Automatically Convert MATLAB code to C code

Generate readable and portable C code
from your MATLAB algorithms

Daryl Ning
Applications Engineer

MathWorks Australia
Level 5, Tower 1
495 Victoria Ave
CHATSWOOD NSW 2067

Example: Euclidean distance measure

euclidean.m

```matlab
function [y,idx,distance] = euclidean(x,codebook) %#codegen
% Initialize minimum distance as first element of codebook
idx=1;
distance=norm(x-codebook(:,1));

% Find the vector in codebook with minimum distance to x
for index=2:size(codebook,2)
    d=norm(x-codebook(:,index));
    if d < distance
        distance=d;
        idx=index;
    end
end

% Output the minimum distance vector
y=codebook(:,idx);
```

MATLAB Coder

```c
void euclidean(const float x[20], const float codebook[2000], float y[20],
               double *idx, float *distance)
{
  float b_x[20];
  int i0;
  int b_index;
  float d;
  *idx = 1.0;
  for (i0 = 0; i0 < 20; i0++) {
    b_x[i0] = x[i0] - codebook[i0];
  }

  *distance = norm(b_x);

  /*  Find the vector in codebook with minimum distance to x */
  for (b_index = 0; b_index < 99; b_index++) {
    for (i0 = 0; i0 < 20; i0++) {
      b_x[i0] = x[i0] - codebook[i0 + 20 * (b_index + 1)];
    }

    d = norm(b_x);
    if (d < *distance) {
      *distance = d;
      *idx = 2.0 + (double)b_index;
    }
  }

  /*  Output the minimum distance vector */
  memcpy(&y[0], &codebook[20 * ((int)*idx - 1)], 20U * sizeof(float));
}
```

iSonea Develops Mobile App and Server Software for Wheeze Detection and Asthma Management

The AirSonea device, which connects to an asthma patient's smartphone and communicates with wheeze analysis algorithms on iSonea's server.

Challenge

Develop and implement an acoustic respiratory monitoring system for wheeze detection and asthma management

Solution

Develop algorithms for detecting wheeze and ambient noise in MATLAB, and use MATLAB Coder to generate code from the algorithms for mobile devices and a web server

Results

- Manual coding effort reduced
- Algorithm development iterations accelerated
- Code maintenance overhead reduced

"MATLAB enables us to rapidly develop, debug, and test sound-processing algorithms, and MATLAB Coder simplifies the process of implementing those algorithms in C. There's no other environment or programming language that we could use to produce similar results in the same amount of time."

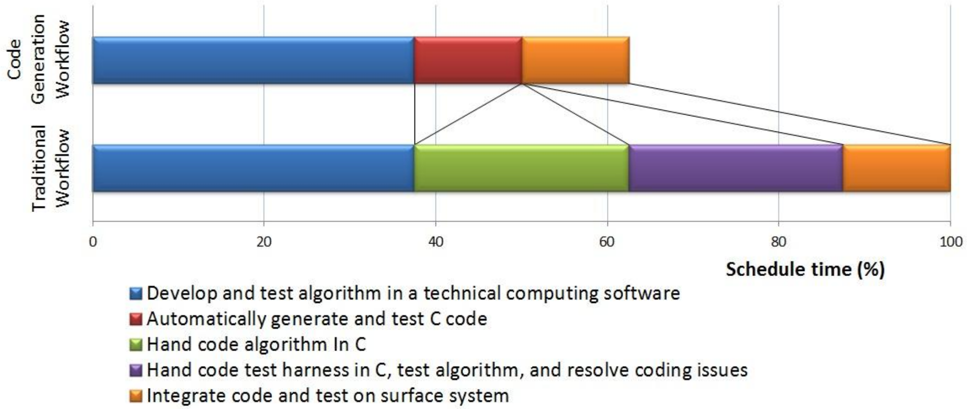
Mark Mulvey
iSonea

[Link to user story]

Baker Hughes
Oilfield Services Company

- Deployed a real time algorithm that optimizes the drilling process and lowers the cost of operations

- "*This workflow shortened the development process by eliminating the need for maintaining and testing the same algorithm in two languages.* " Dr. Christian Hansen, Baker Hughes



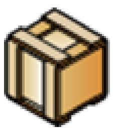
- http://www.edn.com/design/systems-design/4421993/1/Reducing-risk-in-implementing-technical-computing-algorithms

Agenda

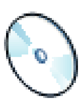
- Motivation
  - Why translate MATLAB to C?
  - Challenges of manual translation
- Using MATLAB Coder
  - Three-step workflow for generating code
- Use cases
  - Integrate algorithms with external C code
  - Accelerate through MEX
  - Prototype by generating EXE
  - Integration with Simulink and Embedded Coder
  - Other deployment solutions
- Summary

Why Engineers Translate MATLAB to C

.c — **Implement** C code on processors or hand off to software engineers



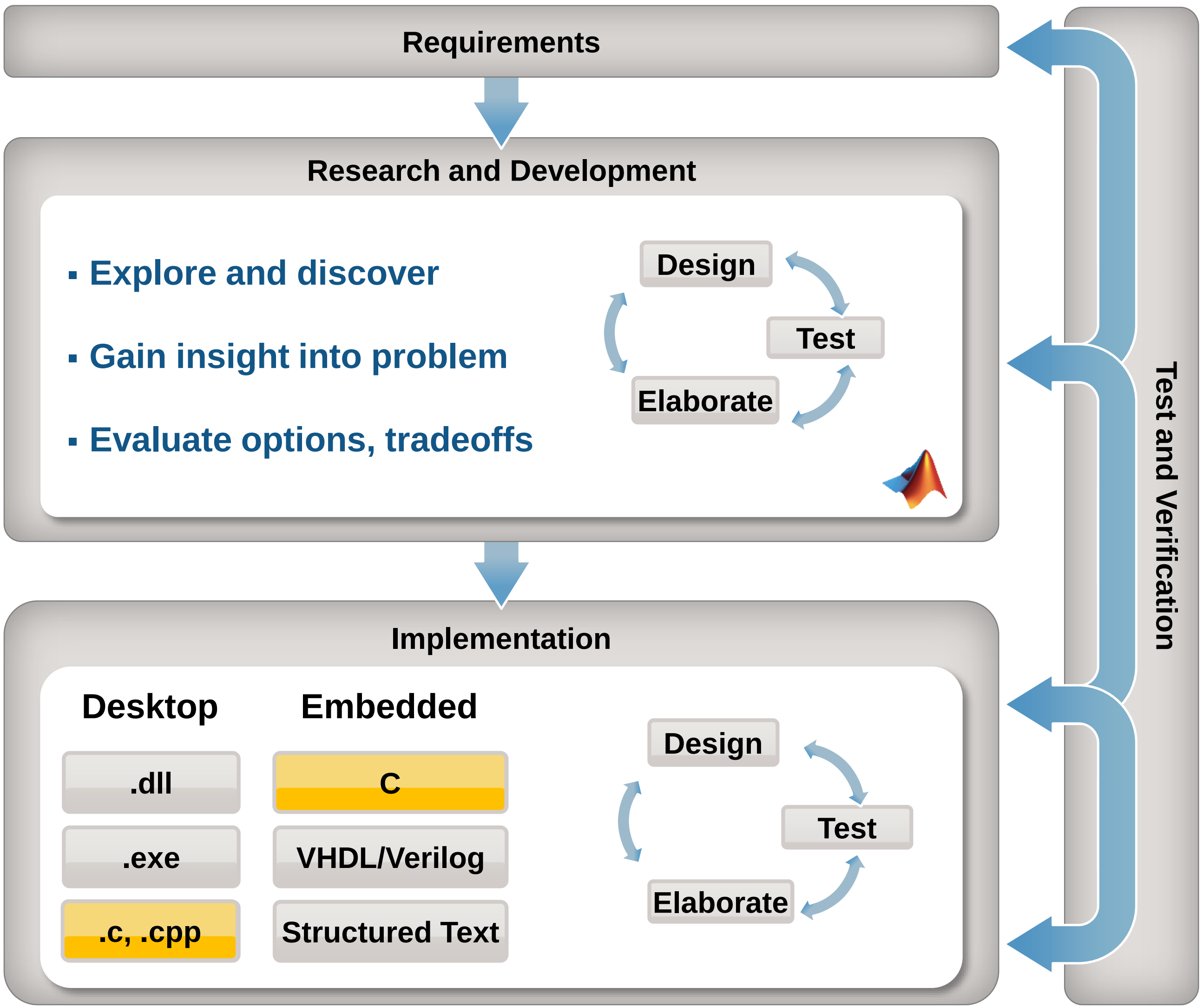
.lib
.dll — **Integrate** MATLAB algorithms with existing C environment using source code and static/dynamic libraries



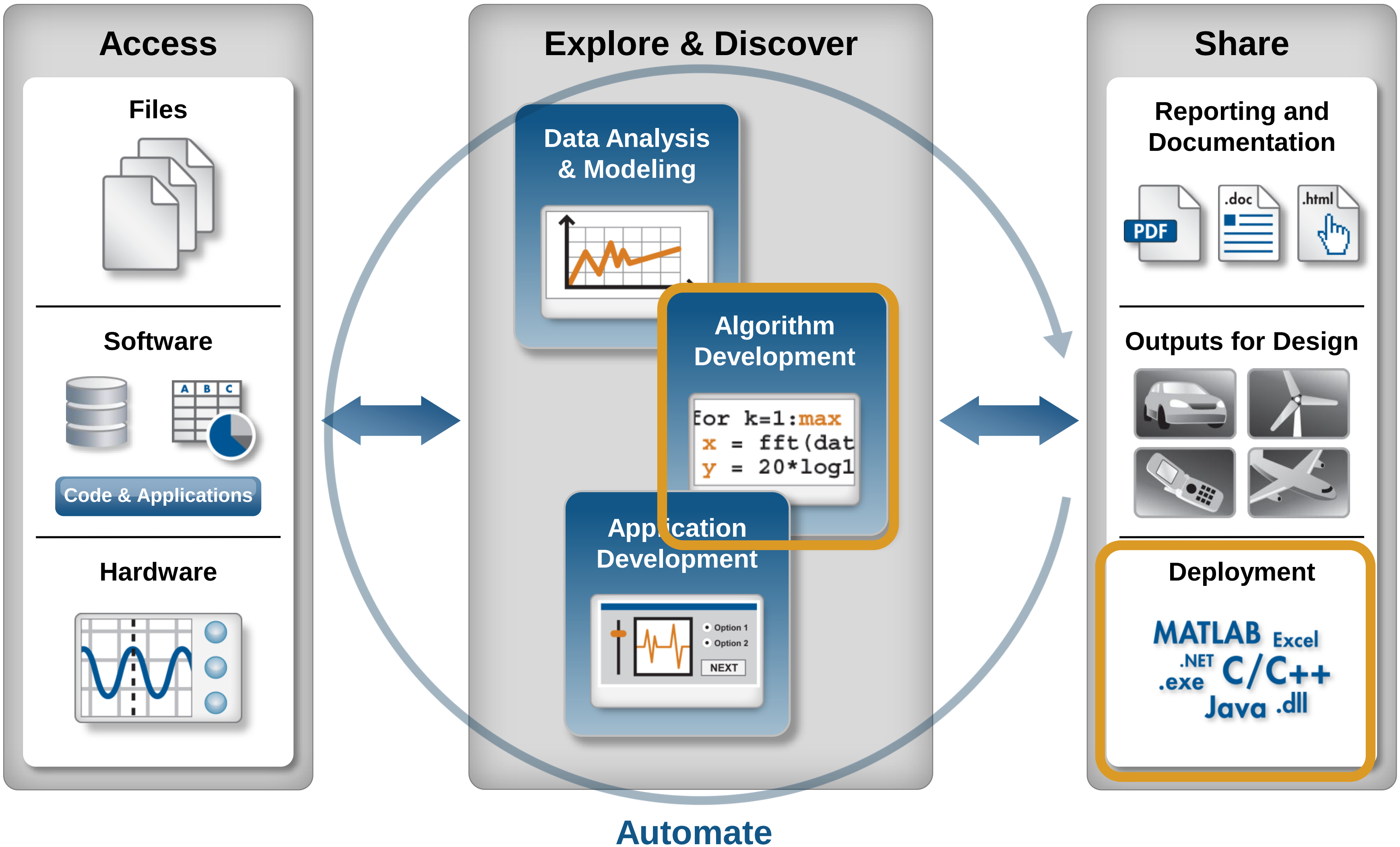
.exe — **Prototype** MATLAB algorithms on desktops as standalone executables

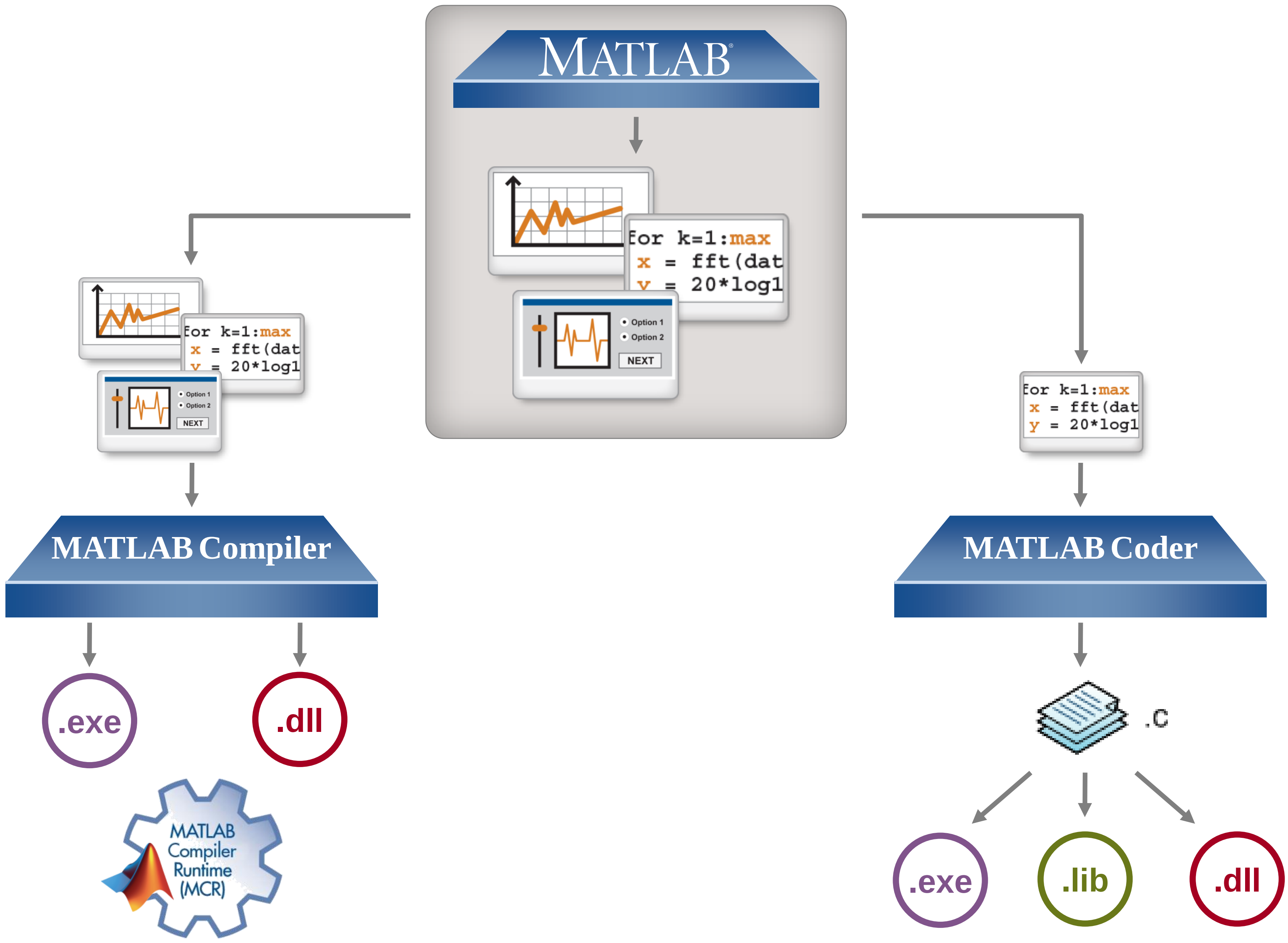
MEX — **Accelerate** user-written MATLAB algorithms

Algorithm Development Process



Technical Computing Workflow

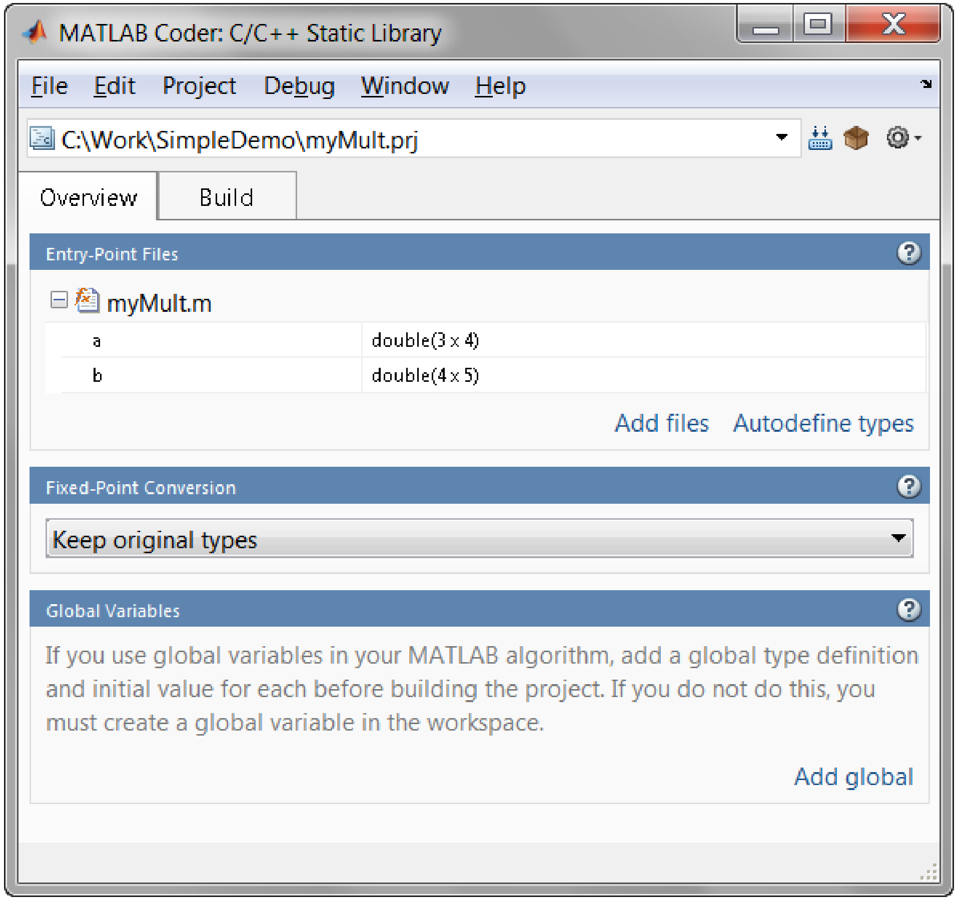



MATLAB
for k=1:max
x = fft(dat
y = 20*log1
Option 1
Option 2
NEXT
MATLAB Compiler
MATLAB Coder
.exe
.dll
MATLAB Compiler Runtime (MCR)
.c
.exe
.lib
.dll

Introductory Demo

`c = a*b`

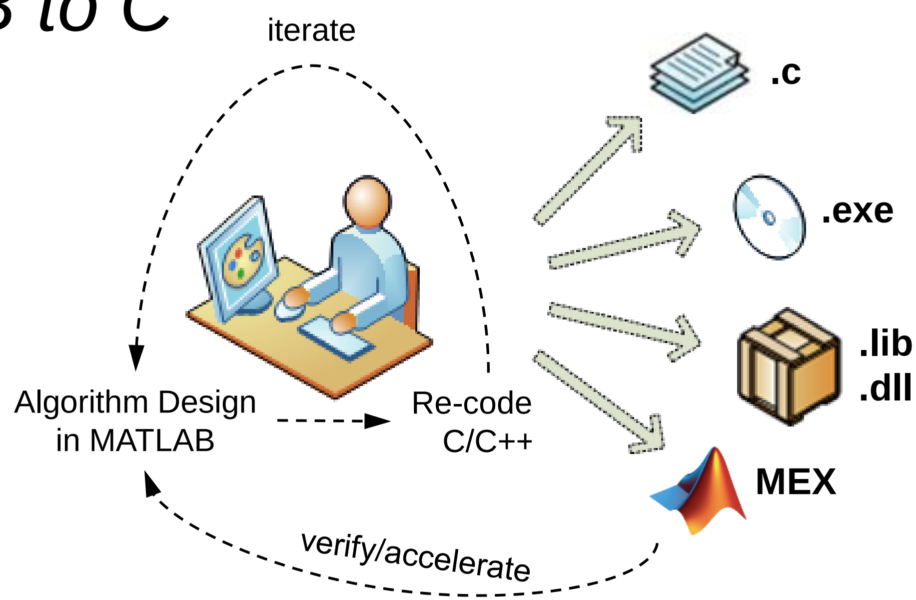
- MATLAB Coder app
- Autodefine input type
- Code generation report



>> Demo

Challenges with Manual Translation

from MATLAB to 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 code in sync
- Manual coding errors
- Time-consuming and expensive process

Challenges with Manual Translation

Implementation Considerations

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

Element by element multiply

Dot product

Matrix multiply

logical
integer
real
complex
…

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];
      }
    }
  }
}
```

MathWorks®

Challenges with Manual Translation

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
```

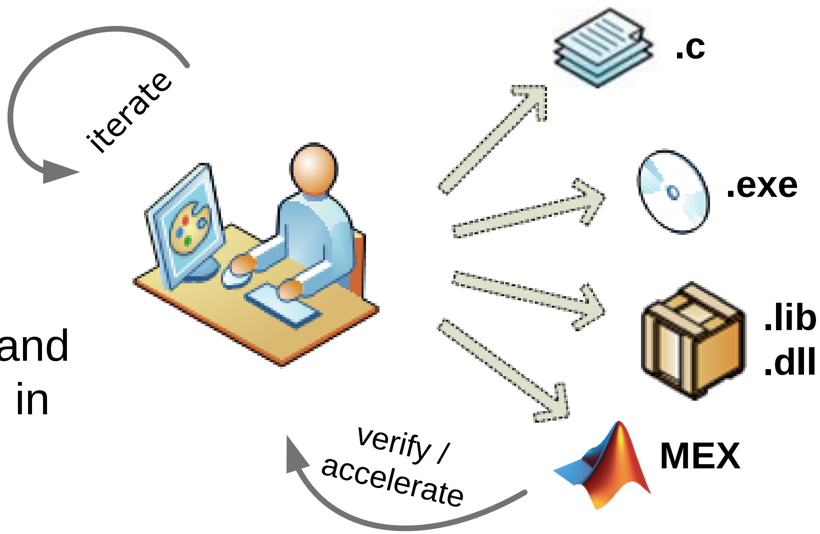
```
#include "kalman_estimate.h"

void kalman_estimate(const double
                     const double
                     double p_es

{
  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)] * p_prd[r2 + 6 * k];
      }
    }
  }

  for (r1 = 0; r1 < 2; r1++) {
    for (r2 = 0;
      a21 = 0.0;
      for (k = 0;
        a21 += k
      }

      S[r1 + (r2
    }
  }

  for (r1 = 0; r1
    for (r2 = 0;
```

```
  if (fabs(S[1]))
    r1 = 1;
    r2 = 0;
  } else {
    r1 = 0;
    r2 = 1;
  }

  a21 = S[r2] /
  a22 = S[2 + r2
  for (k = 0; k
    Y[1 + (k <<
    Y[k << 1] =
  }

  for (r1 = 0; r }
    for (r2 = 0;
      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;
    }
  }
```

Automatic Translation of MATLAB to C



Algorithm Design and Code Generation in MATLAB

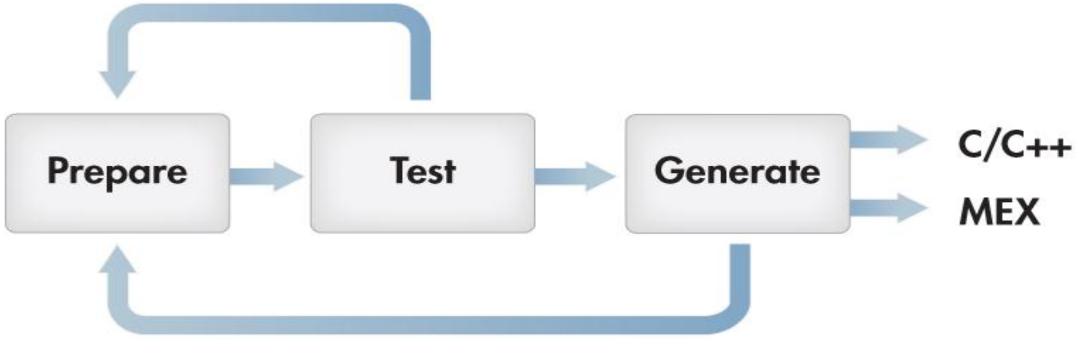
With MATLAB Coder, design engineers can:

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

Agenda

- Motivation
  - Why translate MATLAB to C?
  - Challenges of manual translation
- Using MATLAB Coder
  - Three-step workflow for generating code
- Use cases
  - Integrate algorithms with external C code
  - Accelerate through MEX
  - Prototype by generating EXE
  - Integration with Simulink and Embedded Coder
  - Other deployment solutions
- Summary

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 compliant
- Validate that MATLAB program generates code
- Iterate your MATLAB code to optimize (speed, memory, etc.)
- Verify generated code against testbench using MEX

Generate source code or MEX for final use
- Implement as source, executable, or library

Example: Newton/Raphson Algorithm

Preparing your MATLAB code

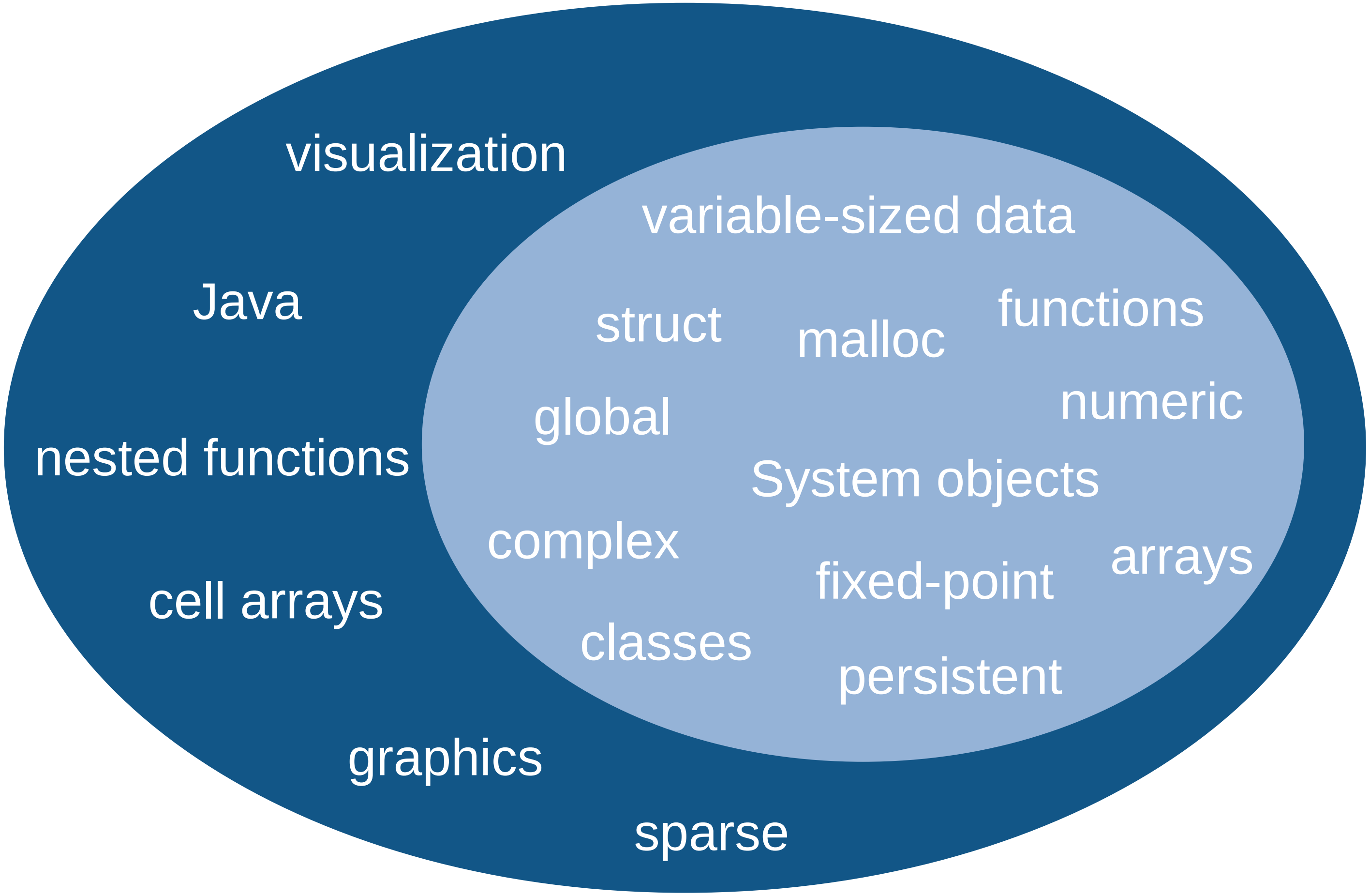
- Code generation readiness tool
- Pre-allocate
- Identify more efficient constructs
- Select code generation options

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

```
function [x,h] = newtonSearchAlgorithm(b,n,tol)
% Given, "a", this function finds the nth root of a
% number by finding where: x^n-a=0.

    notDone = 1;
    aNew    = 0; %Refined Guess Initialization
    a       = 1; %Initial Guess
    cnt     = 0;
    h(1)=a;
    while notDone
        cnt = cnt+1;
        [curVal,slope] = f_and_df(a,b,n); %square
        yint = curVal-slope*a;
        aNew = -yint/slope; %The new guess
        h(cnt)=aNew;
        if (abs(aNew-a) < tol) %Break if it's converged
            notDone = 0;
        elseif cnt>49 %after 50 iterations, stop
            notDone = 0;
            aNew = 0;
```

>> Demo

MATLAB Language Support for Code Generation



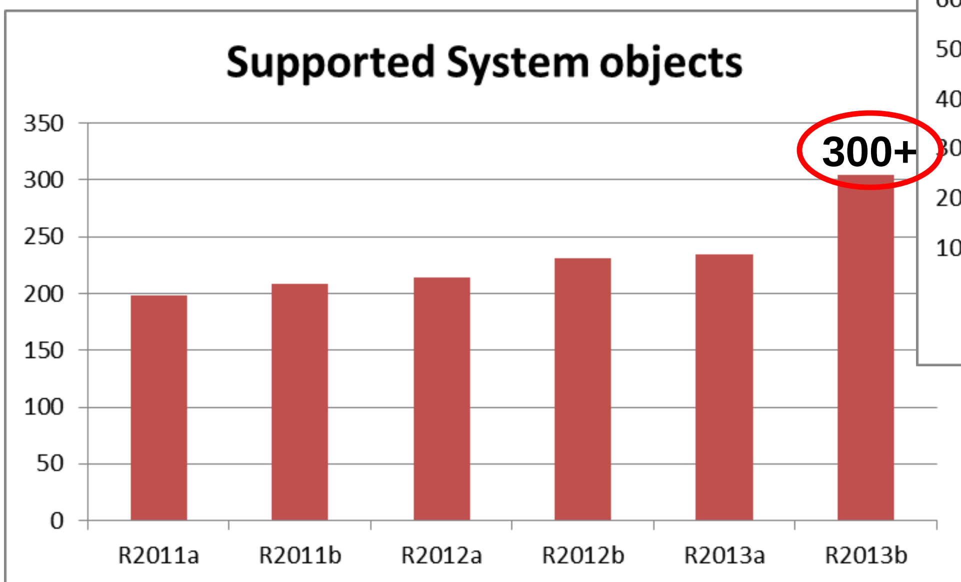
Supported MATLAB Language Features and Functions

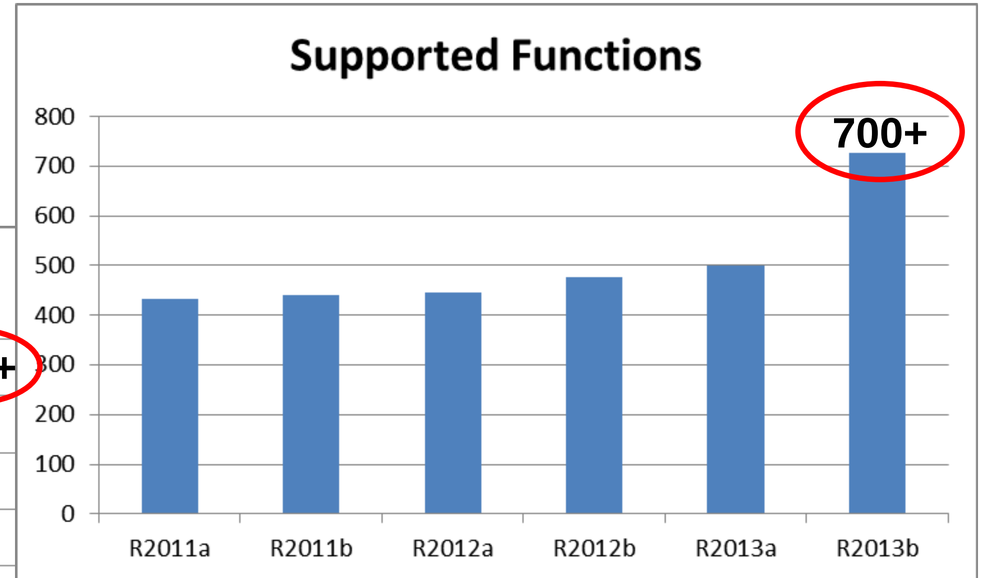
Broad set of language features and functions/system objects supported for code generation

Matrices and Arrays	Data Types	Programming Constructs	Functions

• Matrix operations
• N-dimensional arrays
• Subscripting
• Frames
• Persistent variables
• Global variables
	• Complex numbers
	• Integer math
	• Double/single-precision
	• Fixed-point arithmetic
	• Characters
	• Structures
	• Numeric class
	• Variable-sized data
	• MATLAB Class (MCOS)
	• System objects
		• Arithmetic, relational, and logical operators
		• Program control (`if`, `for`, `while`, `switch`)
			• MATLAB functions and subfunctions
			• Variable-length argument lists
			• Function handles
			
			Supported algorithms
			• More than 700 MATLAB operators and functions
			• More than 300 System objects for:
			  • Signal processing
			  • Communications
			  • Computer vision

Supported Functions & System objects





- Aerospace Toolbox
- Communications System Toolbox
- Computer Vision System Toolbox
- DSP System Toolbox
- Image Processing Toolbox
- Phased Array System Toolbox
- Signal Processing Toolbox
- Statistics Toolbox
- Optimisation Toolbox

Code Generation Support for Statistics Toolbox functions

Use 100+ Statistics Toolbox functions

betacdf	evstat	geornd	kurtosis	nbinrnd	pdf	tstat
betainv	expcdf	geostat	logncdf	nbinstat	poisscdf	unidcdf
betapdf	expinv	gevcdf	logninv	ncfcdf	poissinv	unidinv
betarnd	exppdf	gevinv	lognpdf	ncfinv	poisspdf	unidpdf
betastat	exprnd	gevpdf	lognrnd	ncfpdf	poissrnd	unidrnd
binocdf	expstat	gevrnd	lognstat	ncfrnd	poisstat	unidstat
binoinv	fcdf	gevstat	mad	ncfstat	prctile	unifcdf
binopdf	finv	gpcdf	mnpdf	nctcdf	quantile	unifinv
binornd	fpdf	gpinv	moment	nctinv	randg	unifpdf
binostat	frnd	gppdf	nancov	nctpdf	random	unifrnd
cdf	fstat	gprnd	nanmax	nctrnd	raylcdf	unifstat
chi2cdf	gamcdf	gpstat	nanmean	nctstat	raylinv	wblcdf
chi2inv	gaminv	harmmean	nanmedian	ncx2cdf	raylpdf	wblinv
chi2pdf	gampdf	hygecdf	nanmin	ncx2rnd	raylrnd	wblpdf
chi2rnd	gamrnd	hygeinv	nanstd	ncx2stat	raylstat	wblrnd
chi2stat	gamstat	hygepdf	nansum	normcdf	skewness	wblstat
evcdf	geocdf	hygernd	nanvar	norminv	tcdf	zscore
evinv	geoinv	hygestat	nbincdf	normpdf	tinv	
evpdf	geomean	icdf	nbininv	normrnd	tpdf	
evrnd	geopdf	iqr	nbinpdf	normstat	trnd	

Code Generation Support for Phased Array System Toolbox

Use 80+ functions

```
aictest
albersheim
ambgfun
aperture2gain
az2broadside
azel2phitheta
azel2phithetap
at
azel2uv
azel2uvpat
azelaxes
beat2range
billingsleyicm
broadside2az
cart2sphvec
cbfweights
circpol2pol
dechirp
delayseq
depressionang
dop2speed
dopsteeringvec
effearthradius
espritdoa
fspl
gain2aperture
global2localcoord
grazingang
horizonrange
lcmvweights
local2globalcoord
mdltest
mvdrweights
noisepow
npwgnthresh
phitheta2azel
phitheta2azelpat
phitheta2uv
phitheta2uvpat
physconst
pol2circpol
polellip
polloss
polratio
polsignature
pulsint
radareqpow
radareqrng
radareqsnr
radarvcd
radialspeed
range2beat
range2bw
range2time
rangeangle
rdcoupling
rocpfa
rocsnr
rootmusicdoa
rotx
roty
rotz
sensorcov
sensorsig
shnidman
speed2dop
sph2cartvec
spsmooth
steervec
stokes
stretchfreq2rng
surfacegamma
surfclutterrcs
systemp
time2range
unigrid
uv2azel
uv2azelpat
uv2phitheta
uv2phithetapat
val2ind
```

Code Generation Support for Phased Array System Toolbox

Use 50+ System objects

phased.CosineAntennaElement
phased.CrossedDipoleAntennaElement
phased.CustomAntennaElement
phased.CustomMicrophoneElement
phased.IsotropicAntennaElementIsotropic
phased.OmnidirectionalMicrophoneElement
phased.ShortDipoleAntennaElement
phased.ULA
phased.URA
phased.ConformalArray
phased.PartitionedArray
phased.ReplicatedSubarray
phased.SteeringVector
phased.ArrayGain
phased.ArrayResponse
phased.ElementDelay
phased.Collector
phased.Radiator
phased.WidebandCollector
phased.LinearFMWaveformLinear

phased.PhaseCodedWaveform
phased.RectangularWaveform
phased.SteppedFMWaveform
phased.FMCWWaveform
phased.MatchedFilter
phased.Transmitter
phased.ReceiverPreamp
phased.PhaseShiftBeamformer
phased.LCMVBeamformer
phased.MVDRBeamformer
phased.SubbandPhaseShiftBeamformer
phased.FrostBeamformer
phased.TimeDelayBeamformer
phased.TimeDelayLCMVBeamformer
phased.SteeringVector
phased.SumDifferenceMonopulseTracker
phased.SumDifferenceMonopulseTracker2D
phased.BeamscanEstimator
phased.BeamscanEstimator2D
phased.MVDREstimator

phased.MVDREstimator2D
phased.RootMUSICEstimator
phased.RootWSFEstimator
phased.ESPRITEstimator
phased.BeamspaceESPRITEstimator
phased.STAPSMIBeamformer
phased.DPCACanceller
phased.ADPCACanceller
phased.AngleDopplerResponse
phased.CFARDetector
phased.MatchedFilter
phased.RangeDopplerResponse
phased.StretchProcessor
phased.TimeVaryingGain
phased.FreeSpace
phased.RadarTarget
phased.ConstantGammaClutter
phased.BarrageJammer
phased.Platform

Agenda

- Motivation
  - Why translate MATLAB to C?
  - Challenges of manual translation
- Using MATLAB Coder
  - Three-step workflow for generating code
- Use cases
  - Integrate algorithms with external C code
  - Accelerate through MEX
  - Prototype by generating EXE
  - Integration with Simulink and Embedded Coder
  - Other deployment solutions
- Summary

MATLAB Coder Use Cases

.lib
.dll

.exe

Integrate
algorithms with custom software

Prototype
algorithms on PCs

Accelerate
algorithm execution

Implement
algorithms on embedded processors

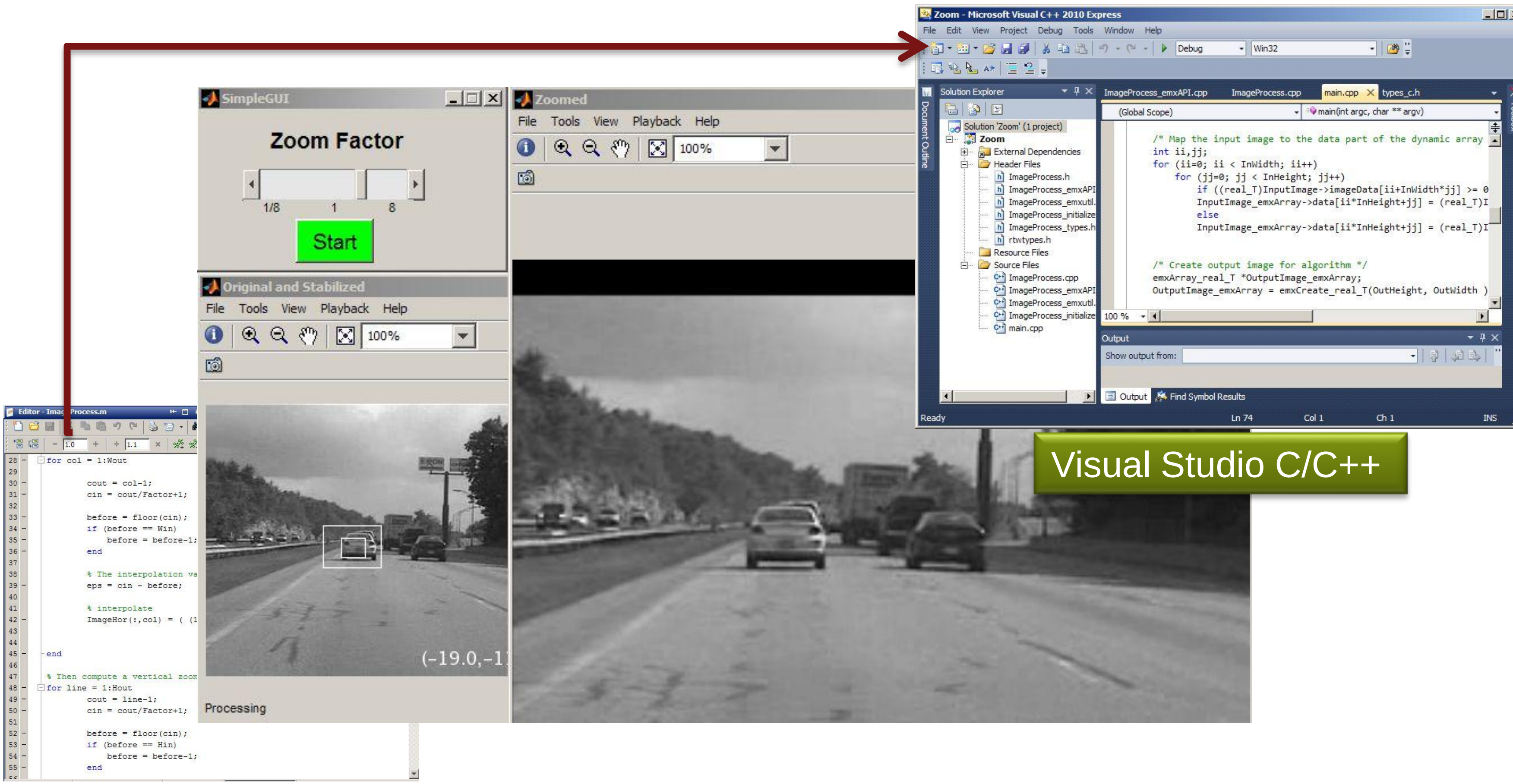
MEX

.c

Example: Code Integration

with Visual Studio Parent Project

Integrate
algorithms with custom software



Visual Studio C/C++

MATLAB

>> Demo

Example: External Code Integration

coder.ceval

Integrate third-party libraries with generated code

- Uses coder.target to distinguish between MATLAB simulation and code generation
- Integrates custom C code to replace automatically generated C code
- Just need to specify additional files and include path

```
function y = custom_conv_ceval(x,h)    %#codegen
% This example uses the coder.target and coder.ceval functions to
% explicitly define what should be run during simulation, and what should
% be used when generating C code.

% intialise size and type of output
y = zeros(length(x)+length(h)-1,1);

if coder.target('MATLAB')
  % Executing in MATLAB
  y = conv(x,h);
else
  % Call to C function 'custom_conv.c' using coder.ceval
  coder.ceval('custom_conv', coder.rref(x), uint32(length(x)), ...
      coder.rref(h), uint32(length(h)), ...
      coder.wref(y));
end
```

```
void custom_conv_ceval(const double x[10], const double h[3], double y[12])
{
  /*  intialise size and type of output */
  /*  Call to C function 'custom_conv.c' using coder.ceval */
  custom_conv(x, 10U, h, 3U, y);
}
```

External Code Integration using *coder.ExternalDependency*

Integrate third-party libraries with generated code

- Encapsulates API to an external library, object file, or C/C++ source code
- Integrates with external libraries without user intervention
- Automatically adds necessary compiler and linker flags and objects

End user calls a toolbox function

```
[h, theta, rho] = hough(bw, 'Theta', theta);
```

Toolbox function invokes an external-dependency API function

```
h = images.HoughBuildable.houghcore(bwIn, rows, columns, rho, rhoLength, ...
    theta, thetaLength, h);
```

External dependency API is defined by deriving from coder.ExternalDependency

```
classdef HoughBuildable < coder.ExternalDependency
% HOUGHBUILDABLE ExternalDependency class for the Hough transform
%
% Implements the methods to update the BuildInfo objects at compile-time
% and build-time.
%
% Abstracts the API to the Hough transform external library.
%
```

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 (e.g. BLAS and LAPACK)

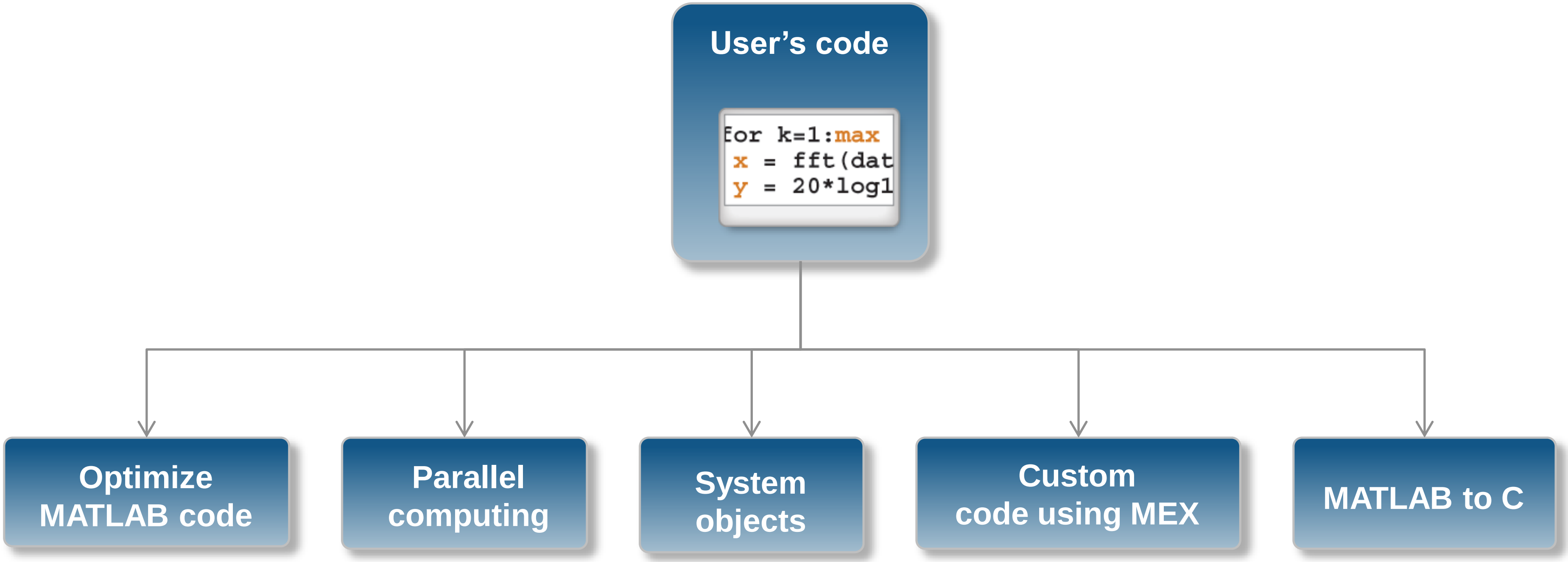
  - Different optimization of the same algorithm

- More computational resources

  Single-threaded vs. multithreaded (multithreaded BLAS)

  - Leveraging additional processors, cores, GPUs, FPGAs, etc.

Accelerating Algorithm Execution



Example: Newton-Raphson

Accelerate
algorithm execution

```matlab
%% Script to compare execution time of MATLAB code to generated MEX code
iter = 1000;

%% Time MATLAB code
elTime = zeros(iter,1);
for i = 1:iter
    tic
    nrt(1e8,12,1e-9);
    t = toc;
    elTime(i) = t;
end
matTime = mean(elTime);
disp(['Mean MATLAB time is: ' num2str(matTime) ' seconds']);

%% Time MEX Code
elTime = zeros(iter,1);
for i = 1:iter
    tic
    nrt_mex(1e8,12,1e-9);
    t = toc;
    elTime(i) = t;
end
mexTime = mean(elTime);
disp(['Mean MEX time is: ' num2str(mexTime) ' seconds']);

%% Speed Up Factor
speedUp = matTime/mexTime;
disp(['Speed up factor is: ' num2str(speedUp) 'X']);
```

Command Window

```
>> speedTest
Mean MATLAB time is: 0.00033502 seconds
Mean MEX time is: 2.8008e-05 seconds
Speed up factor is: 11.9615X
>>
```

>> Demo

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 vectorisation isn’t possible

- When you **may not** see a speedup:
  - MATLAB implicitly multithreads computation.
  - Built-in functions call IPP or BLAS libraries.

Multicore `parfor` Support

in MEX Functions

MathWorks R2012b

Run MATLAB faster by generating MEX functions that execute on multiple cores

- Relies on OpenMP technology to parallelize `parfor` loops
- OpenMP supported by Microsoft, Intel, and GCC C compilers

Editor - C:\MATLAB\test_parfor.m

test_parfor.m

```
function a = test_parfor %#codegen
size=80;
a=ones(size,256);
r=rand(size,256);
parfor i=1:size
    a(i,:)=real(fft(r(i)));
end
```

Command Window

```
>> codegen test_parfor
>> tic; test_parfor_mex; toc;
Elapsed time is 0.162044 seconds.
>>
```

Multicore `parfor` Support

for Standalone Code Generation

Use `parfor` to generate parallel C/C++ code using OpenMP

```
parfor plane = 1:sz(3)
    planeHist = zeros(1,L);
    for y = 1:N
        for x = 1:M
            r = originalImage(y,x,plane);
            planeHist(r+1) = planeHist(r+1) + 1;
        end
    end
    originalHist(plane,:) = planeHist;
end
```

```
#pragma omp parallel for \
 num_threads(omp_get_max_threads()) \
 private(planeHist_data,loop_ub,i0,y,x,i1)

  for (plane = 0; plane < 3; plane++) {
    loop_ub = (int)L;
    for (i0 = 0; i0 < loop_ub; i0++) {
      planeHist_data[i0] = 0.0;
    }
```

- Requires C/C++ compiler supporting OpenMP

```
>> coderdemo_contrast_enhancer
```

Example: Standalone Executable

Video Stabilisation

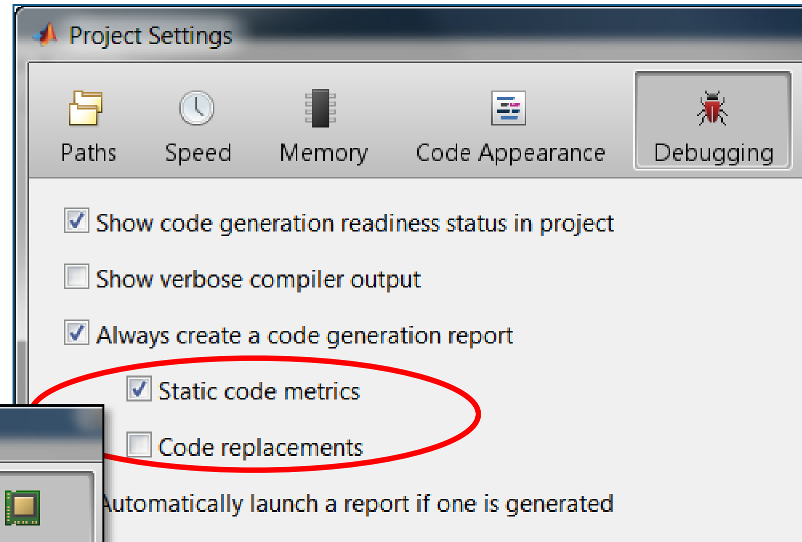
Prototype
algorithms on PCs

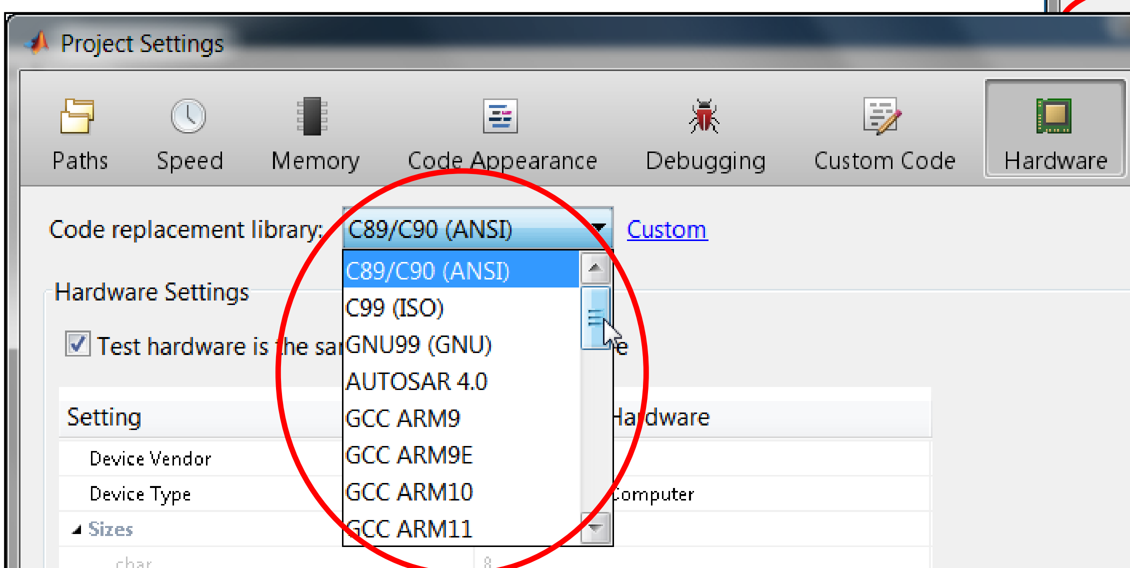
- Need to provide `main.c` for entry point
- Use System objects from Computer Vision System Toolbox to stream, process, and display video

>> Demo

Working with Embedded Coder

- Advanced support for MATLAB Coder, including:
  - Speed
  - Memory
  - Code appearance
  - Hardware-specific optimization





2. Global Variables [hide]

Global variables defined in the generated code.

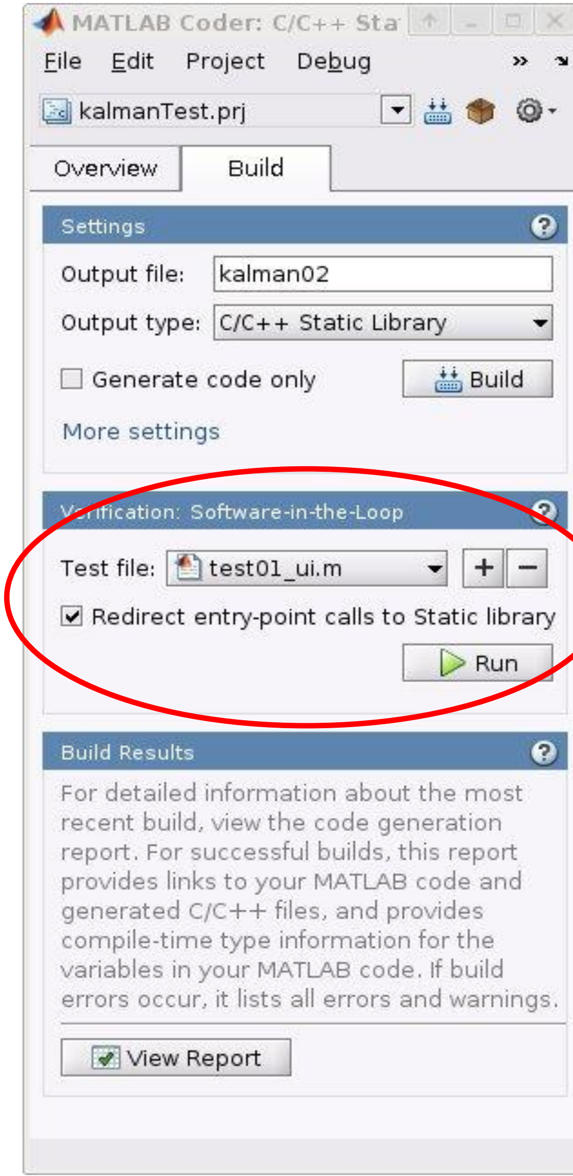
Global Variable	Size (bytes)
myglobal	240
Total	240

Working with Embedded Coder

Software-in-the-Loop Verification

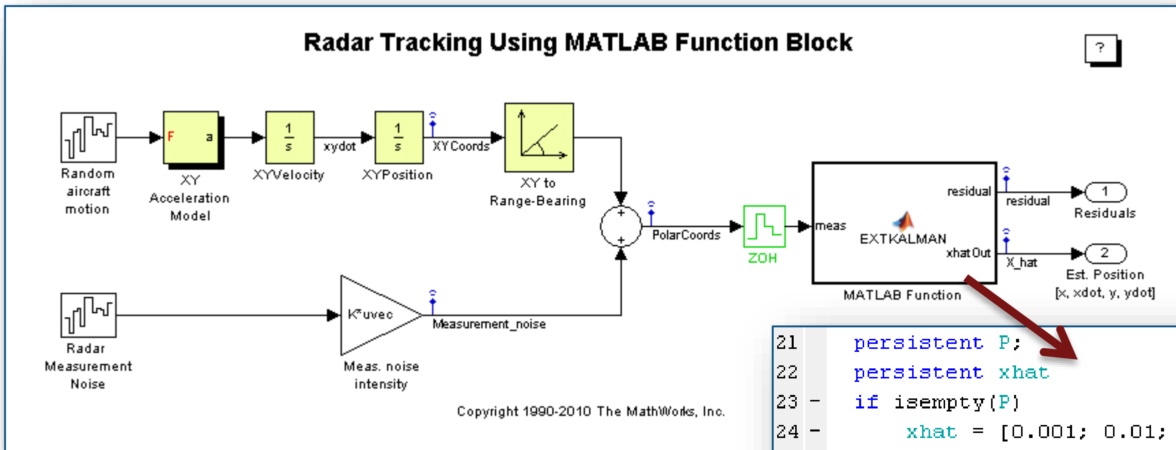
Verify numerical behavior of generated source code through software-in-the-loop testing

- Reuse MATLAB tests to exercise standalone source code compiled for host computer
- Integrate SIL verification with the existing Project verification tool and command-line utility `coder.runTest`
- Step through generated code in Microsoft Visual Studio debugger during SIL testing when using `coder.runTest`



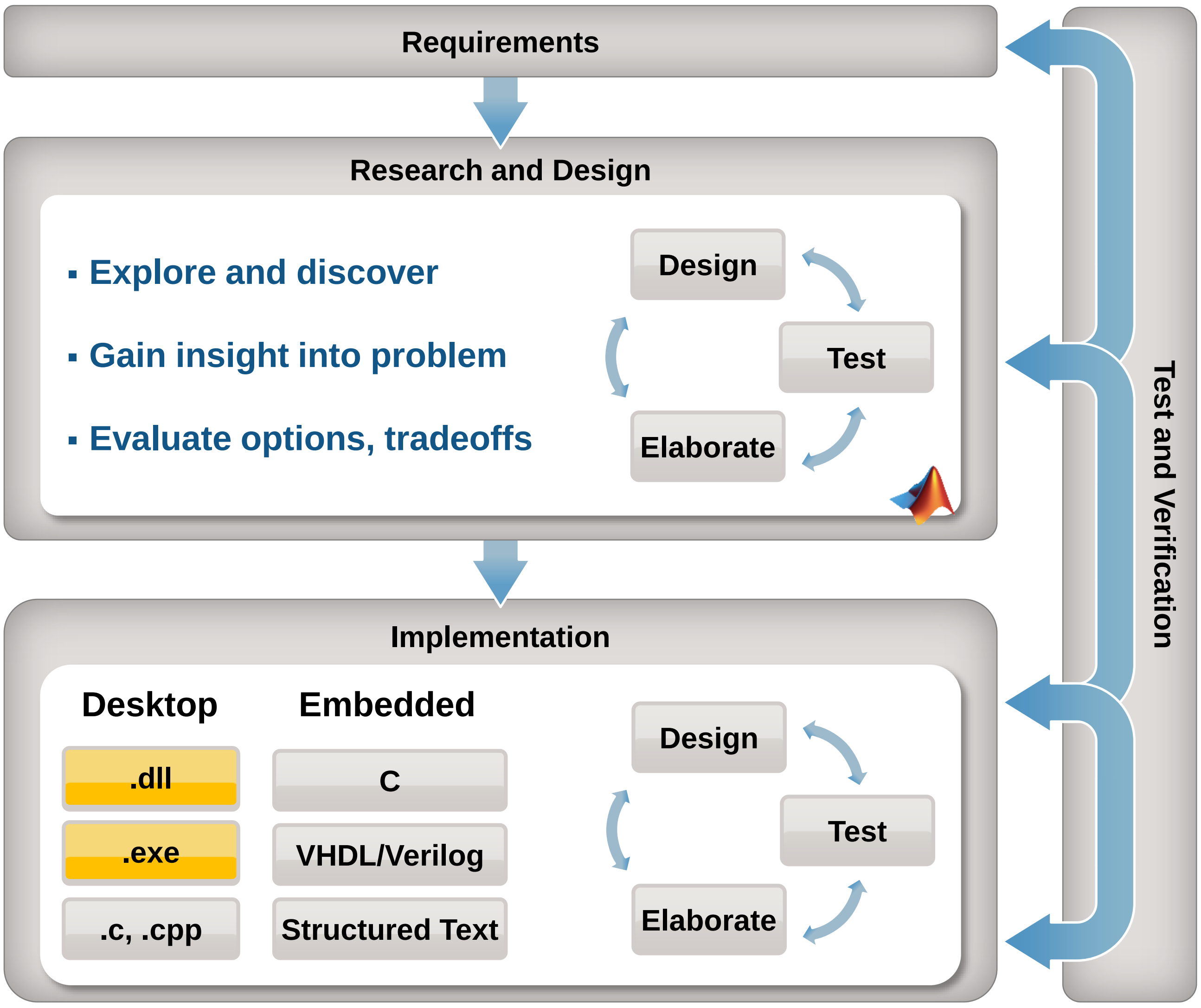
Working with Simulink

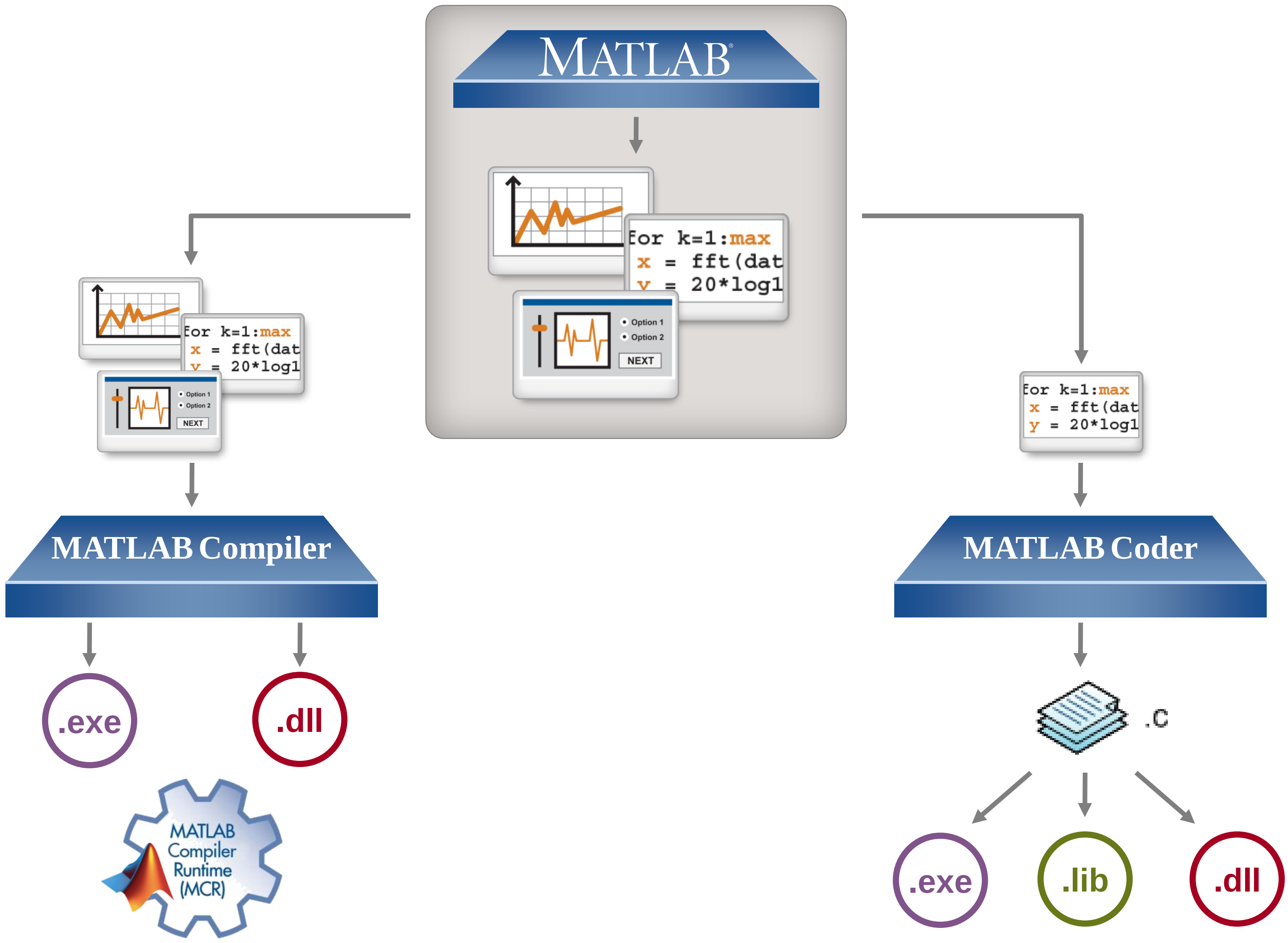
MATLAB Function block in Simulink




```
persistent P;
persistent xhat
if isempty(P)
    xhat = [0.001; 0.01; 0.001; 400];
    P = zeros(4);
end

%  Radar update time deltat is inherited from mo

% 1. Compute Phi, Q, and R
Phi = [1 deltat 0 0; 0 1 0 0 ; 0 0 1 deltat; 0 0
Q =  diag([0 .005 0 .005]);
R =  diag([300^2 0.001^2]);

% 2. Propagate the covariance matrix:
P = Phi*P*Phi' + Q;

% 3. Propagate the track estimate::
xhat = Phi*xhat;

% 4 a). Compute observation estimates:
```

Other Desktop Deployment Options

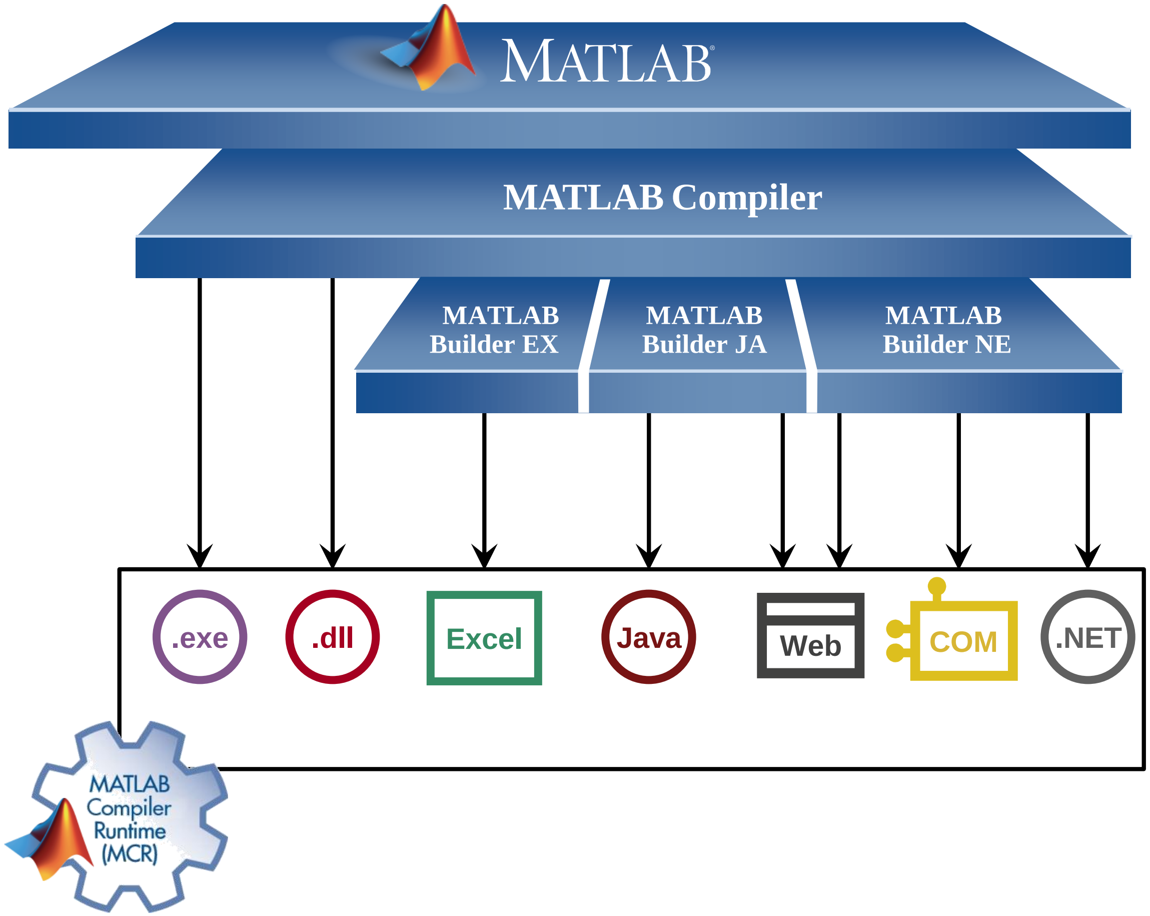



MATLAB
for k=1:max
x = fft(dat
y = 20*log1
Option 1
Option 2
NEXT
MATLAB Compiler
MATLAB Coder
.exe
.dll
MATLAB Compiler Runtime (MCR)
.c
.exe
.lib
.dll

Other Deployment Options

MATLAB Compiler

- Share applications
  - Creates desktop or web software components
  - Supports full MATLAB language and most toolboxes
  - Requires MATLAB Compiler Runtime
    - Free run-time library
    - Royalty-free deployment



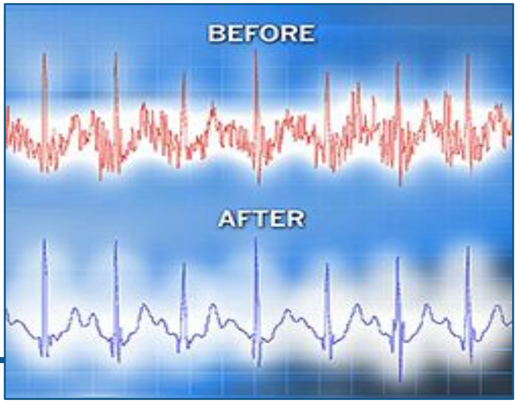
Choosing the Right Deployment Solution

MATLAB Coder or MATLAB Compiler

.c

	MATLAB Coder	MATLAB Compiler
Output	Portable and readable C source code	Executable or software component/library
MATLAB support	Subset of language Some toolboxes	Full language Most toolboxes Graphics
Runtime requirement	None	MATLAB Compiler Runtime
License model	Royalty-free	Royalty-free

VivaQuant Accelerates Development and Validation of Embedded Device for Ambulatory ECG Sensing



ECG snippet before and after processing with VivaQuant's embedded in-band noise removal algorithm.

Challenge

Design and implement an embedded system for extracting accurate information from noisy electrocardiogram signals

Solution

Use MATLAB to develop an algorithm for removing in-band noise, and use Fixed-Point Designer and MATLAB Coder to implement it on an ARM Cortex-M series processor

Results

- Development accelerated by 300%
- Power and memory consumption minimized
- Rigorous testing enabled

"MATLAB, MATLAB Coder, and Fixed-Point Designer enabled our small team to develop a complex real-time signal processing algorithm, optimize it to reduce power and memory requirements, accelerate embedded code implementation, and perform the rigorous testing required for medical device validation."

Marina Brockway

VivaQuant

[Link to user story]

Agenda

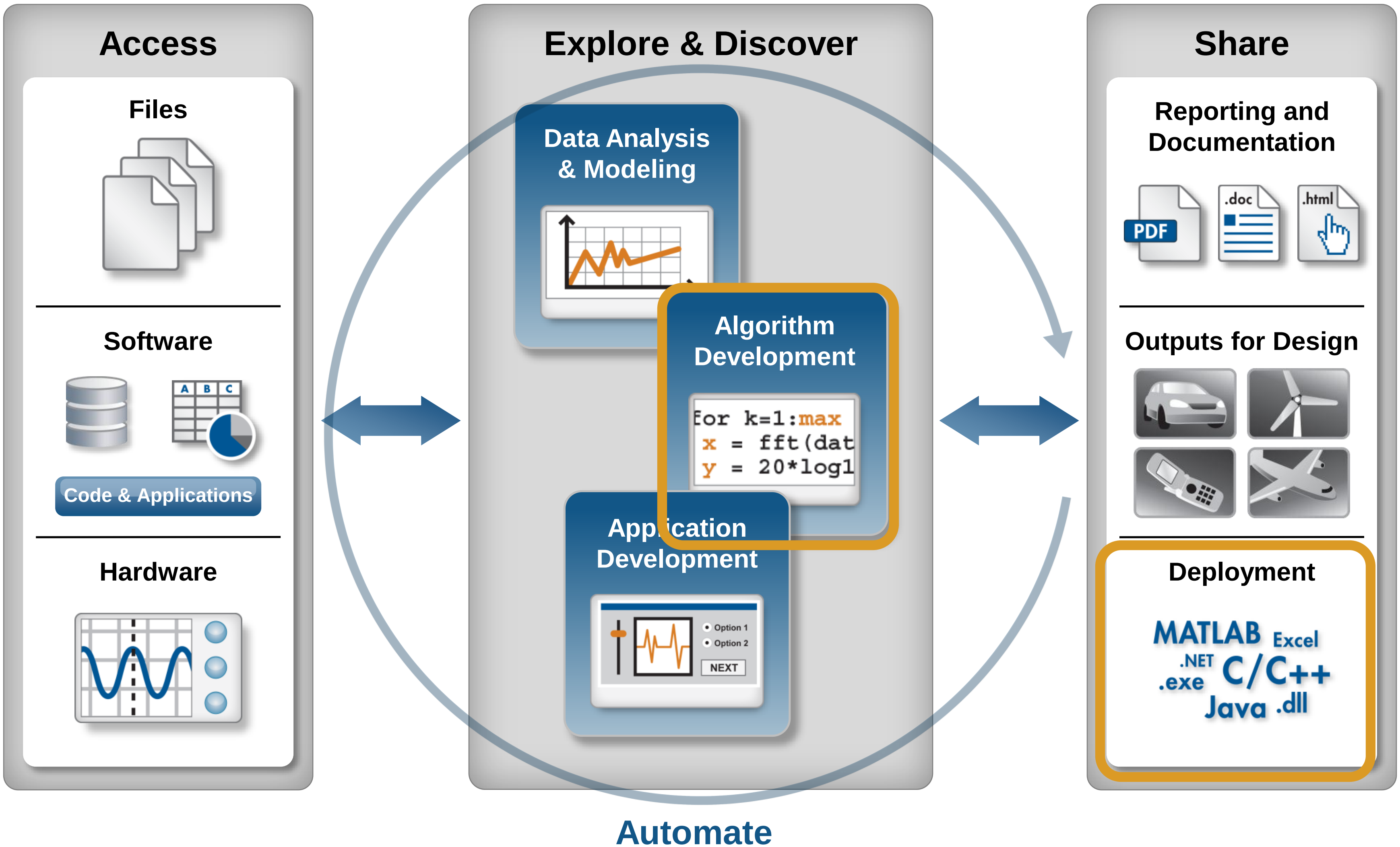
- Motivation
  - Why translate MATLAB to C?
  - Challenges of manual translation

- Using MATLAB Coder
  - Three-step workflow for generating code

- Use cases
  - Integrate algorithms with external C code
  - Accelerate through MEX
  - Prototype by generating EXE
  - Integration with Simulink and Embedded Coder
  - Other deployment solutions

- Summary

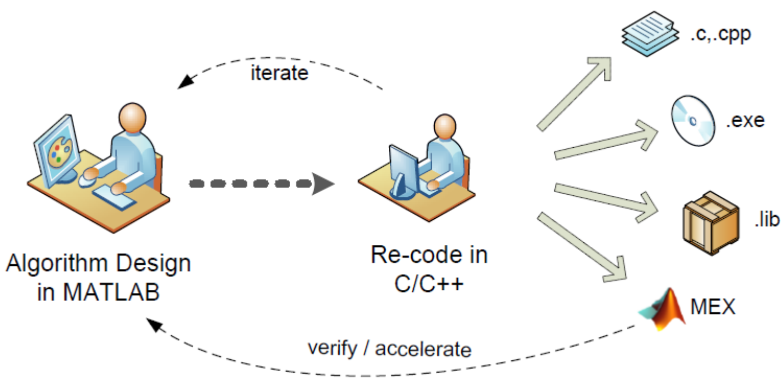
Summary



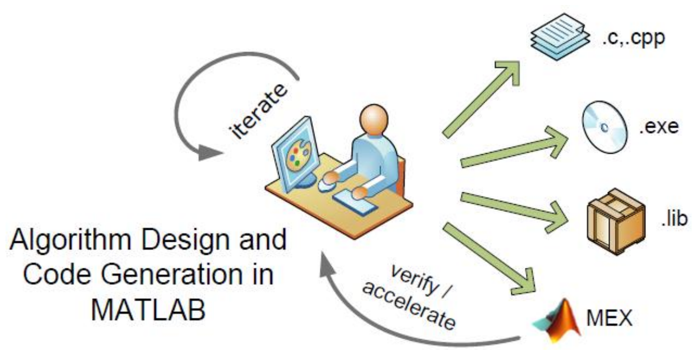
Access
Files
Software
Code & Applications
Hardware
Explore & Discover
Data Analysis & Modeling
Algorithm Development
for k=1:max
x = fft(dat
y = 20*log1
Application Development
Option 1
Option 2
NEXT
Share
Reporting and Documentation
PDF
.doc
.html
Outputs for Design
Deployment
MATLAB
Excel
.NET
.exe
C/C++
Java
.dll
Automate

Automatic C Code Generation

Accelerates Development



Develop algorithm | Test | Convert to C/C++ | Test | Iterate | Test | Deploy



Develop algorithm | Test and Iterate | Test | Deploy | Time savings

Generate code

Takeaways

- MATLAB provides a direct path to C code
  - Both floating-point and fixed-point
- Suitable for applications where
  - Source code is required
  - Small memory footprint is required
- Automatic Code Generation
  - accelerates design iterations
  - reduces verification effort

More Information

- To learn more, visit the product page:
  mathworks.com/products/matlab-coder

- To request a trial license:
  - Talk to your MathWorks account manager to request a trial license and set up a guided evaluation with an application engineer

- Contact us
  - info@mathworks.com.au
  - 02 8669 4700

http://www.mathworks.com.au/services/training/

Training courses - Sydney

Course Name	Course Code	Start Date	End Date	City
MATLAB Fundamentals	MLBE	5-Aug-14	7-Aug-14	Sydney
Stateflow for Logic Driven System Modeling	SLSF	14-Aug-14	15-Aug-14	Sydney
Embedded Coder for Production Code Generation	SLEC	16-Sep-14	18-Sep-14	Sydney
MATLAB Fundamentals	MLBE	30-Sep-14	2-Oct-14	Sydney
MATLAB Programming Techniques	MLPR	14-Oct-14	15-Oct-14	Sydney
Physical Modeling of Multidomain Systems with Simscape	SLPM-S	16-Oct-14	16-Oct-14	Sydney
Statistical Methods in MATLAB	MLST	11-Nov-14	12-Nov-14	Sydney
Image Processing with MATLAB	MLIP	13-Nov-14	14-Nov-14	Sydney

Course Name	Course Code	Start Date	End Date	City
MATLAB Programming Techniques	MLPR	12-Aug-14	13-Aug-14	e-learning
MATLAB Fundamentals	MLBE	2-Sep-14	4-Sep-14	e-learning
Simulink for System and Algorithm Modeling	SLBE	11-Sep-14	12-Sep-14	e-learning
MATLAB Fundamentals	MLBE	9-Dec-14	11-Dec-14	e-learning

Training courses - Adelaide

Course Name	Course Code	Start Date	End Date	City
Simulink for System and Algorithm Modeling	SLBE	21-Aug-14	22-Aug-14	Adelaide
Image Processing with MATLAB	MLIP	9-Sep-14	10-Sep-14	Adelaide
MATLAB Fundamentals	MLBE	4-Nov-14	6-Nov-14	Adelaide
Parallel Computing with MATLAB	MLPC	18-Nov-14	19-Nov-14	Adelaide

Course Name	Course Code	Start Date	End Date	City
MATLAB Programming Techniques	MLPR	12-Aug-14	13-Aug-14	e-learning
MATLAB Fundamentals	MLBE	2-Sep-14	4-Sep-14	e-learning
Simulink for System and Algorithm Modeling	SLBE	11-Sep-14	12-Sep-14	e-learning
MATLAB Fundamentals	MLBE	9-Dec-14	11-Dec-14	e-learning

Training courses - Brisbane

Course Name	Course Code	Start Date	End Date	City
Simulink for System and Algorithm Modeling	SLBE	22-Oct-14	23-Oct-14	Brisbane
Optimization Techniques in MATLAB	MLOP	24-Oct-14	24-Oct-14	Brisbane
MATLAB Fundamentals	MLBE	11-Nov-14	13-Nov-14	Brisbane

Course Name	Course Code	Start Date	End Date	City
MATLAB Programming Techniques	MLPR	12-Aug-14	13-Aug-14	e-learning
MATLAB Fundamentals	MLBE	2-Sep-14	4-Sep-14	e-learning
Simulink for System and Algorithm Modeling	SLBE	11-Sep-14	12-Sep-14	e-learning
MATLAB Fundamentals	MLBE	9-Dec-14	11-Dec-14	e-learning

Training courses - Melbourne

Course Name	Course Code	Start Date	End Date	City
MATLAB Fundamentals	MLBE	19-Aug-14	21-Aug-14	Melbourne
Building Interactive Applications in MATLAB	MLGU	22-Aug-14	22-Aug-14	Melbourne
Physical Modeling of Multidomain Systems with Simscape	SLMPM-S	22-Sep-14	22-Sep-14	Melbourne
MATLAB Programming Techniques	MLPR	23-Sep-14	24-Sep-14	Melbourne
Statistical Methods in MATLAB	MLST	25-Sep-14	26-Sep-14	Melbourne
MATLAB Fundamentals	MLBE	14-Oct-14	16-Oct-14	Melbourne
Optimization Techniques in MATLAB	MLOP	17-Oct-14	17-Oct-14	Melbourne
Parallel Computing with MATLAB	MLPC	28-Oct-14	29-Oct-14	Melbourne
Signal Processing with MATLAB	MLSG	30-Oct-14	31-Oct-14	Melbourne
Signal Processing with Simulink	SLBE-G	18-Nov-14	20-Nov-14	Melbourne

Course Name	Course Code	Start Date	End Date	City
MATLAB Programming Techniques	MLPR	12-Aug-14	13-Aug-14	e-learning
MATLAB Fundamentals	MLBE	2-Sep-14	4-Sep-14	e-learning
Simulink for System and Algorithm Modeling	SLBE	11-Sep-14	12-Sep-14	e-learning
MATLAB Fundamentals	MLBE	9-Dec-14	11-Dec-14	e-learning

MathWorks®

Training courses - Perth

Course Name	Course Code	Start Date	End Date	City
Statistical Methods in MATLAB	MLST	26-Aug-14	27-Aug-14	Perth
Simulink for System and Algorithm Modeling	SLBE	28-Aug-14	29-Aug-14	Perth
MATLAB Fundamentals	MLBE	23-Sep-14	25-Sep-14	Perth

Course Name	Course Code	Start Date	End Date	City
MATLAB Programming Techniques	MLPR	12-Aug-14	13-Aug-14	e-learning
MATLAB Fundamentals	MLBE	2-Sep-14	4-Sep-14	e-learning
Simulink for System and Algorithm Modeling	SLBE	11-Sep-14	12-Sep-14	e-learning
MATLAB Fundamentals	MLBE	9-Dec-14	11-Dec-14	e-learning

MathWorks

Training courses - New Zealand

Course Name	Course Code	Start Date	End Date	City
Signal Processing with Simulink	SLBE-G	3-Sep-14	5-Sep-14	Christchurch
Statistical Methods in MATLAB	MLST	7-Oct-14	8-Oct-14	Wellington

Course Name	Course Code	Start Date	End Date	City
MATLAB Programming Techniques	MLPR	12-Aug-14	13-Aug-14	e-learning
MATLAB Fundamentals	MLBE	2-Sep-14	4-Sep-14	e-learning
Simulink for System and Algorithm Modeling	SLBE	11-Sep-14	12-Sep-14	e-learning
MATLAB Fundamentals	MLBE	9-Dec-14	11-Dec-14	e-learning

Training courses - OnLine

Course Name	Course Code	Start Date	End Date	City
MATLAB Programming Techniques	MLPR	12-Aug-14	13-Aug-14	e-learning
MATLAB Fundamentals	MLBE	2-Sep-14	4-Sep-14	e-learning
Simulink for System and Algorithm Modeling	SLBE	11-Sep-14	12-Sep-14	e-learning
MATLAB Fundamentals	MLBE	9-Dec-14	11-Dec-14	e-learning