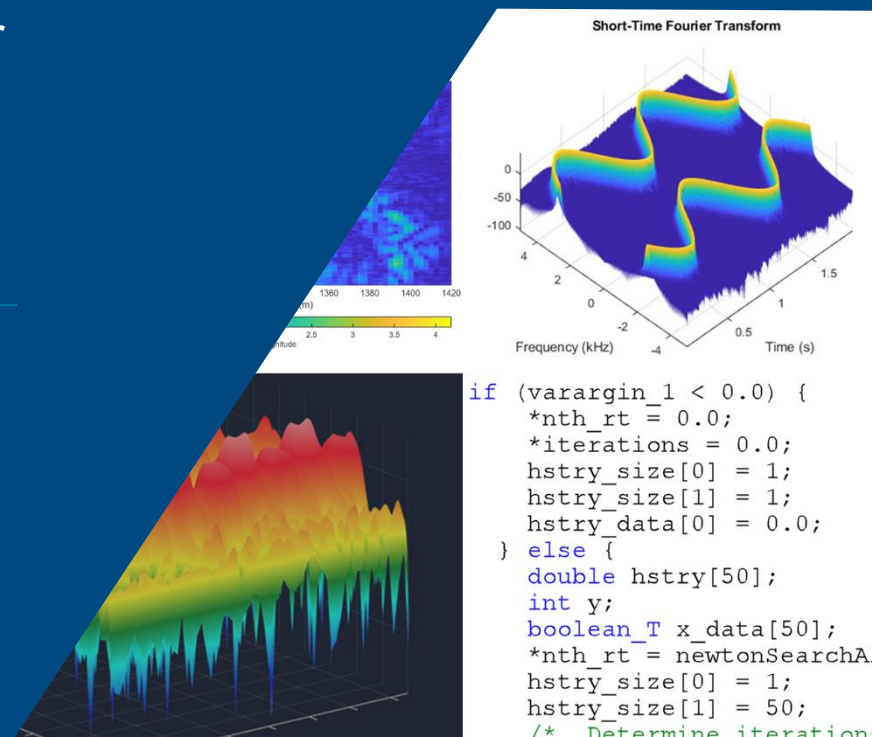




MATLAB to C/C++ Made Easy

Generating **readable** and **portable** C/C++ code from your MATLAB algorithms



```
if (varargin_1 < 0.0) {  
    *nth_rt = 0.0;  
    *iterations = 0.0;  
    hstry_size[0] = 1;  
    hstry_size[1] = 1;  
    hstry_data[0] = 0.0;  
} else {  
    double hstry[50];  
    int y;  
    boolean_T x_data[50];  
    *nth_rt = newtonSearchA  
    hstry_size[0] = 1;  
    hstry_size[1] = 50;  
    /* Determine iteration
```

Agenda

- Motivation
 - Why translate MATLAB to C/C++?
 - Challenges of manual translation
- Using MATLAB Coder
 - Three-step workflow for generating code
- Use cases
 - Integrate algorithms using source code/libraries
 - Accelerate through MEX
 - Prototype by generating EXE
- Conclusion
 - Integration with Simulink, Embedded Coder, and GPU Coder
 - Other deployment solutions

Why Engineers Translate MATLAB to C/C++ Today



.c/cpp

Implement C/C++ code on processors or hand off to software engineers



.lib
.dll

Integrate MATLAB algorithms with existing C/C++ environment using source code and static/dynamic libraries



.exe

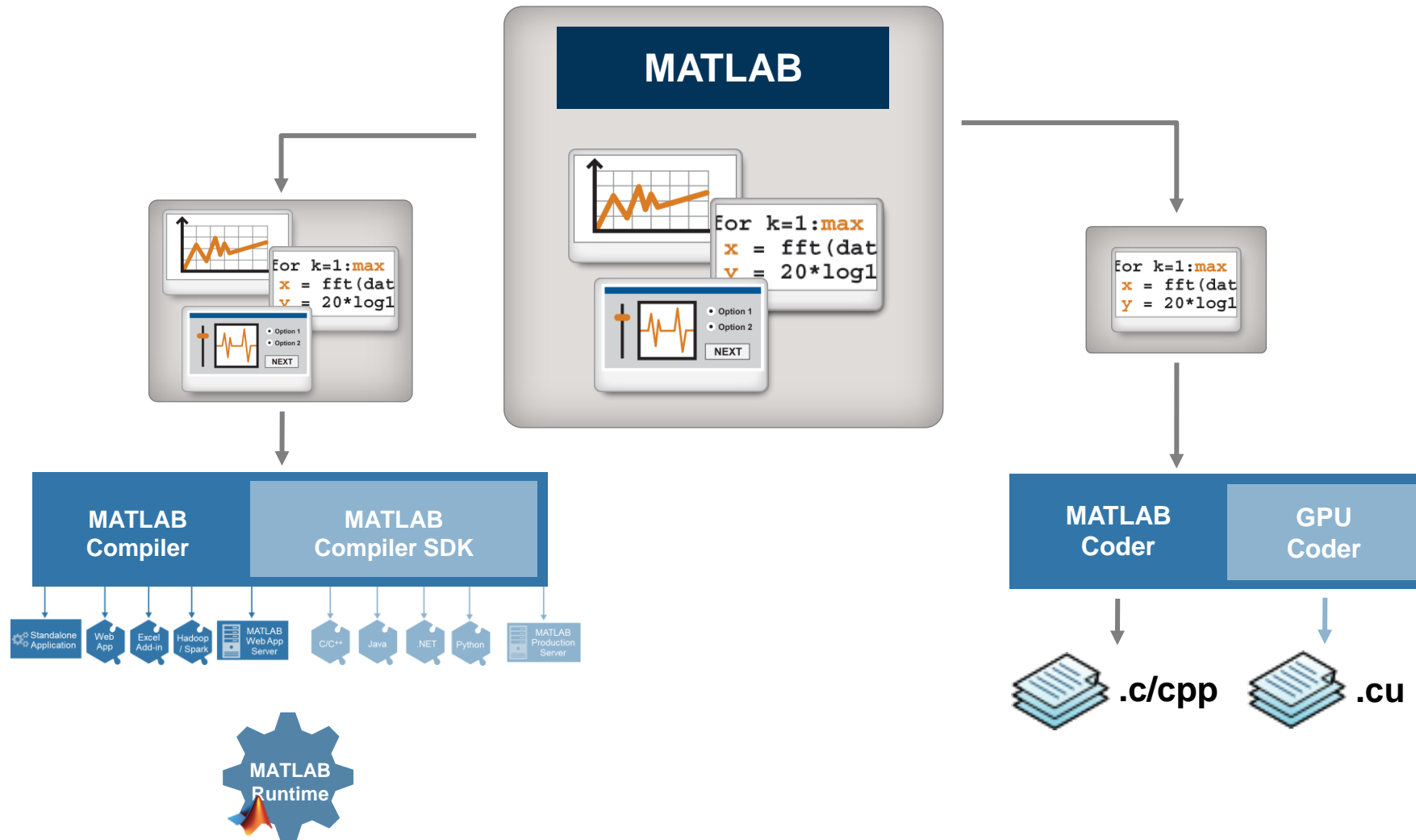
Prototype MATLAB algorithms on desktops as standalone executables



MEX

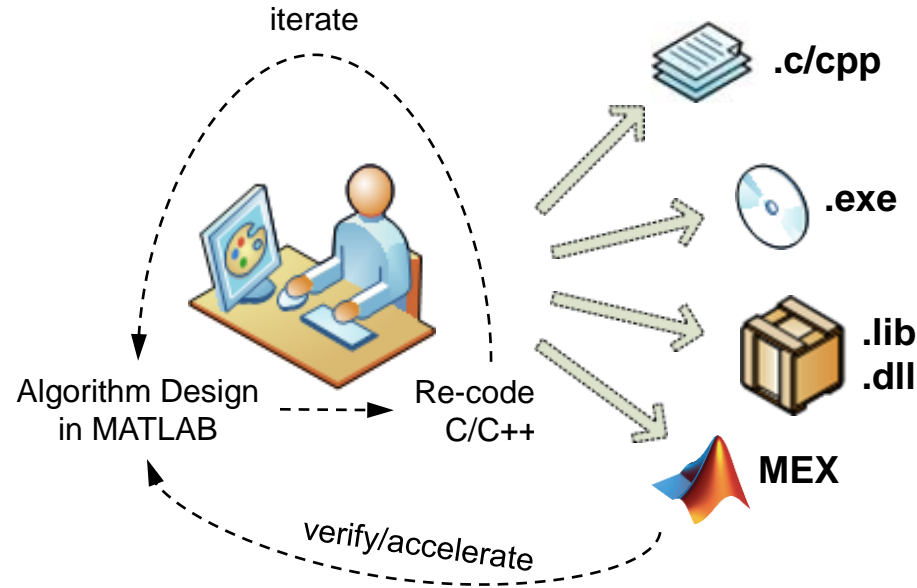
Accelerate user-written MATLAB algorithms

Deploying MATLAB Algorithms



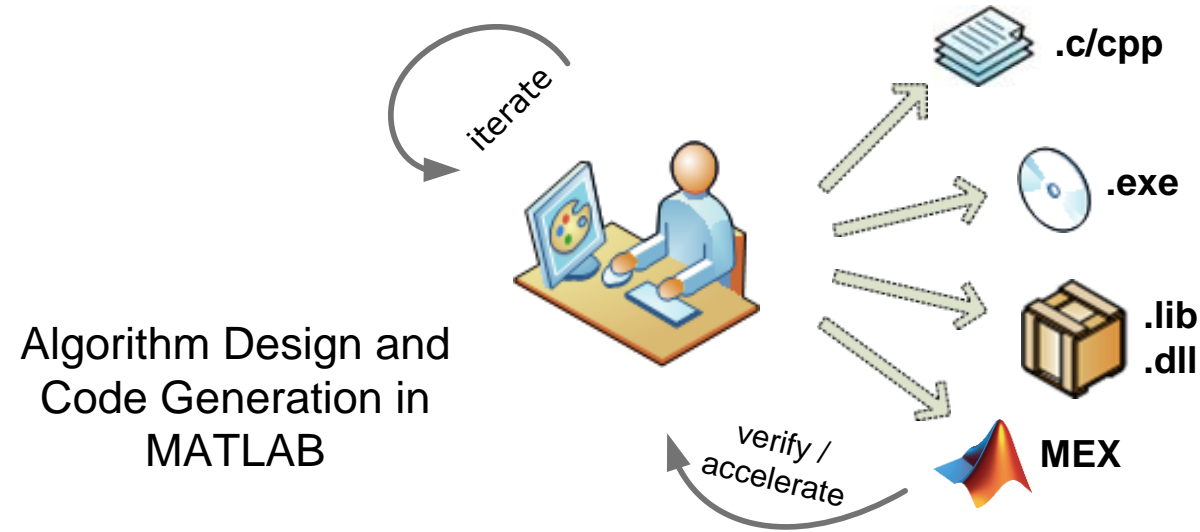
Challenges with Manual Translation

from MATLAB to C/C++



- Separate functional and implementation specification
 - Leads to multiple implementations that are inconsistent
 - Hard to modify requirements during development
 - Difficult to keep reference MATLAB code and C/C++ code in sync
- Manual coding errors
- Time-consuming and expensive process

Automatic Translation of MATLAB to C/C++

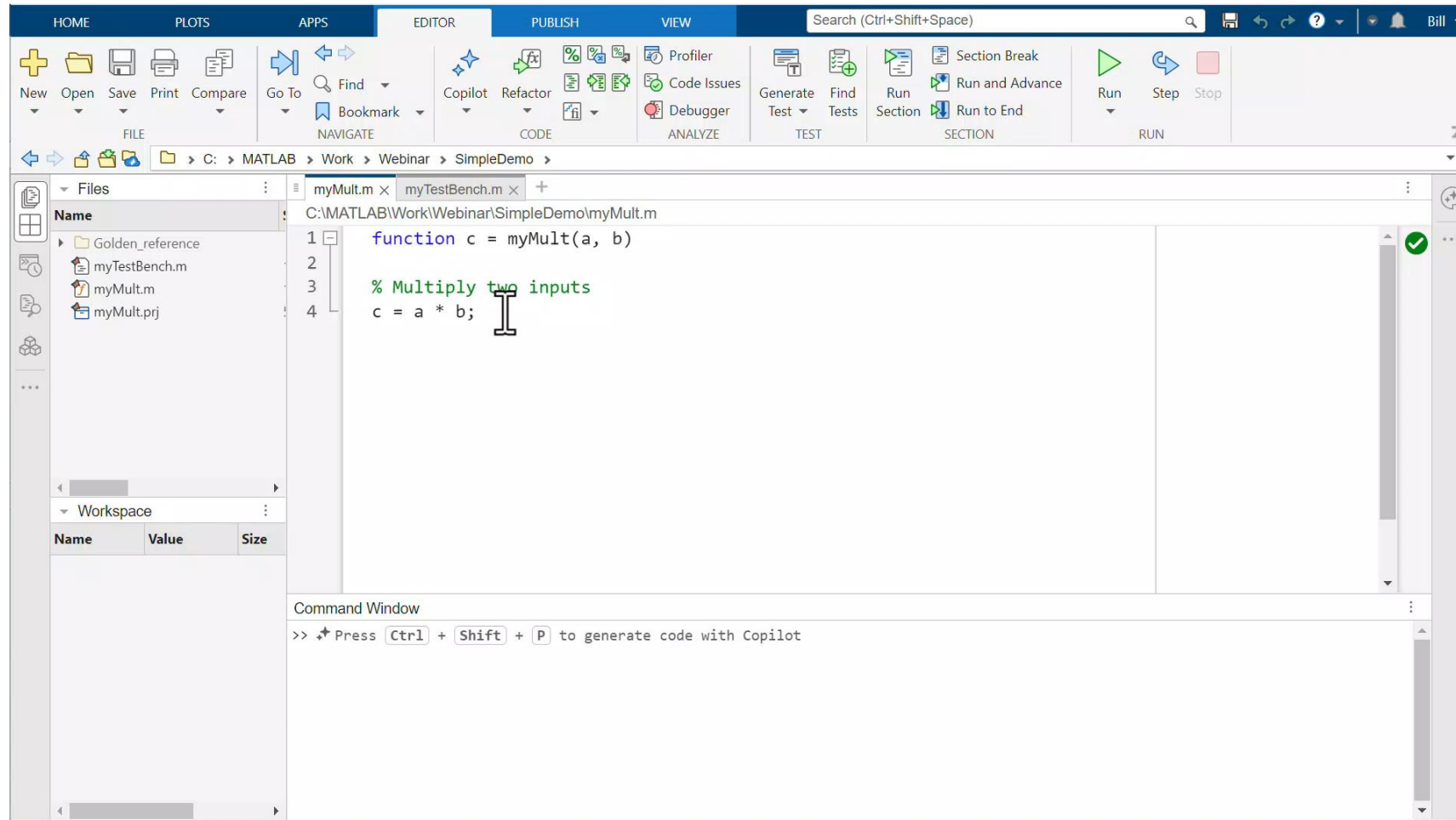


With MATLAB Coder, design engineers can:

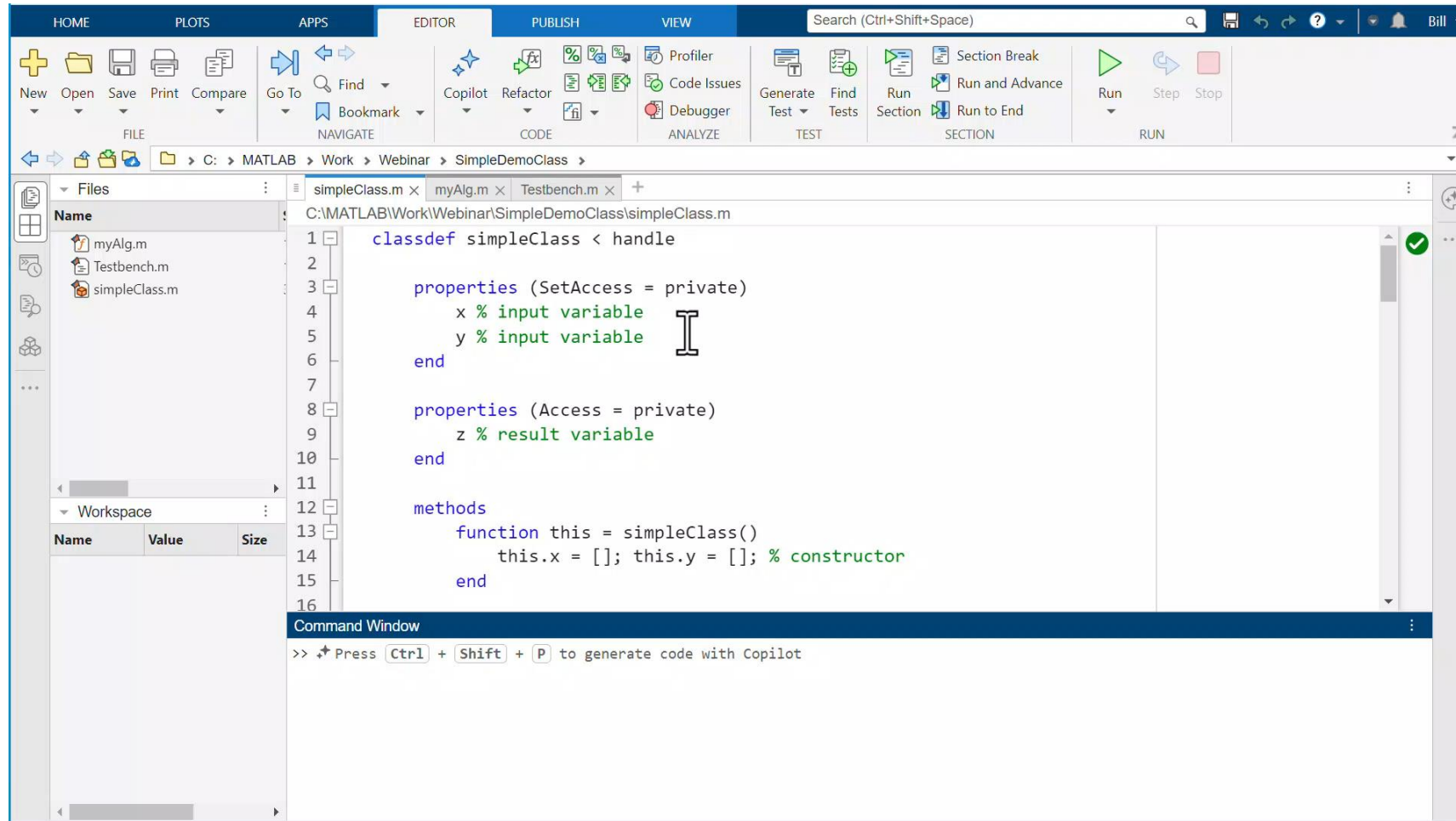
- Maintain one design in MATLAB
- Design faster and get to C/C++ quickly
- Test more systematically and frequently
- Spend more time improving algorithms in MATLAB

Simple Example

$c = a * b$



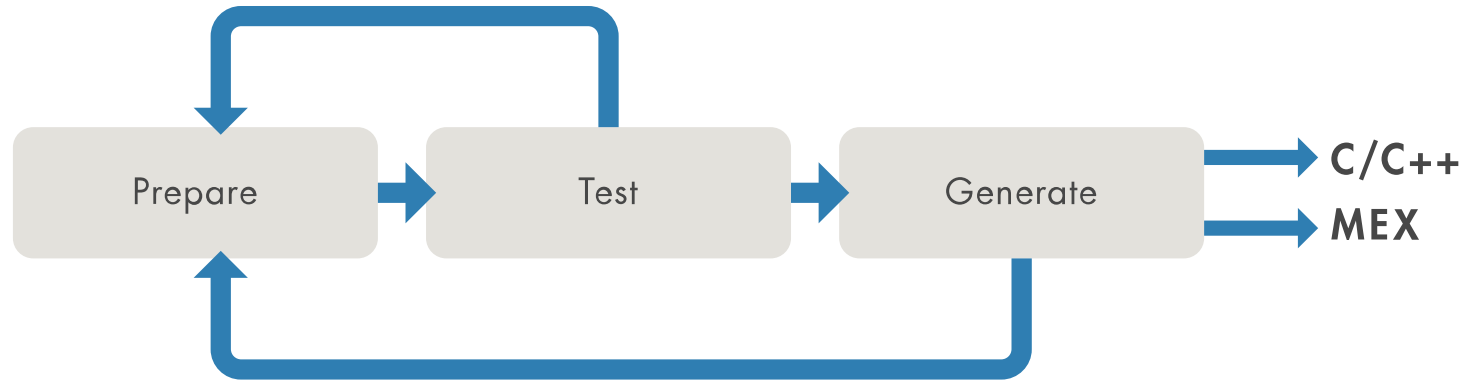
MATLAB Class to C++ Class Example



Agenda

- Motivation
 - Why translate MATLAB to C/C++?
 - Challenges of manual translation
- Using MATLAB Coder
 - Three-step workflow for generating code
- Use cases
 - Integrate algorithms using source code/libraries
 - Accelerate through MEX
 - Prototype by generating EXE
- Conclusion
 - Integration with Simulink, Embedded Coder, and GPU Coder
 - Other deployment solutions

Using MATLAB Coder: Three-Step Workflow



Prepare your MATLAB algorithm for code generation

- Make implementation choices
- Use supported language features

Test if your MATLAB code is ready for code generation

- Validate that MATLAB program generates code
- Accelerate execution of user-written algorithm

Generate source code or MEX for final use

- Iterate your MATLAB code to optimize
- Implement as source, executable, or library

Implementation Considerations

logical
integer
real
complex ...

```
function a= foo(b,c)  
a = b * c;
```

Element by element multiply

Dot product

Matrix multiply

C

```
double foo(double b, double c)  
{  
    return b*c;  
}
```

```
void foo(const double b[15],  
         const double c[30], double a[18])  
{  
    int i0, i1, i2;  
    for (i0 = 0; i0 < 3; i0++) {  
        for (i1 = 0; i1 < 6; i1++) {  
            a[i0 + 3 * i1] = 0.0;  
            for (i2 = 0; i2 < 5; i2++) {  
                a[i0 + 3 * i1] += b[i0 + 3 * i2] * c[i2 + 5 * i1];  
            }  
        }  
    }  
}
```

Implementation Considerations

- Polymorphism
- Memory allocation
- Processing matrices and arrays
- Fixed-point data types

7 Lines of MATLAB
105 Lines of C

```
function [x_est, p_est] = kalman_estimate(R,H,x_prd,p_prd,z)
S = H * p_prd' * H' + R;
B = H * p_prd';
klm_gain = (S \ B)';
x_est = x_prd + klm_gain * (z - H * x_prd);
p_est = p_prd - klm_gain * H * p_prd;
```

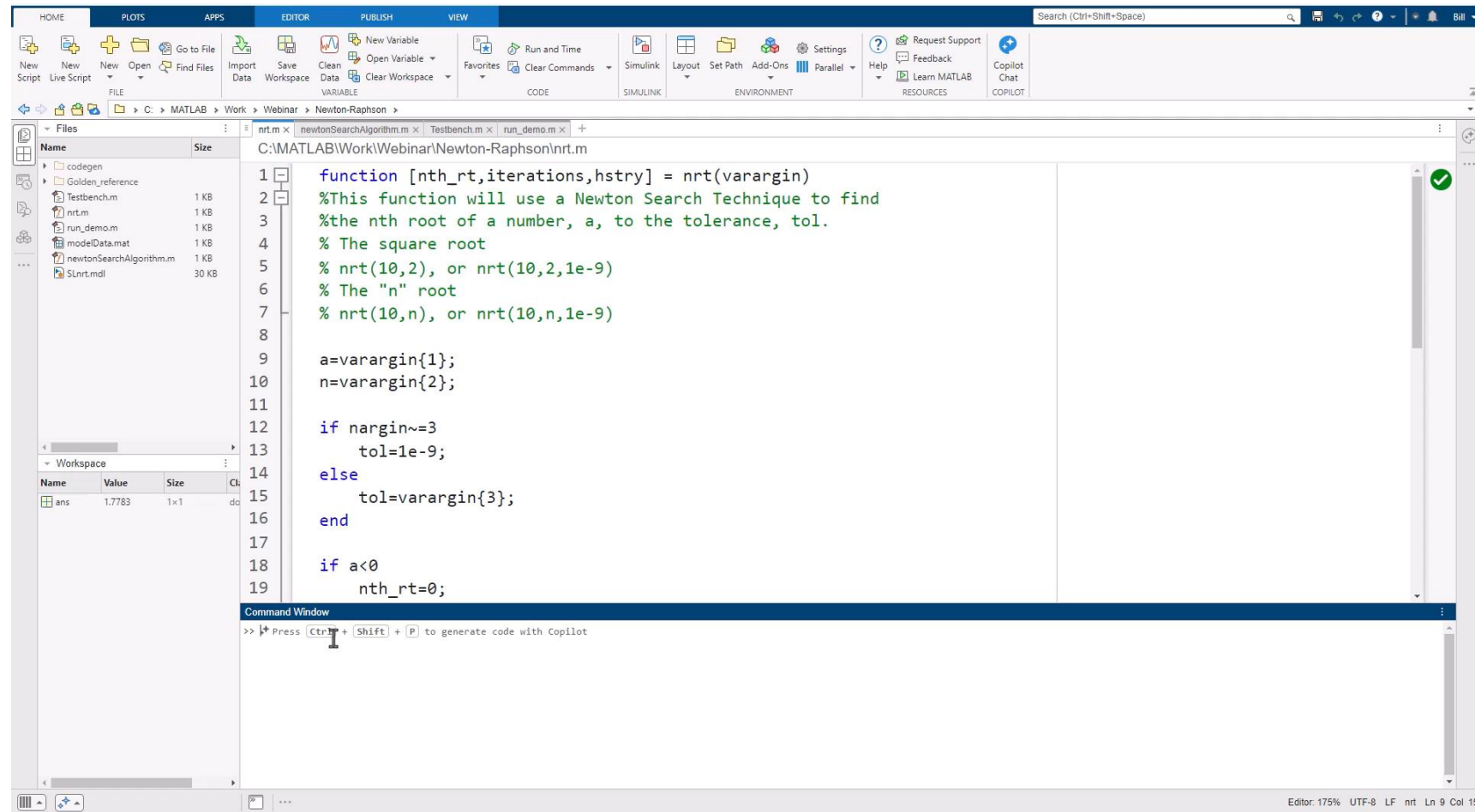
```
#include "kalman_estimate.h"

void kalman_estimate(const double R[4], const double H[2][6],
double p[6], double z[2])
{
    double klm_gain[12];
    int r1;
    int r2;
    int k;
    double S[4];
    double a21;
    double B[12];
    double a22;
    double Y[12];
    double b_z[2];
    double b_klm_gain[36];
    for (r1 = 0; r1 < 2; r1++) {
        for (r2 = 0; r2 < 6; r2++) {
            klm_gain[r1 + (r2 << 1)] = 0.0;
```

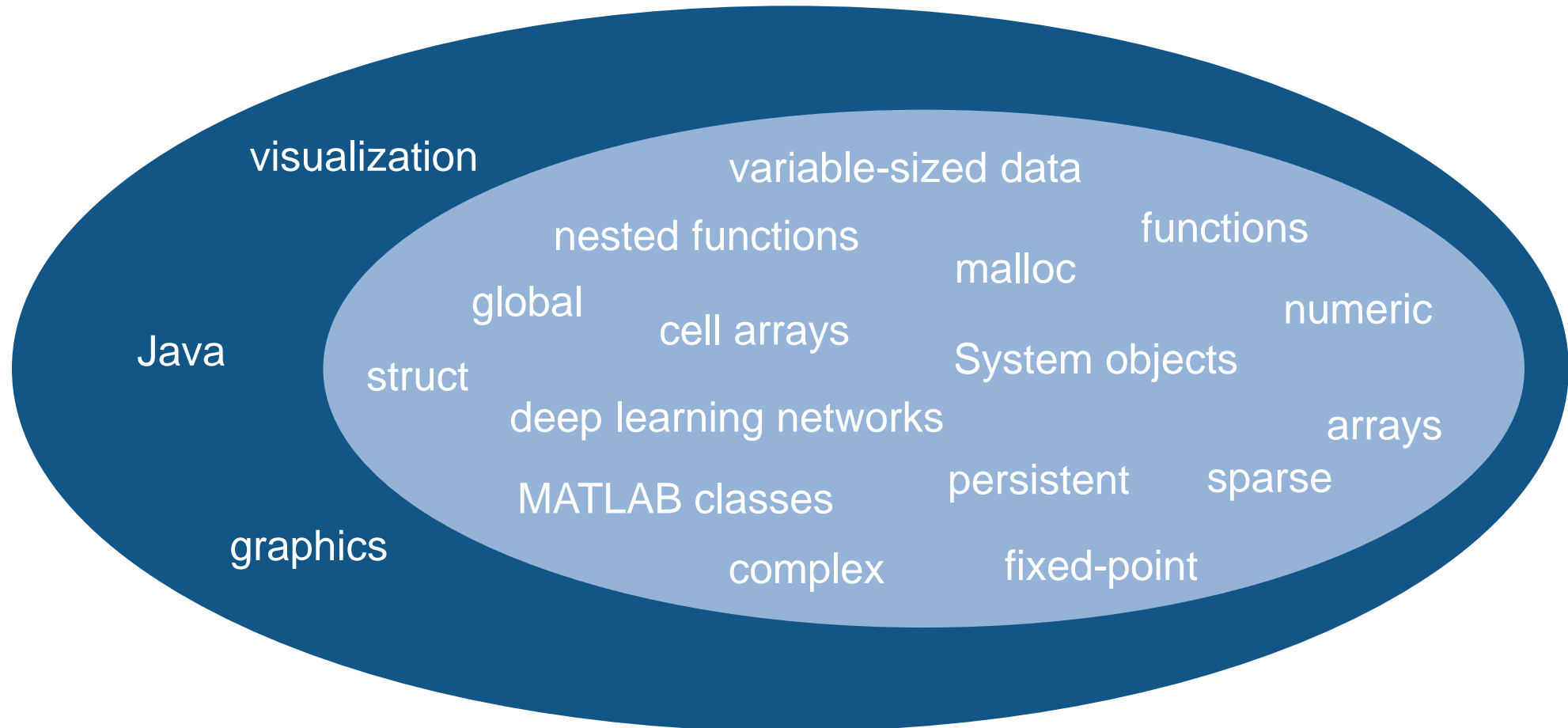
```
        for (k = 0; k < 6; k++) {
            klm_gain[r1 + (r2 << 1)] += H[r1 + (k << 1)] *
        }
    }
    for (r1 = 0; r1 < 2; r1++) {
        for (r2 = 0; r2 < 6; r2++) {
            a21 = 0.0;
            for (k = 0; k < 6; k++) {
                a21 += klm_gain[r1 + (r2 << 1)] * H[r1 + (k << 1)];
            }
            S[r1 + (r2 << 1)] = a21;
        }
    }
    for (r1 = 0; r1 < 2; r1++) {
        for (r2 = 0; r2 < 6; r2++) {
            if (fabs(S[r1 + (r2 << 1)]) > fabs(S[0])) {
                r1 = 1;
                r2 = 0;
            } else {
                r1 = 0;
                r2 = 1;
            }
            a21 = S[r2] / S[r1];
            a22 = S[2 + r2] - a21 * S[2 + r1];
            for (k = 0; k < 6; k++) {
                Y[1 + (k << 1)] = (B[r2 + (k << 1)] - B[r1 + (k << 1)] * a21) / a22;
                Y[k << 1] = (B[r1 + (k << 1)] - Y[1 + (k << 1)] * S[2 + r1]) / S[r1];
            }
            for (r1 = 0; r1 < 2; r1++) {
                for (r2 = 0; r2 < 6; r2++) {
                    klm_gain[r2 + 6 * r1] = Y[r1 + (r2 << 1)];
                }
            }
        }
    }
```

```
        for (r1 = 0; r1 < 6; r1++) {
            for (r2 = 0; r2 < 6; r2++) {
                b_klm_gain[r1 + 6 * r2] = 0.0;
                for (k = 0; k < 2; k++) {
                    b_klm_gain[r1 + 6 * r2] += klm_gain[r1 + 6 * k] * H[k + (r2 << 1)];
                }
            }
        }
        for (r1 = 0; r1 < 6; r1++) {
            for (r2 = 0; r2 < 6; r2++) {
                a21 = 0.0;
                for (k = 0; k < 6; k++) {
                    a21 += b_klm_gain[r1 + 6 * k] * p_prd[k + 6 * r2];
                }
                p_est[r1 + 6 * r2] = p_prd[r1 + 6 * r2] - a21;
            }
        }
    }
```

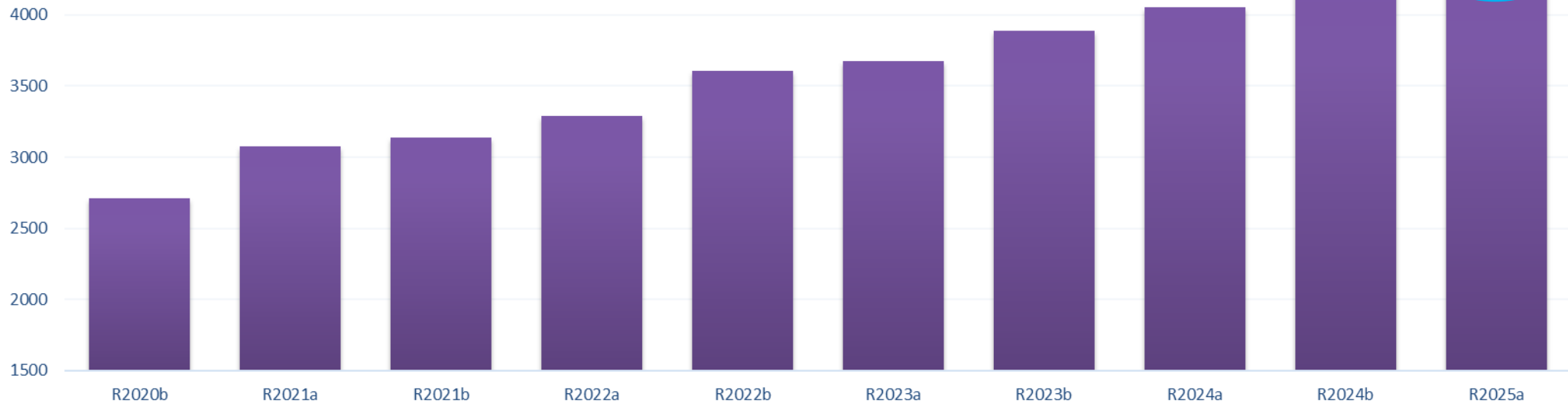
Newton/Raphson Example



Growing MATLAB Language Support for Code Generation



4,200 Functions & 41 Toolboxes Supported



- 5G Toolbox
- Aerospace Toolbox
- Antenna Toolbox
- Audio Toolbox
- Automated Driving Toolbox
- Bluetooth Toolbox
- Communications Toolbox
- Computer Vision Toolbox
- Control System Toolbox
- Deep Learning Toolbox
- DSP HDL Toolbox
- DSP System Toolbox
- Financial Toolbox
- Fixed-Point Designer
- Fuzzy Logic Toolbox
- Image Acquisition Toolbox
- Image Processing Toolbox
- Industrial Communication Toolbox
- Lidar Toolbox
- LTE Toolbox
- Mapping Toolbox
- Medical Imaging Toolbox
- Model Predictive Control Toolbox
- Navigation Toolbox
- Optimization Toolbox
- Phased Array System Toolbox
- Predictive Maintenance Toolbox
- Radar Toolbox
- Reinforcement Learning Toolbox
- Robotics System Toolbox
- ROS Toolbox
- Satellite Communications Toolbox
- Sensor Fusion and Tracking Toolbox
- SerDes Toolbox
- Signal Processing Toolbox
- Statistics and Machine Learning Toolbox
- UAV Toolbox
- Vision HDL Toolbox
- Wavelet Toolbox
- Wireless Testbench
- WLAN Toolbox

Agenda

- Motivation
 - Why translate MATLAB to C/C++?
 - Challenges of manual translation
- Using MATLAB Coder
 - Three-step workflow for generating code
- Use cases
 - Integrate algorithms using source code/libraries
 - Accelerate through MEX
 - Prototype by generating EXE
- Conclusion
 - Integration with Simulink, Embedded Coder, and GPU Coder
 - Other deployment solutions

MATLAB Coder Use Cases



.lib
.dll

Integrate
algorithms with custom software



.exe

Prototype
algorithms on PCs

Accelerate
algorithm execution

Implement
algorithms on embedded processors



MEX

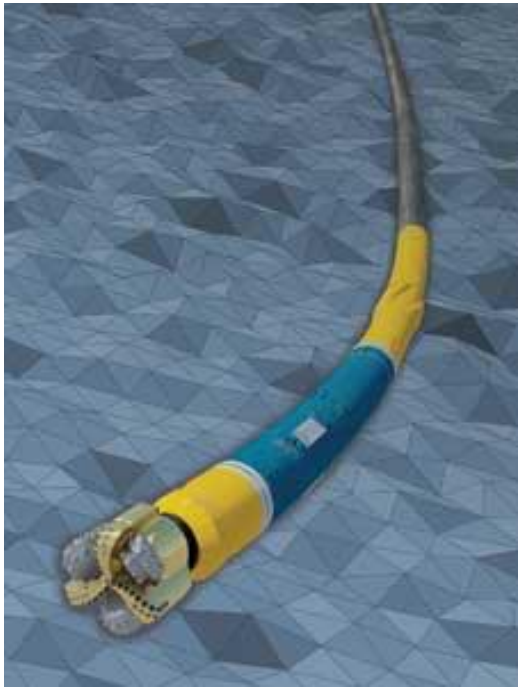


.c/cpp

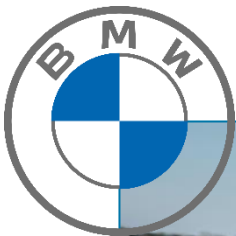
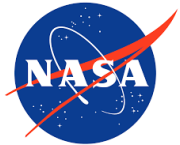
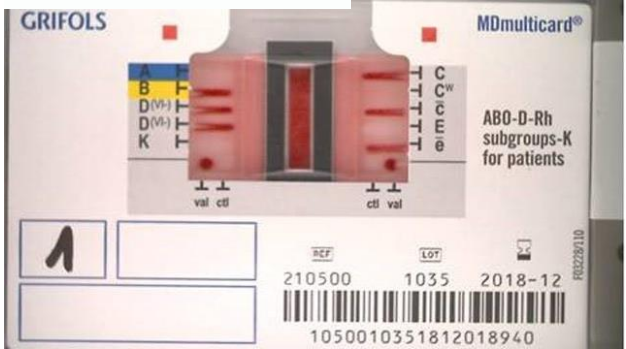
Examples of MATLAB Coder Usage

Integrate
algorithms with custom software

Implement
algorithms on embedded processors



Qualcomm



**BAKER
HUGHES**



dorsaVi
Inspiring the world to move

Coca-Cola
freestyle®



Deep Learning on Jetson (ARM-only) Example

Implement
algorithms on embedded processors

The image shows the MATLAB Live Editor interface. The top toolbar includes tabs for HOME, PLOTS, APPS, MATLAB CODER, LIVE EDITOR, INSERT, and VIEW. Below the toolbar is a ribbon with various icons for file operations, navigation, text editing, code generation, and running. The main workspace is divided into three panes: Files, Workspace, and Command Window.

Files Pane: Shows a directory structure with files like `codegen`, `images`, `getTargetDir.m`, `cnn_predict.m`, `ocv2mat.m`, `mat2ocv.m`, `myNDNet_Postprocess.m`, `targetFunction.m`, `targetFunction_CPU.m`, and `myNDNet_Preprocess.m`.

Workspace Pane: Shows variables in the workspace, including `ans`, `argsforexe`, `cfg`, `ch`, `command`, `convnet`, `exeName`, `GPUWor...`, and `he`.

Command Window: Displays the command `>> Press Ctrl + Shift + P to generate code with Copilot`.

Code Editor: The main pane shows the MATLAB code for `targetFunction.m`. The code is titled "Generate Code for Deep Learning Network for Classification and Run Algorithms on the Jetson (ARM Core Only)". The code includes comments and function definitions for `targetFunction`.

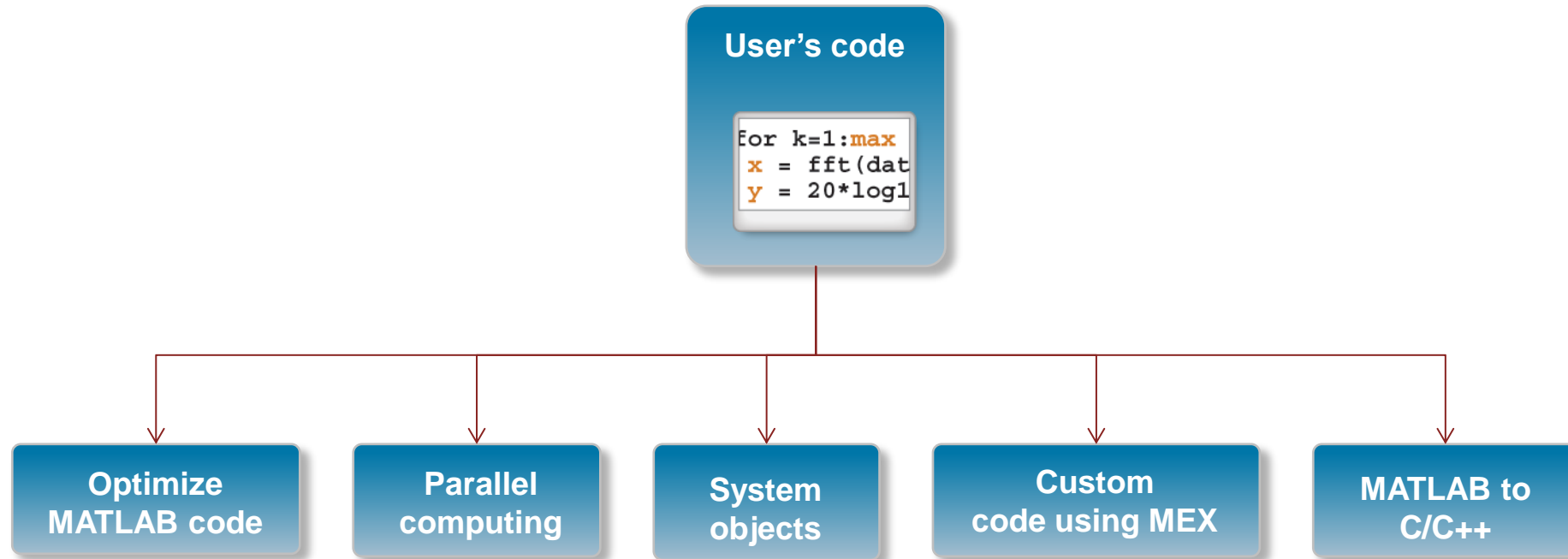
MATLAB CODER Pane: Shows the "Next Steps" section with a button "Run using MEX". Below this is the "INPUT" section with a variable `targetFunction (img, Weig...` and a "Success" message indicating that code generation was successful. The "OUTPUT" section shows the generated source files and the project file `targetFunction.cod...`.

Bottom Status Bar: Shows the editor settings: Editor: 125% UTF-8 LF Script.

Acceleration Strategies

- Better algorithms
Matrix inversion vs. QR or SVD
 - Different approaches to solving the same problem
- More efficient implementation
Hand-coded vs. optimized library (BLAS and LAPACK)
 - Different optimization of the same algorithm
- More computational resources
Single-threaded vs. multithreaded (multithreaded BLAS)
 - Leveraging additional processors, cores, GPUs, FPGAs, etc.

Accelerating Algorithm Execution



Acceleration Using MEX

- Speed-up factor will vary
- When you **may** see a speedup:
 - Often for communications and signal processing
 - Always for fixed point
 - Likely for loops with states or when vectorization isn't possible
- When you **may not** see a speedup:
 - MATLAB implicitly multithreads computation.
 - Built-functions call IPP or BLAS libraries.

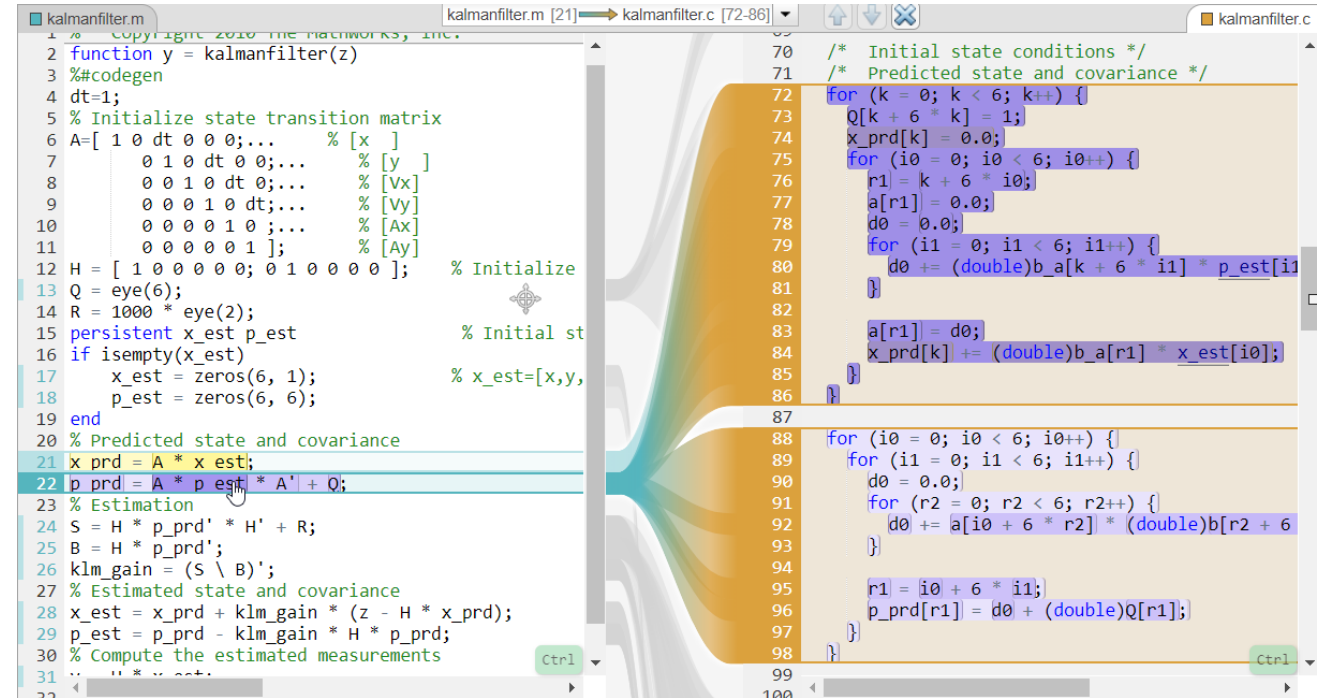
Agenda

- Motivation
 - Why translate MATLAB to C/C++?
 - Challenges of manual translation
- Using MATLAB Coder
 - Three-step workflow for generating code
- Use cases
 - Integrate algorithms using source code/libraries
 - Accelerate through MEX
 - Prototype by generating EXE
- Conclusion
 - Integration with Simulink, Embedded Coder, and GPU Coder
 - Other deployment solutions

Working with Embedded Coder

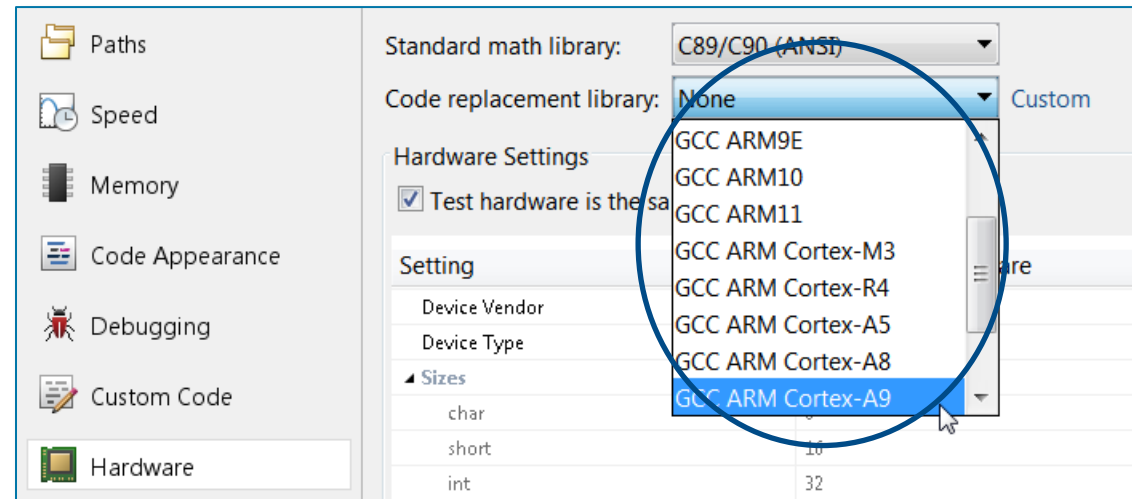
Advanced support for MATLAB Coder, including:

- Speed & Memory
- Code appearance
- Bidirectional traceability
- Hardware-specific optimization
- Software/Processor-in-the-loop verification
- Execution profiling

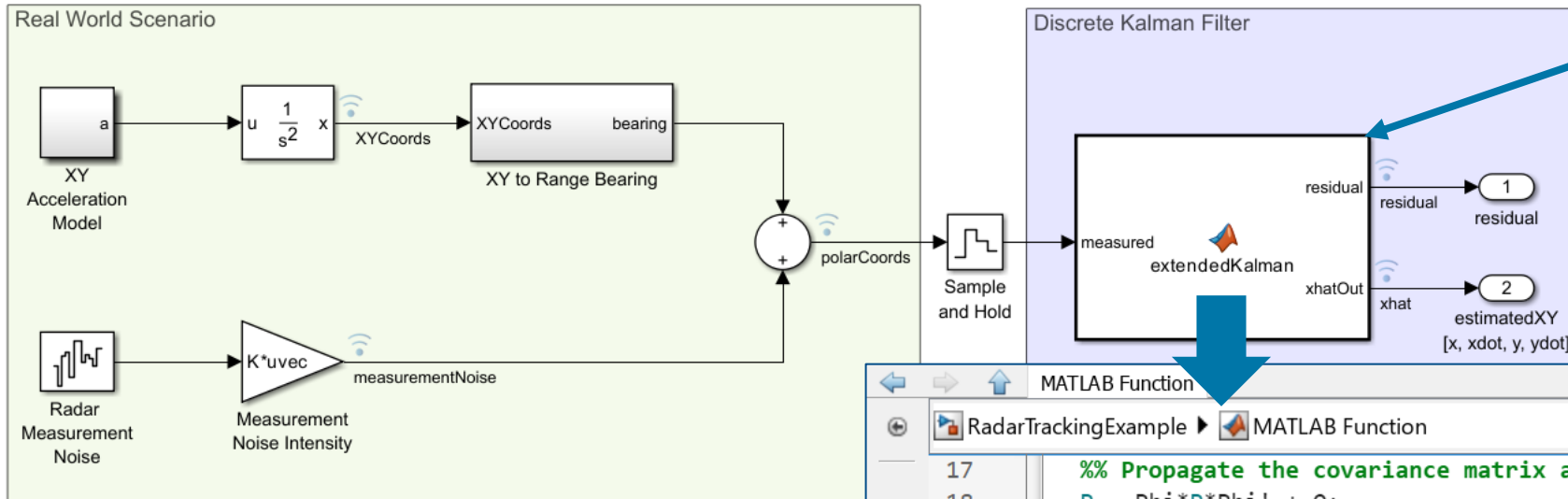


```
1 % Copyright 2010 The MathWorks, Inc.
2 function y = kalmanfilter(z)
3 %codegen
4 dt=1;
5 % Initialize state transition matrix
6 A=[ 1 0 dt 0 0 0;... % [x ]
7     0 1 0 dt 0 0;... % [y ]
8     0 0 1 0 dt 0;... % [vx]
9     0 0 0 1 0 dt;... % [vy]
10    0 0 0 0 1 0;... % [Ax]
11    0 0 0 0 0 1]; % [Ay]
12 H = [ 1 0 0 0 0 0; 0 1 0 0 0 0 ]; % Initialize
13 Q = eye(6);
14 R = 1000 * eye(2);
15 persistent x_est p_est % Initial st
16 if isempty(x_est)
17     x_est = zeros(6, 1); % x_est=[x,y,
18     p_est = zeros(6, 6);
19 end
20 % Predicted state and covariance
21 x_prd = A * x_est;
22 p_prd = A * p_est * A' + Q;
23 % Estimation
24 S = H * p_prd' * H' + R;
25 B = H * p_prd';
26 klm_gain = (S \ B)';
27 % Estimated state and covariance
28 x_est = x_prd + klm_gain * (z - H * x_prd);
29 p_est = p_prd - klm_gain * H * p_prd;
30 % Compute the estimated measurements
31 ...
```

```
70 /* Initial state conditions */
71 /* Predicted state and covariance */
72 for (k = 0; k < 6; k++) {
73     Q[k + 6 * k] = 1;
74     x_prd[k] = 0.0;
75     for (i0 = 0; i0 < 6; i0++) {
76         r1 = k + 6 * i0;
77         a[r1] = 0.0;
78         d0 = 0.0;
79         for (i1 = 0; i1 < 6; i1++) {
80             d0 += (double)b_a[k + 6 * i1] * p_est[i1
81         ]
82     }
83     a[r1] = d0;
84     x_prd[k] += (double)b_a[r1] * x_est[i0];
85 }
86
87
88 for (i0 = 0; i0 < 6; i0++) {
89     for (i1 = 0; i1 < 6; i1++) {
90         d0 = 0.0;
91         for (r2 = 0; r2 < 6; r2++) {
92             d0 += a[i0 + 6 * r2] * (double)b[r2 + 6
93         ]
94     }
95     r1 = i0 + 6 * i1;
96     p_prd[r1] = d0 + (double)Q[r1];
97 }
98
99
100
```



Working with Simulink and Embedded Coder



MATLAB Function block in Simulink

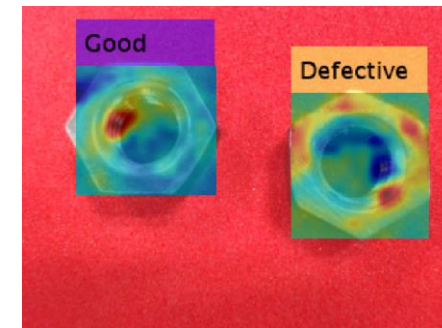
```
MATLAB Function

17 %% Propagate the covariance matrix and track estimate
18 P = Phi*P*Phi' + Q;
19 xhat = Phi*xhat;
20
21 %% Compute observation estimates:
22 Rangehat = sqrt(xhat(1)^2+xhat(3)^2);
23 Bearinghat = atan2(xhat(3),xhat(1));
24
25 % Compute observation vector y and linearized measurement matrix M
26 yhat = [Rangehat;
27         Bearinghat];
28 M = [ cos(Bearinghat)      0 sin(Bearinghat)      0
29       -sin(Bearinghat)/Rangehat 0 cos(Bearinghat)/Rangehat 0 ];
30
31 %% Compute residual (Estimation Error)
32 residual = measured - yhat;
33
34 % Compute Kalman Gain:
35 K = P*M'/(M*P*M' + R);
```

Working with GPU Coder

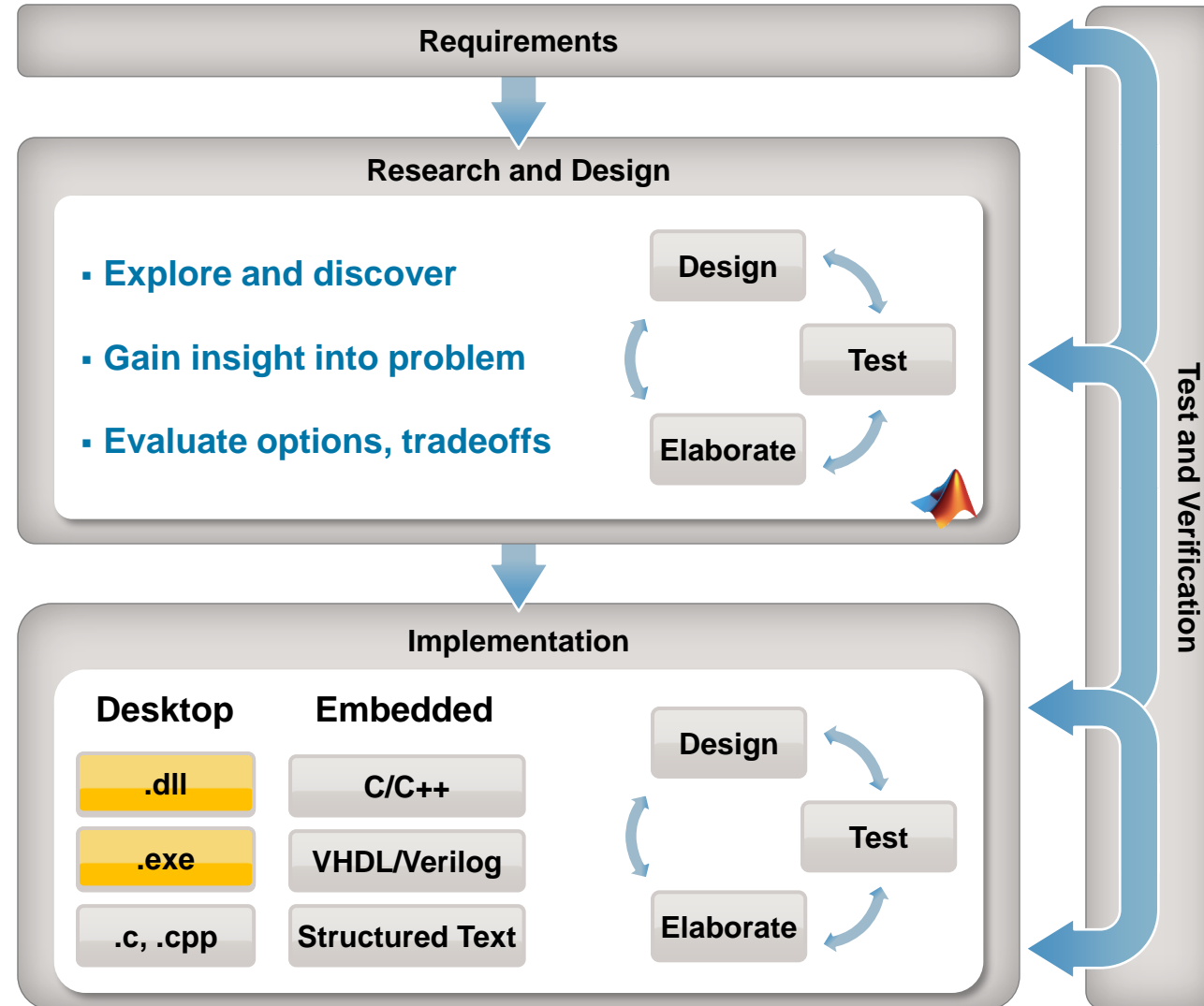
Generate CUDA for NVIDIA GPUs

- Deploy deep learning applications, include pre/post-processing
- Create CUDA kernels from MATLAB algorithms for acceleration on GPUs
- Automated deployment to NVIDIA GPUs, including Jetson/DRIVE

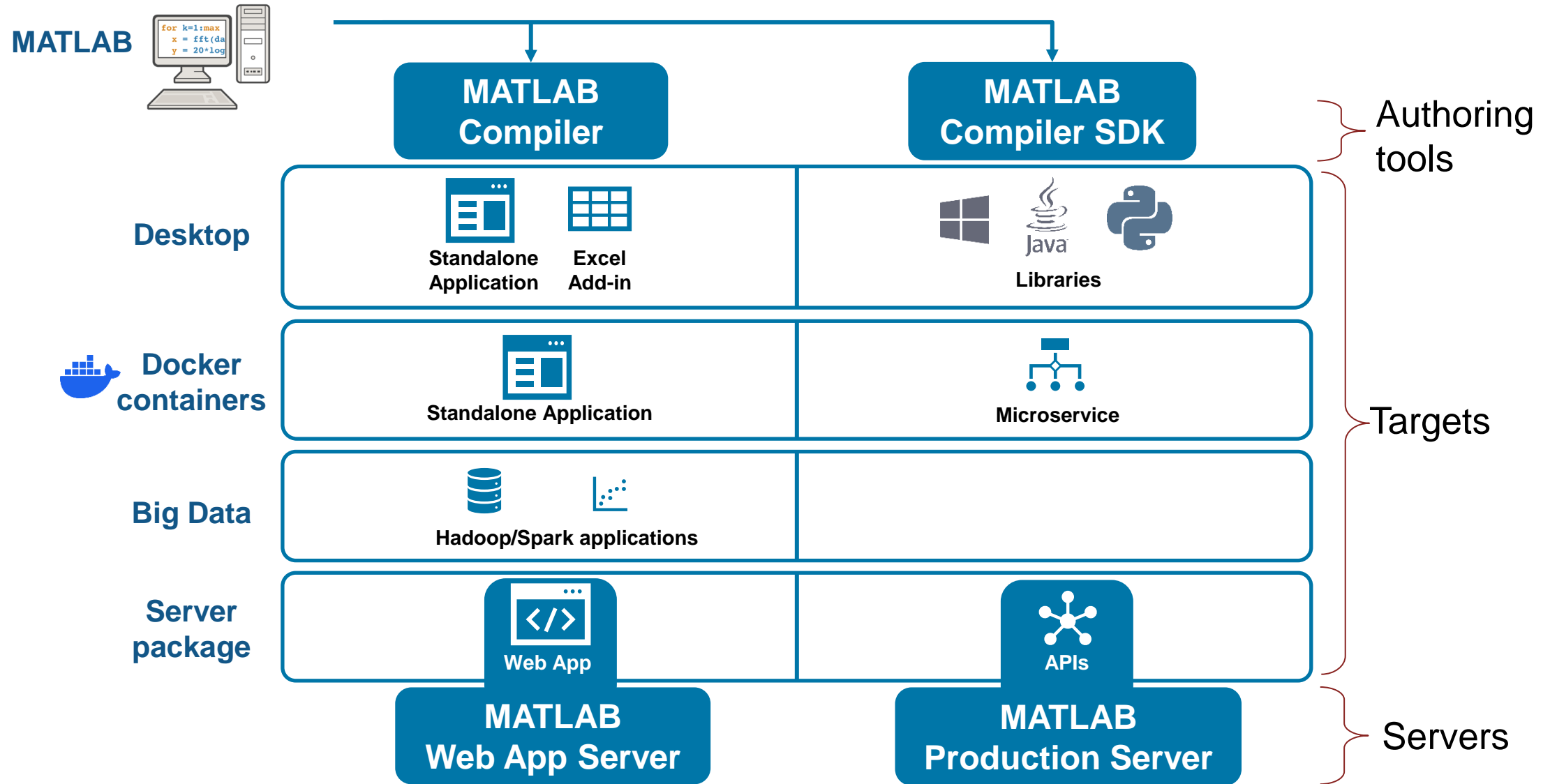


Jetson Orin	
MATLAB Coder	GPU Coder
119s	12s

Other Desktop Deployment Options



Other Deployment Options



Choosing the Right Deployment Solution



.c/cpp

MATLAB Coder



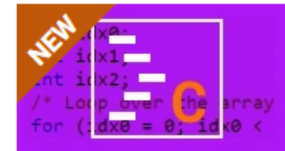
**MATLAB Compiler
MATLAB Compiler SDK**

Output	Portable and readable C/C++ source code	Executable or software component/shared library
Main Use Case	Deploy MATLAB code as portable C/C++ code on embedded platforms or desktop	Deploy MATLAB programs as standalone applications on desktop or production servers
MATLAB language support	Subset	Full
Supported toolboxes	Some toolboxes	Most toolboxes
Production	Embedded Coder	MATLAB Production Server
Graphics Support	None	Full
Library Dependency	None	MATLAB Runtime

[More details](#)

More Information

- [Product page and request trial license](#)
- [MATLAB Coder Onramp](#) (self-paced)
- [Quick Start Guide for MATLAB Coder](#)
- [Article: The Joy of Generating C Code from MATLAB](#)
- [MATLAB to C with MATLAB Coder](#) (instructor-led training)



MATLAB Coder Onramp

1 hour | [Languages](#)

Learn the basics of C code generation from MATLAB functions.

Free Quick Start Guide for MATLAB Coder

Learn best practices for generating standalone ANSI-C source code and MEX-files from your MATLAB algorithm using MATLAB Coder.



This downloadable guide includes modules that show you how to:

- ✓ Prepare MATLAB code for MATLAB Coder
- ✓ Accelerate your MATLAB algorithms by generating MEX-files using MATLAB Coder
- ✓ Generate standalone ANSI-C code with MATLAB Coder

Each module includes best practices, tips and tricks, and FAQs.

MATLAB and Simulink Training

[Training Overview](#) | [Find a Course](#) | [My Courses](#) | [Get Certified](#) | [Why MathWorks Training?](#) | [Professional Education](#) | [More](#)

MATLAB to C with MATLAB Coder

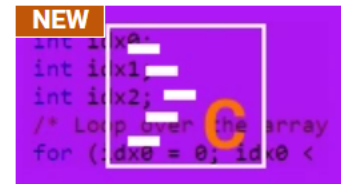
[View schedule and enroll](#)

MATLAB Coder Onramp

Free, online self-paced course that covers the basics of generating C code from MATLAB functions

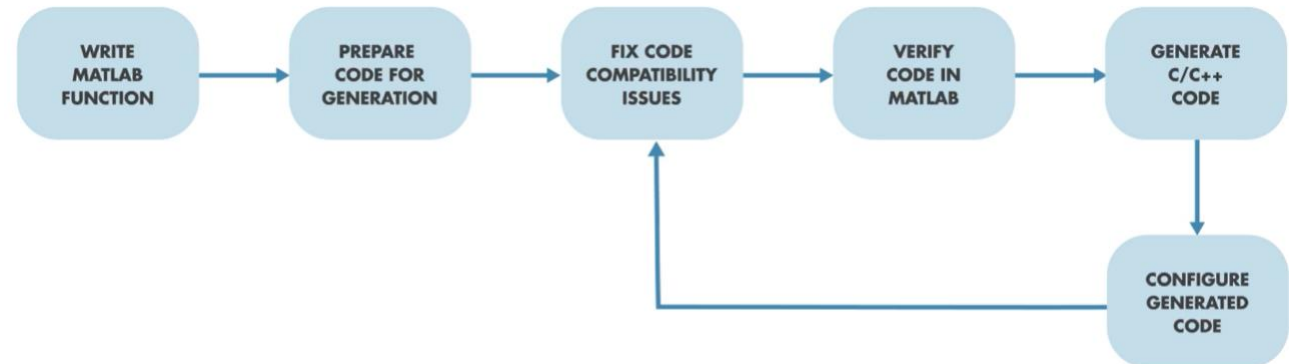
Topics include

- Generate MEX and C Code using programmatic workflows
- View C code in code generation report
- Fix common issues that prevent code generation
- Call MEX functions from MATLAB
- Allow variable-size function inputs
- Modify code generation settings



MATLAB Coder Onramp

Start course



>> [Access the Onramp](#)

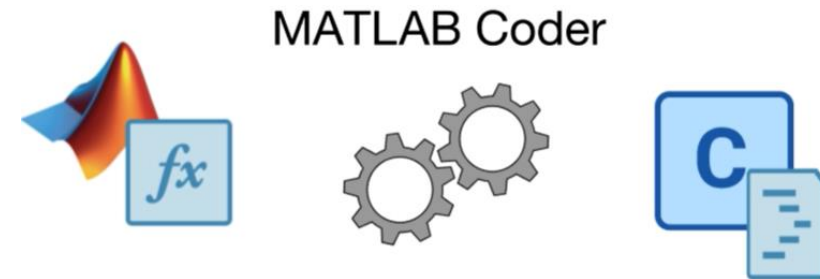
MATLAB to C with MATLAB Coder Training

Instructor-led two-day in-person training with options for customization

Topics include

- Using MATLAB Coder from the app and automating with scripts
- Calling unsupported MATLAB functions
- Working with fixed-size data and variable-size data
- Integrating with External Code
- Optimizing Generated Code
- Customized training tailored to your needs

» [Learn more about the training class here](#)



MATLAB Coder

Level: Advanced

Prerequisites:

- [MATLAB Fundamentals](#) and knowledge of C programming language

Duration: 2 days

Languages: English, 한국어

Q&A