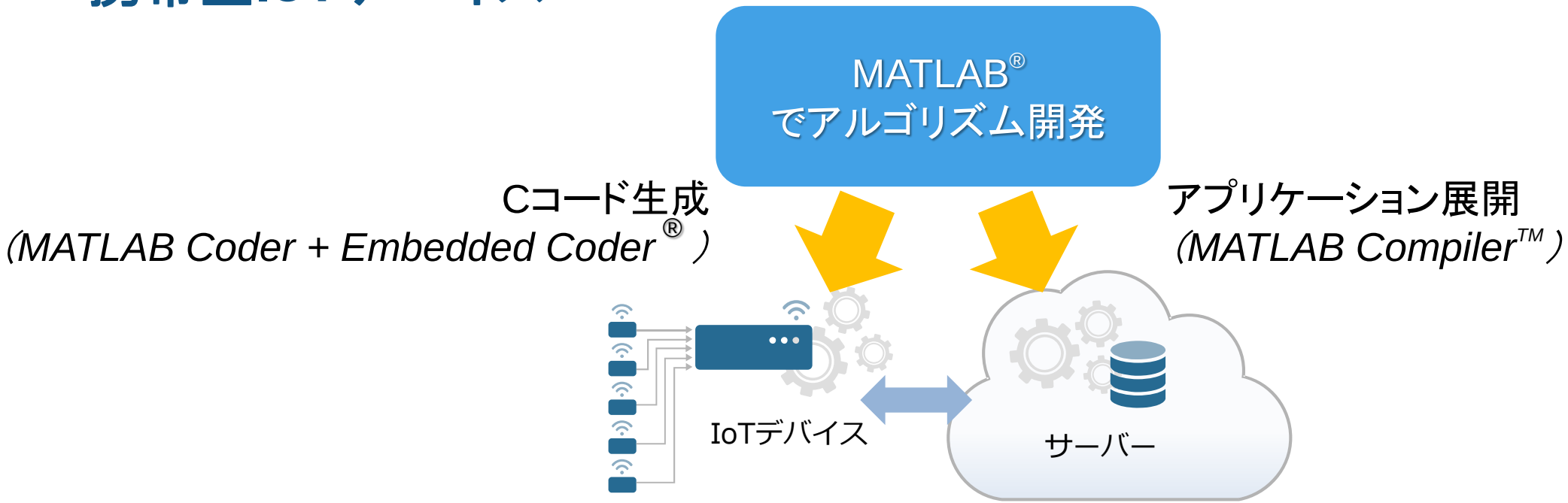
MATLABアルゴリズムから Cコード生成のワークフローと最適化

～信号・画像処理・機械学習編～

MathWorks Japan
アプリケーションエンジニアリング部
松本　充史

*MATLAB Coder™*導入事例
携帯型IoTデバイス



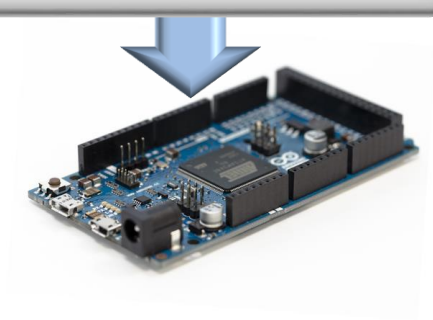
- 実装対象
  - > バッテリー駆動IoT機器の信号処理アルゴリズム実装(Low Power←→高度な信号処理)
  - > ターゲットプロセッサコア:Cortex-M4(最適化のためコード置換ライブラリ(CRL)適用)
  - > サーバー側のアプリケーション展開
- 効果
  - > MATLABアルゴリズム開発⇒実装の開発効率化を達成

アルゴリズム開発のみで使われていたMATLAB

アルゴリズム開発 — *MATLAB & Toolbox*

Cコード — 手書き、バグ混入

等価性検証
実機動作確認 — 困難



課題・・・

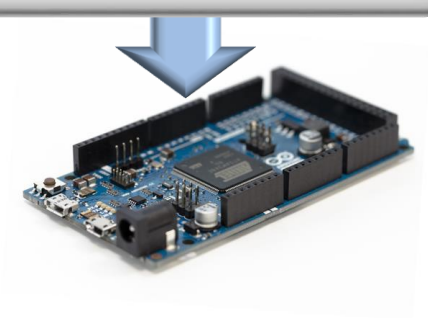
- Cコード手書きに時間がかかり、仕様変更毎の修正が大変
- 等価性の喪失
  ⇒性能低下、バグの原因
- 実装パフォーマンスが最後までわからない

コード生成で効率的かつバグなしで実装

アルゴリズム開発 — *MATLAB & Toolbox*

Cコード — *MATLAB Coder*
GPU Coder

等価性検証と
パフォーマンス解析 — *Embedded Coder*



課題の解決

Cコード自動生成により
実装工数も削減

SIL/PILによる
等価性検証⇒バグなし

実装パフォーマンスを
早期段階で確認

アジェンダ

MATLAB Coderの
基本的な利用方法

効率的なC生成のための
コーディングテクニック

パフォーマンス解析
と最適化

アジェンダ

MATLAB Coderの
基本的な利用方法

効率的なC生成のための
コーディングテクニック

パフォーマンス解析
と最適化

MATLABファイル（信号・画像処理・機械学習など）からCコードを生成

MATLAB Coder

MATLABファイル
（信号処理、通信、オーディオ、画像処理、機械学習などの関数に対応）

MATLAB Coder

.c

.exe

.lib

MEX

コード生成/実装
- PCアプリケーション開発
- 組み込み用Cコード生成
（Embedded Coderが必要）

スタンドアロンの
実行ファイル作成

他のソフトウェアへの
アルゴリズム移植

MATLABファイルの
実行形式化

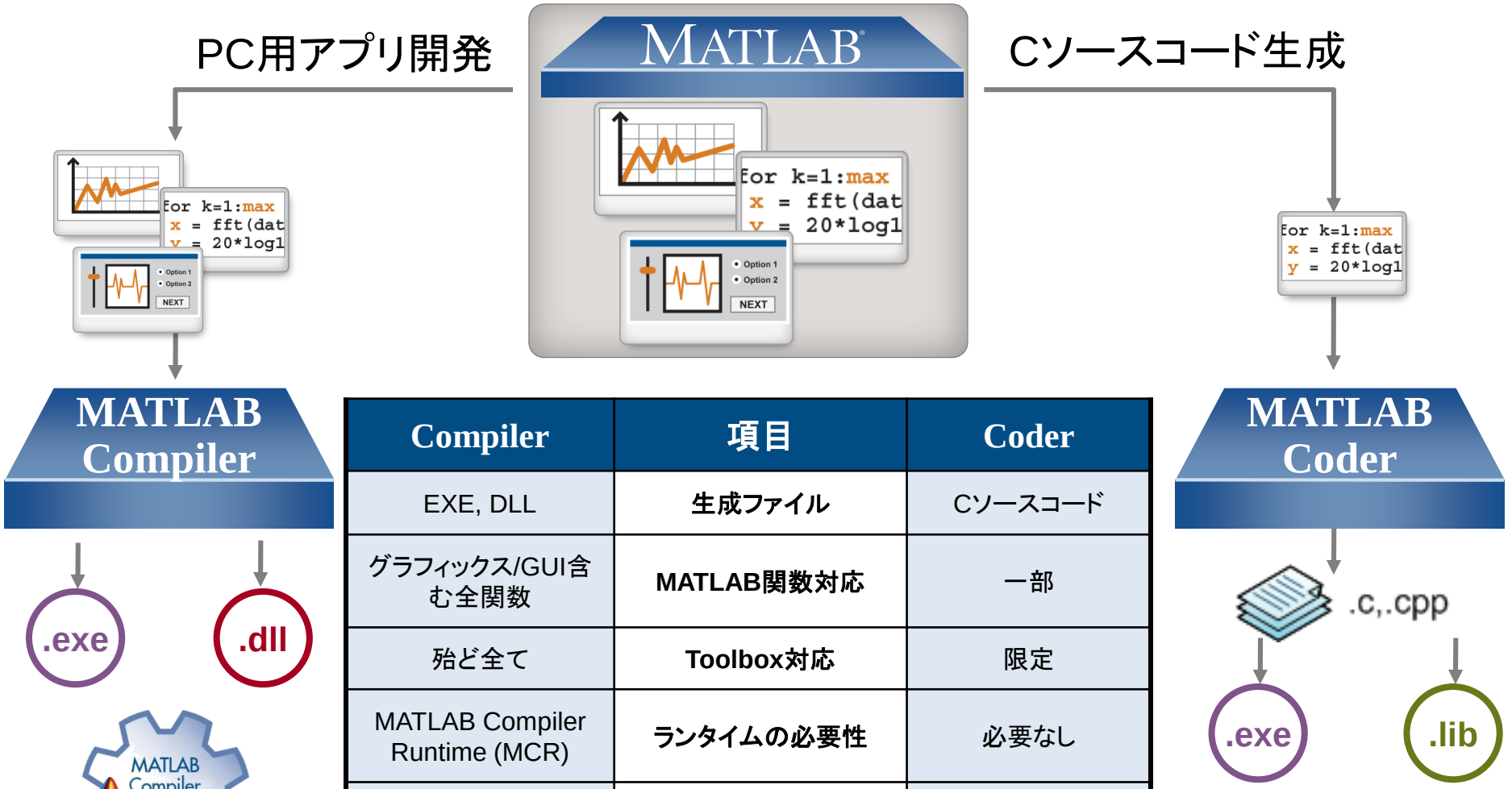
GPU Coder™を使ったディープラーニングの実装

MATLAB
学習済みネットワーク

MATLAB Coder
+
GPU Coder

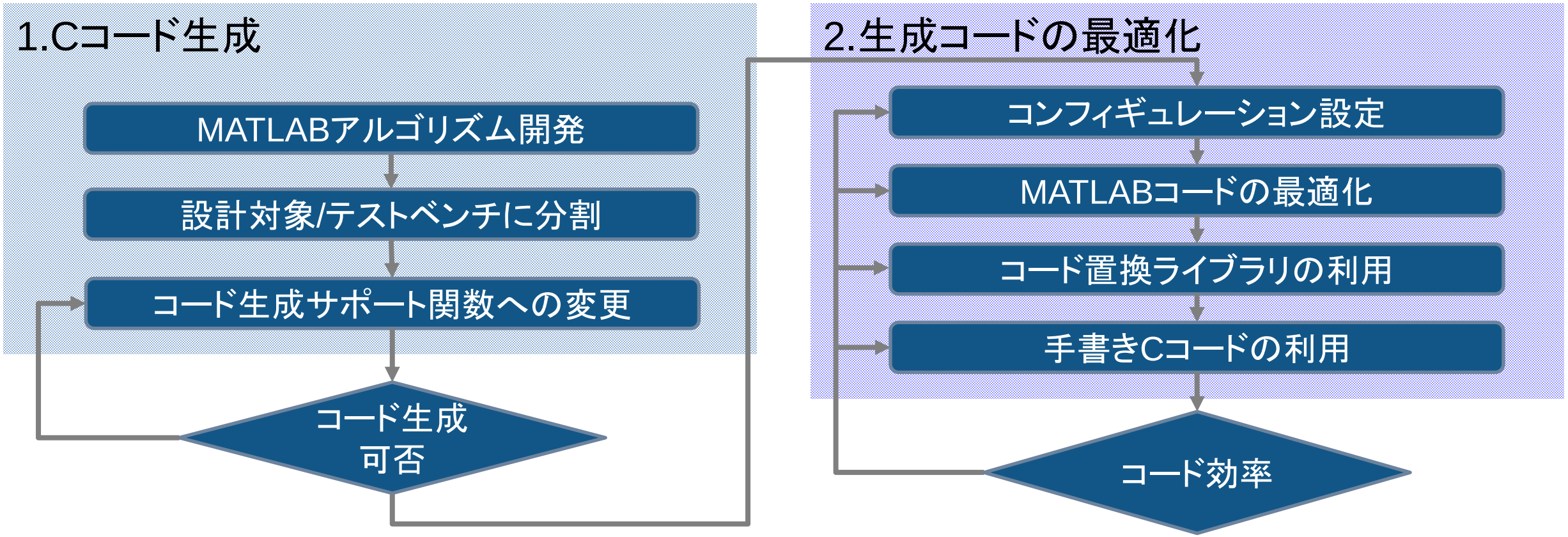
- cuDNN → NVIDIA社GPU
- TensorRT → NVIDIA社GPU
- MKL-DNN → Intel社Xeon, Xeon Phi
- ARM Compute → ARM社Mali, Cortex-A

MATLAB CompilerとMATLAB Coderの違い



Compiler	項目	Coder
EXE, DLL	生成ファイル	Cソースコード
グラフィックス/GUI含む全関数	MATLAB関数対応	一部
殆ど全て	Toolbox対応	限定
MATLAB Compiler Runtime (MCR)	ランタイムの必要性	必要なし
PC	生成物の実行環境	PC, 組込み
ロイヤルティフリー	生成物のライセンス	ロイヤルティフリー

MATLABプログラムからのCコード生成ワークフロー概要



MATLABコードの構成は
Cコード生成対象をFunctionにして、テストベンチを用意

オリジナル
（信号生成+設計対象+可視化）

テストベンチ
- 記述内容
  - 信号生成
  - 生成対象呼び出し
  - 可視化・解析
- コードの制限
  - なし

生成対象
- 記述内容
  - 引数有りの function MATLABファイル
- コードの制限
  - コード生成対応関数・文法

Cソースコード

シミュレーション用コードとコード生成用コードの切り替え

目的ごとに使用するMATLABコードを切り替え可能

```
function a = switch_fcn(r)    %#codegen
a=ones(10,256);

if coder.target('Rtw')
    % for code generation
    out = myCodeGenFcn(input);
else
    % for MATLAB and etc.
    out = mySimFcn(input);
end
```

Cコード生成用
最適化コード

シミュレーション用
リファレンス

オプション	ターゲット
MATLAB	MATLABでの実行
MEX	MEX関数生成時
Sfun	Simulinkシミュレーション
Rtw	LIB, DLL, EXE生成時
HDL	HDL生成時
Custom	カスタムターゲットの生成

コード生成用プロジェクトの作成

デモ

- アプリとコマンドでコード生成の設定および実行
  - アプリ：初心者、設定に慣れない人向け➔アプリメニューまたは `>> coder , >> gpucoder`
  - コマンド：経験者、設定が固まっている人向け `>> codegen filename -options`

詳細設定画面

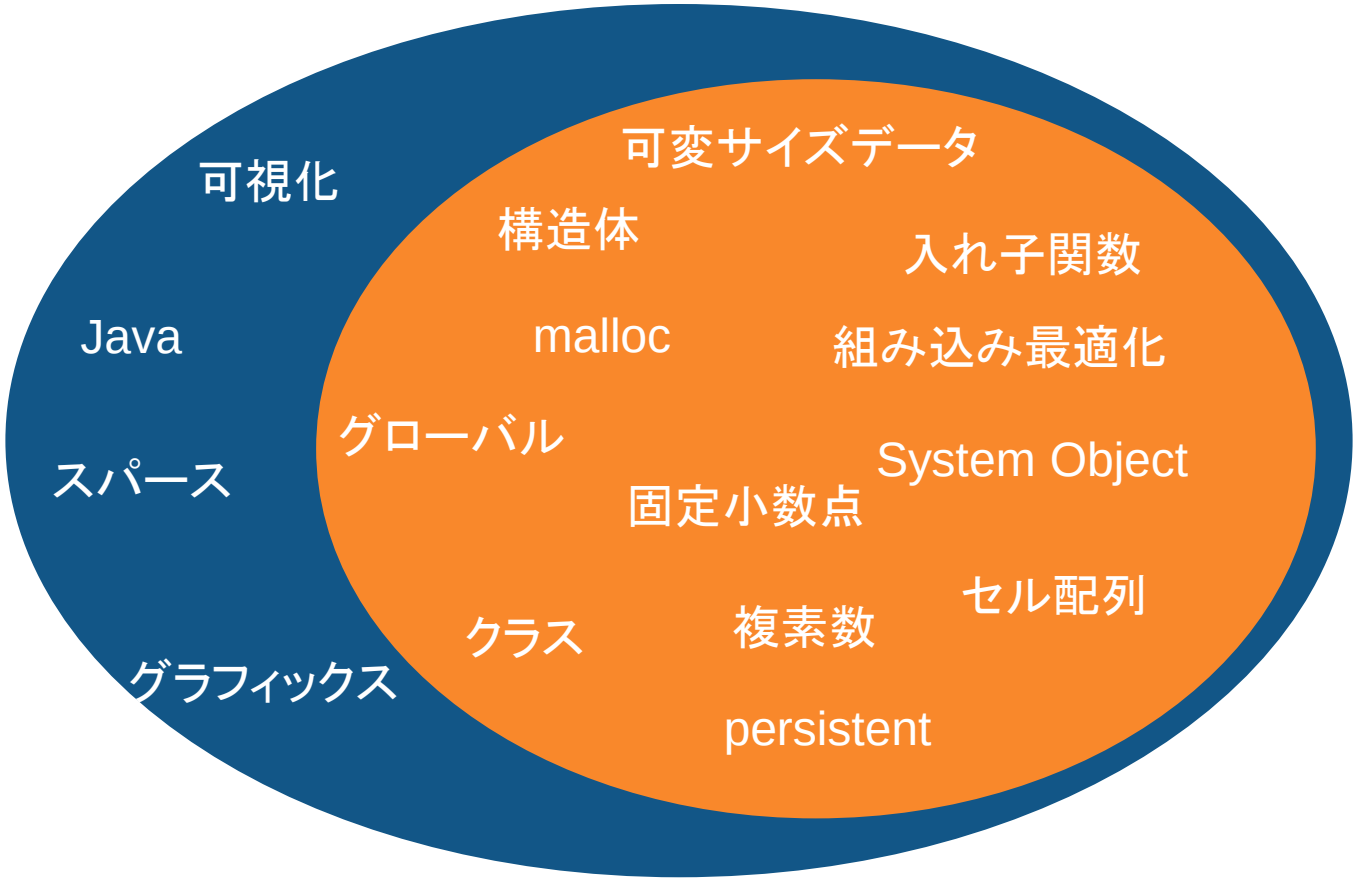
コード生成に対応するMATLAB関数

ドキュメントの対応関数一覧

- *MATLAB Coder*
  >コード生成のためのMATLABプログラミング
  >言語、関数、オブジェクトのサポート
- *GPU Coder*
  >MATLAB Algorithm Design for GPU

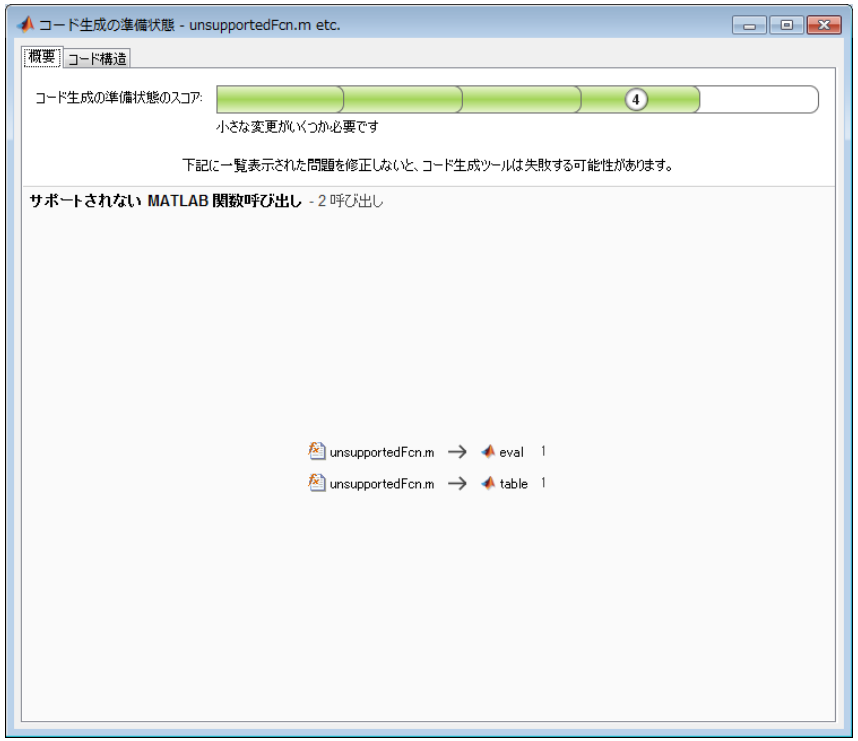
Webドキュメント

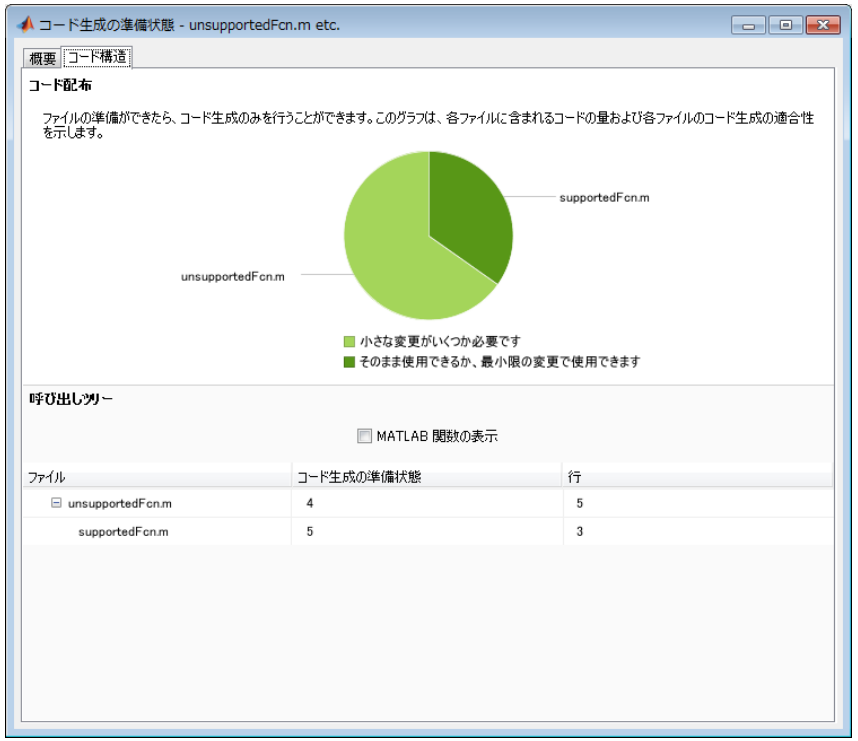
- *MATLAB Coder*
  http://www.mathworks.com/help/coder/language-supported-for-code-generation.html
- *GPU Coder*
  https://www.mathworks.com/help/gpucoder/algorithm-design.html



コード生成対応の事前簡易チェック

- coder.screenerによるコードのチェック
  - >> coder.screener('filename') により、MATLAB Coderアプリ実行前に確認可能
  - ユーザ関数ごとにチェックして問題箇所の表示

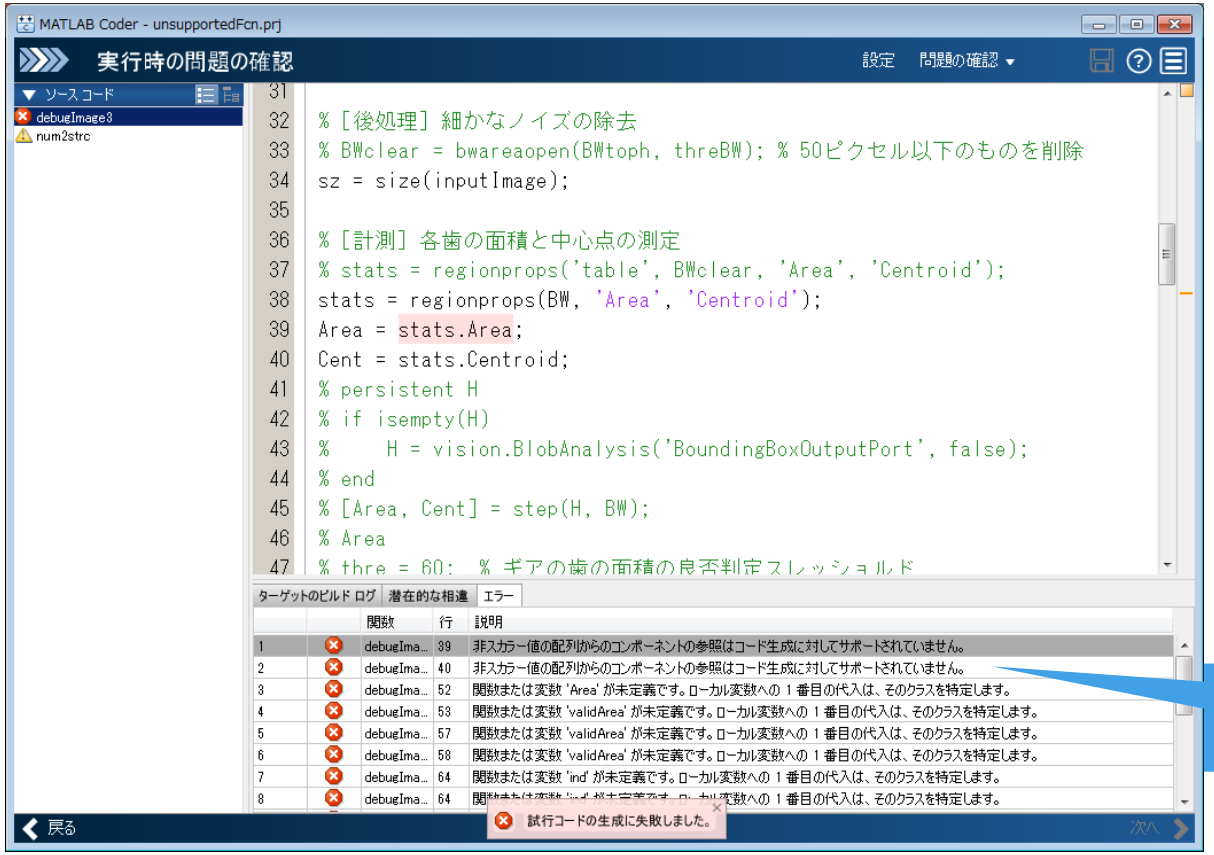




coder.screenerの確認結果例

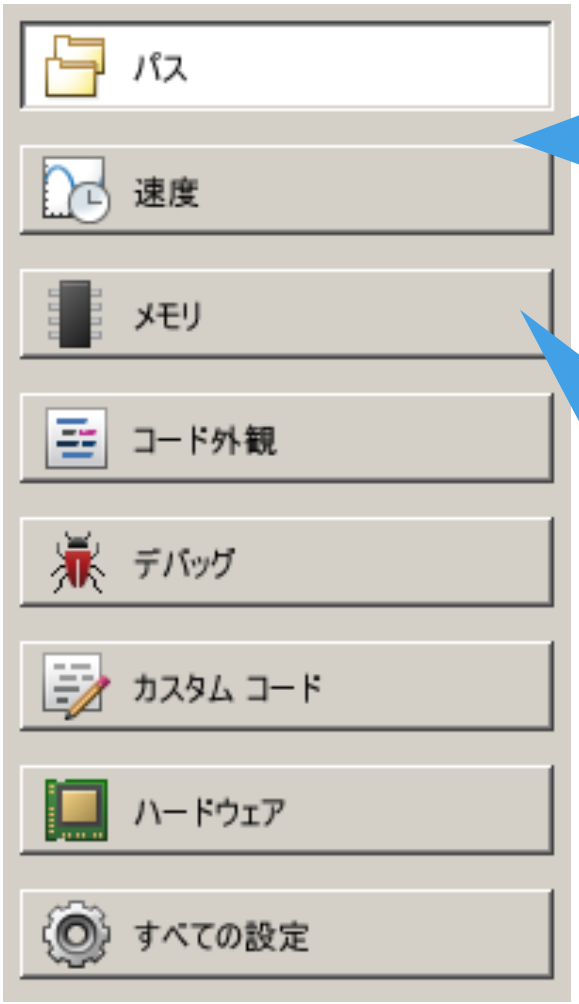
コード生成対応の詳細チェック方法

- Coderアプリの「実行時の問題の確認」画面でチェック
  - 未対応関数、文法の検出
  - 該当行と説明のレポート表示

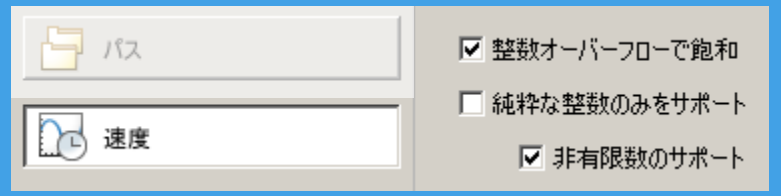


エラー内容の表示

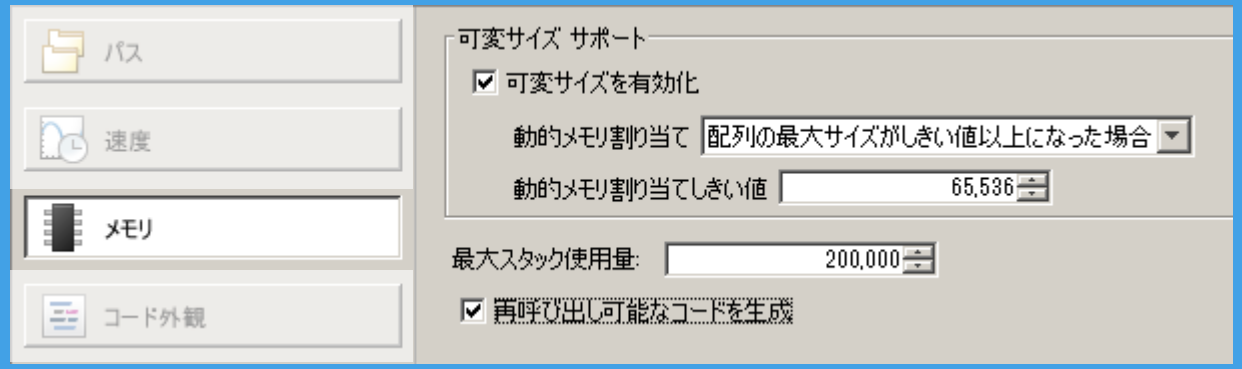
コード生成/詳細設定の例（*MATLAB Coder*）



- 「整数オーバー･･･」を有効化すると飽和処理が追加され、速度低下
- 固定小数点プロセッサ向けは浮動小数点コードを生成しないよう「純粋な整数･･･」を有効化（Embedded Coderのオプション）



- 「可変サイズ･･･」有効化➔動的メモリ割り当てにより大きなデータセットではスタックオーバーフローの危険性、パフォーマンス低下
- 「再呼び出し･･･」を有効にすると、関数の引数によってデータ受取り、複数のプロセスから呼び出し可能



基本利用におけるTips

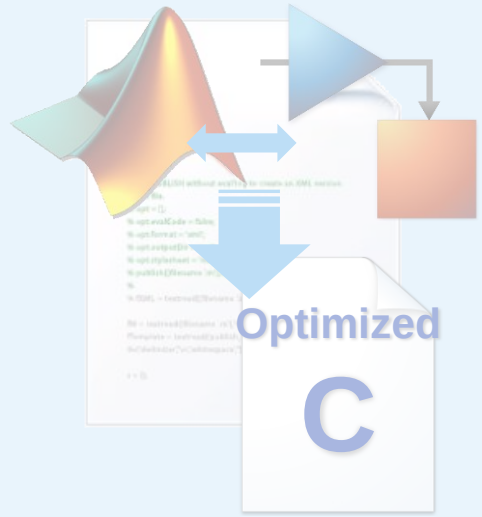
- %#codegenを付けることでエディターのコードアナライザーを有効化
- plot, disp, figureなどの表示系関数はCコード生成非対応、
  Cコード生成時には無視されるので削除不要
- 非対応関数はcoder.extrinsic('func')を使ってコード生成無効化
  （コード生成準備段階で問題がある関数を除外したいときにも便利）
- 生成対象コードにassert命令を入れることで、データ型や行列サイズを指定
  例：
  - assert(isa(param,'single'));　　% パラメータはsingle型
  - assert(all(size(param) == [3, 4]));　　% 行列サイズは3x4
  - assert(isscalar(param));　　% スカラー

アジェンダ

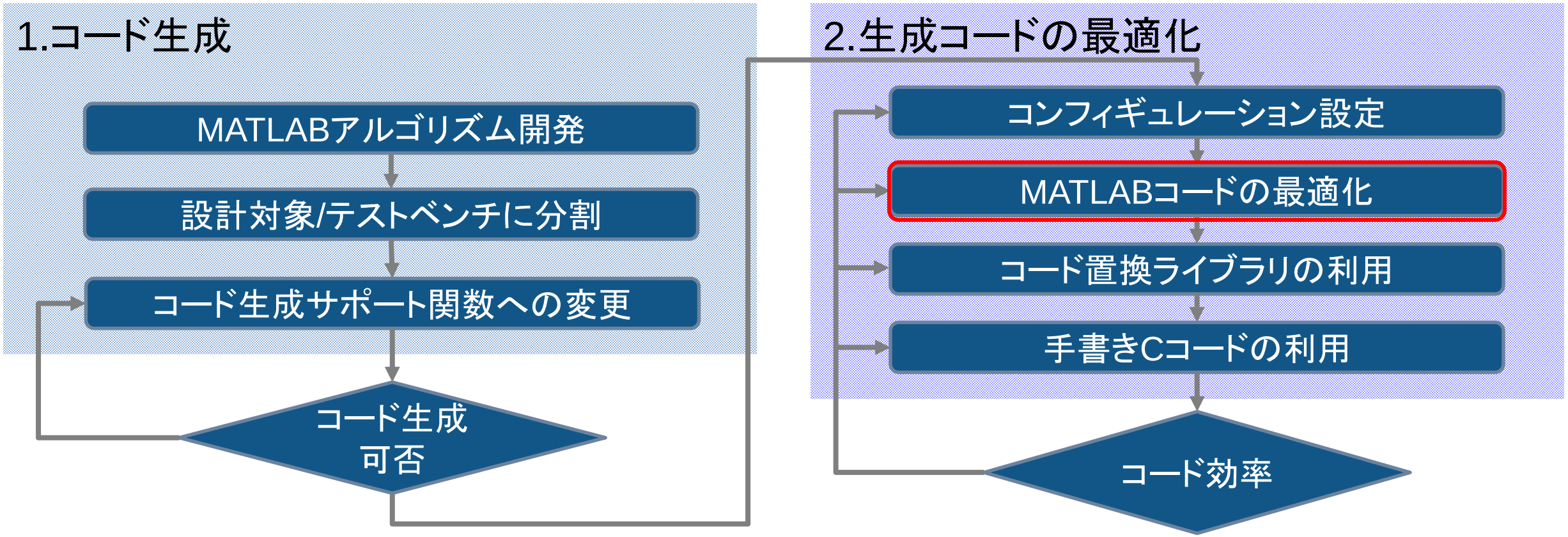
MATLAB Coderの
基本的な利用方法

効率的なC生成のための
コーディングテクニック

パフォーマンス解析
と最適化



MATLABプログラムからのCコード生成ワークフロー概要



なぜMATLABコードの最適化が必要なのか？

MATLABコードは
多様な入出力に対応する言語

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

- 要素ごとの演算
- 内積演算
- 行列演算

整数/浮動小数点数
/固定小数点数
実数/複素数
…

```
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];
      }
    }
  }
}
```

Cコードでは表現が異なる

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

MATLABコードの最適化
ループ内演算の最小化

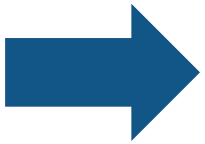
- ループ内の冗長演算の最小化➔Cコード実行速度の向上

改良前

```
function C=SeriesFunc(A,B,n)

C=zeros(size(A));

for i=1:n
    C=C+inv(B)*A^i*B;
end
```



改良後

```
function C=SeriesFunc(A,B,n)

C=zeros(size(A));

Bi = inv(B);

for i=1:n
    C = C+Bi*A^i*B;
end
```

ループ変数に依存しないinv(B)の処理をループ外に移動

MATLABコードの最適化
未実行分岐パスや変数の削減(coder.Constant)

- 未実行分岐パスの削減➔Cコード実行速度の向上
- 変数の削減➔メモリの削減

```
>> codegen MF -args {coder.Constant(1), coder.Constant(4), zeros(10,1)}
```

```
function out = MF(flag, gain, in)
%# codegen
switch flag
    case 1
        out = gain * sin(in);
    case 2
        out = gain * cos(in);
    otherwise
        out = gain * sqrt(in);
end
```

入力引数flag, gainが**変数**だと
生成コードでも変数として処理

入力引数flag, gainが**定数**だと
分岐の削除、定数のdefine定義

```
#define gain      (4.0)

void MF(const double in[10], double out[10])
{
  int k;
  for (k = 0; k < 10; k++) {
    out[k] = gain * sin(in[k]);
  }
}
```

MATLABコードの最適化
OpenMPを使ったマルチスレッドコード生成

- マルチスレッド化➔処理速度向上

改良前

```
function a = test_for(r)    %#codegen
a=ones(10,256);

for i=1:10
    a(i,:)=real(fft(r(i,:)));
end
```

改良後

```
function a = test_for(r)    %#codegen
a=ones(10,256);

parfor i=1:10
    a(i,:)=real(fft(r(i,:)));
end
```

forを並列実行するparfor

```
void test_for(const double r[2560], double
a[2560])
{
  ....
#pragma omp parallel for ¥
 num_threads(omp_get_max_threads()) ¥
 private(i0) ¥
 firstprivate(dcv0,b_r)

  for (i = 0; i < 10; i++) {
  .....
}
```

OpenMPのプラグマ追加

※ parforはParallel Computing Toolboxの並列演算用のコマンド
※ OpenMPはC/C++並列コンピューティング用API

MATLABコードの最適化
関数入力のデータコピーの削減

- 変数の再利用➔RAM使用量の削減、パフォーマンス向上

改良前

```
function y = foo( A, B )
%#codegen

y = A * B;

end
```

```
double foo(double A, double B)
{
  double y;
  y = A * B;
  return y;
}
```

改良後

```
function A = foo( A, B )
%#codegen

A = A * B;

end
```

```
void foo(double *A, double B)
{
      *A *= B;
}
```

参照渡し➔メモリ使用量の削減

MATLABコードの最適化
coder.nullcopyによる変数初期化

- coder.nullcopyによる変数初期化➔不要な初期化の削減、パフォーマンス向上

改良前

```
function X = fcn(I) %#codegen
N = size(I);
X = zeros(N);
for i = 1:numel(N)
    if mod(i,2) == 0
        X(i) = i;
    elseif mod(i,2) == 1 ....
```

Xを初期化しないと代入できない

```
void fcn(const double I[64000], double X[64000])
{
  int i;
  (void)I;
  memset(&X[0], 0, 64000U * sizeof(double));
  for (i = 0; i < 2; i++) {
    if (b_mod(1.0 + (double)i) == 0.0) {
      X[i] = 1.0 + (double)i;
    } else { ....
```

改良後

```
function X = fcn(I) %#codegen
N = size(I);
X = coder.nullcopy(zeros(N));
for i = 1:numel(N)
    if mod(i,2) == 0
        X(i) = i;
    elseif mod(i,2) == 1 ....
```

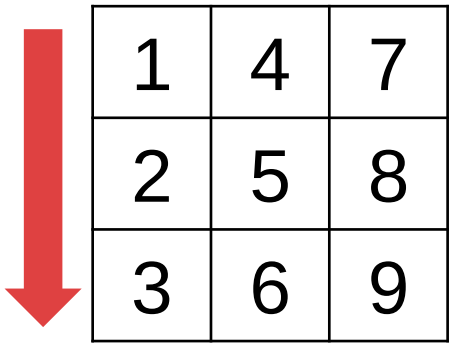
```
void fnc(const double I[64000], double X[64000])
{
  int i;
  (void)I;
  for (i = 0; i < 2; i++) {
    if (b_mod(1.0 + (double)i) == 0.0) {
      X[i] = 1.0 + (double)i;
    } else {  ....
```

不要な初期化の削減

行列データの取扱

画像処理などで多次元配列の計算

```
function z = addMatrix(x, y) %#codegen
z = coder.nullcopy(x);
% 行列の加算
for row = 1:size(x,1)   % 行
    for col = 1:size(x,2) % 列=各行の要素
        z(row,col) = x(row,col) + y(row,col);
    end
end
```



1	4	7
2	5	8
3	6	9

列方向インデックス

1	2	3
4	5	6
7	8	9

行方向インデックス

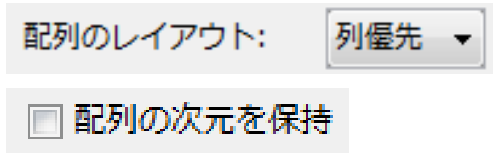
行優先、N次元インデックスオプション（R2018a~）

```
function z = addMatrix(x, y) %#codegen
z = coder.nullcopy(x);
% 行列の加算
for row = 1:size(x,1)              % 行
    for col = 1:size(x,2) % 列=各行の要素
        z(row,col) = x(row,col) + y(row,col);
    end
end
```

生成

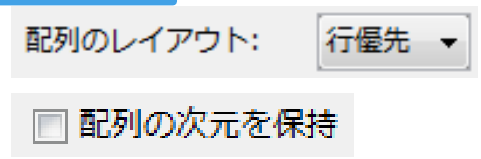
- 2次元配列の行方向優先インデックス
  →キャッシュ効率化により速度向上
  →既存Cコードとの統合が容易
- N次元インデックス
  →可読性向上

デフォルト設定



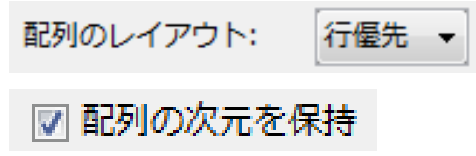

```
for (row = 0; row < 20; row++) {
   for (col = 0; col < 10; col++) {
      z[row+20*col] = x[row+20*col] + y[row+20*col];
...
```

```
>> codegen -rowmajor
```




```
for (row = 0; row < 20; row++) {
   for (col = 0; col < 10; col++) {
      z[col+20*row] = x[col+20*row] + y[col+20*row];
...
```

```
>> codegen -rowmajor -preservearraydims
```



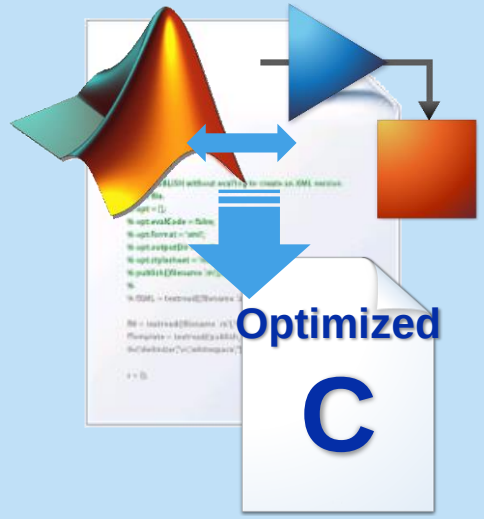

```
for (row = 0; row < 20; row++) {
   for (col = 0; col < 10; col++) {
      z[row][col] = x [row][col] + y [row][col];
...
```

アジェンダ

MATLAB Coderの
基本的な利用方法

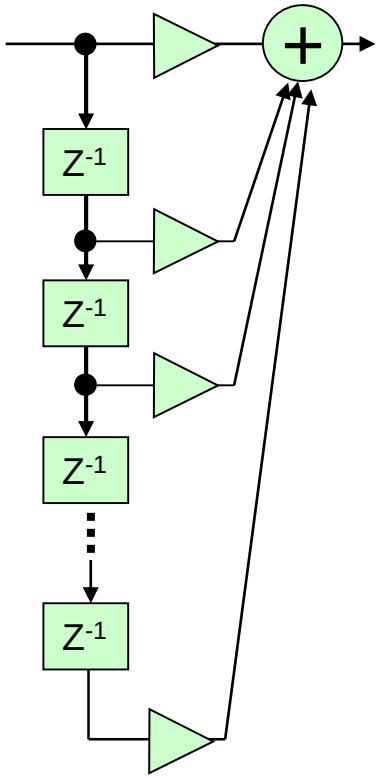
効率的なC生成のための
コーディングテクニック

パフォーマンス解析
と最適化

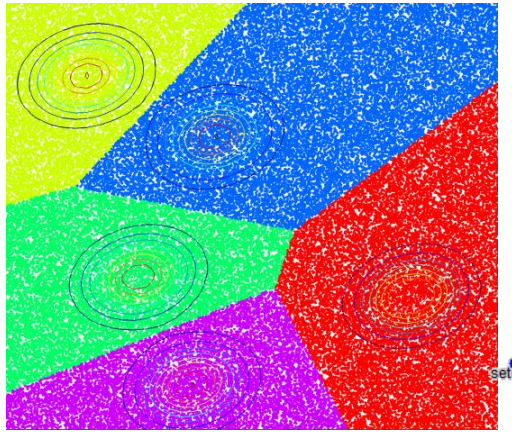


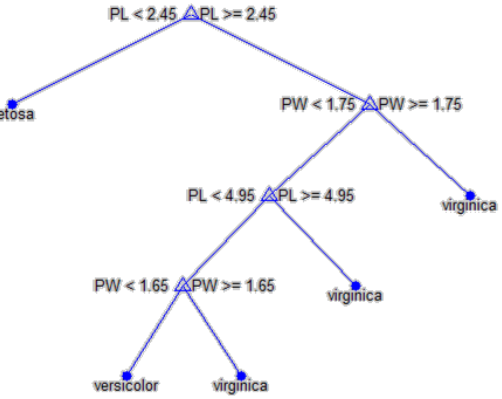
処理負荷と性能の関係

積和演算ベースの処理
（1D/2Dフィルタ, FFTなど）
⇒性能は処理におおよそ比例



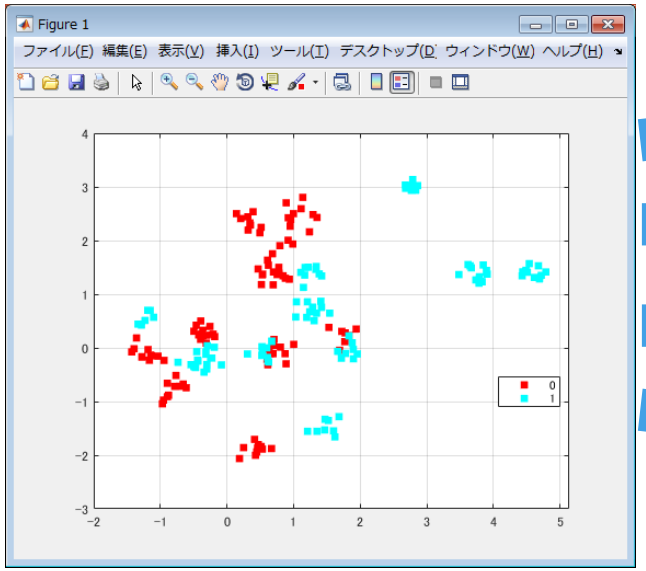
機械学習の予測
（KNN, 決定木, SVM, Neural）
⇒性能が処理に比例しない



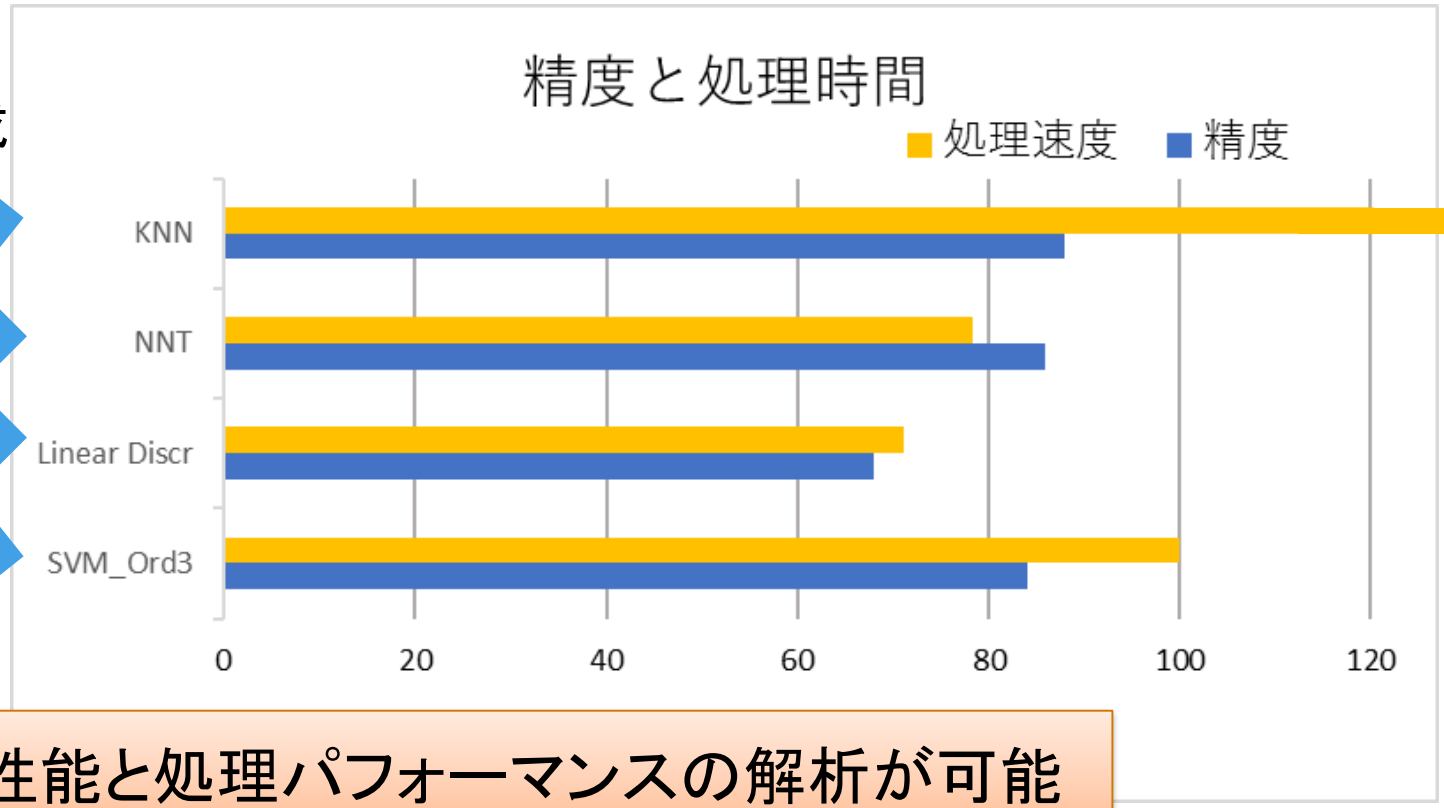


機械学習の実装は分類精度と処理時間が必ずしも比例しない

- 同じデータ/異なるアルゴリズムの分類器からCコード生成
  ⇒PILにより処理時間を測定（QEMU Cortex-A エミュレータを使用）



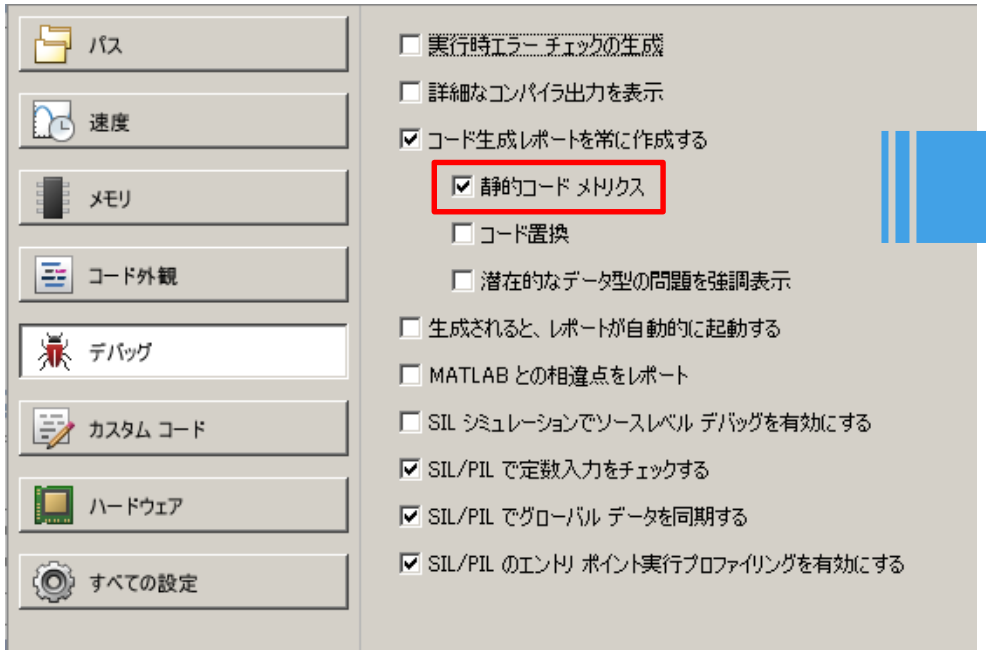
Cコード生成

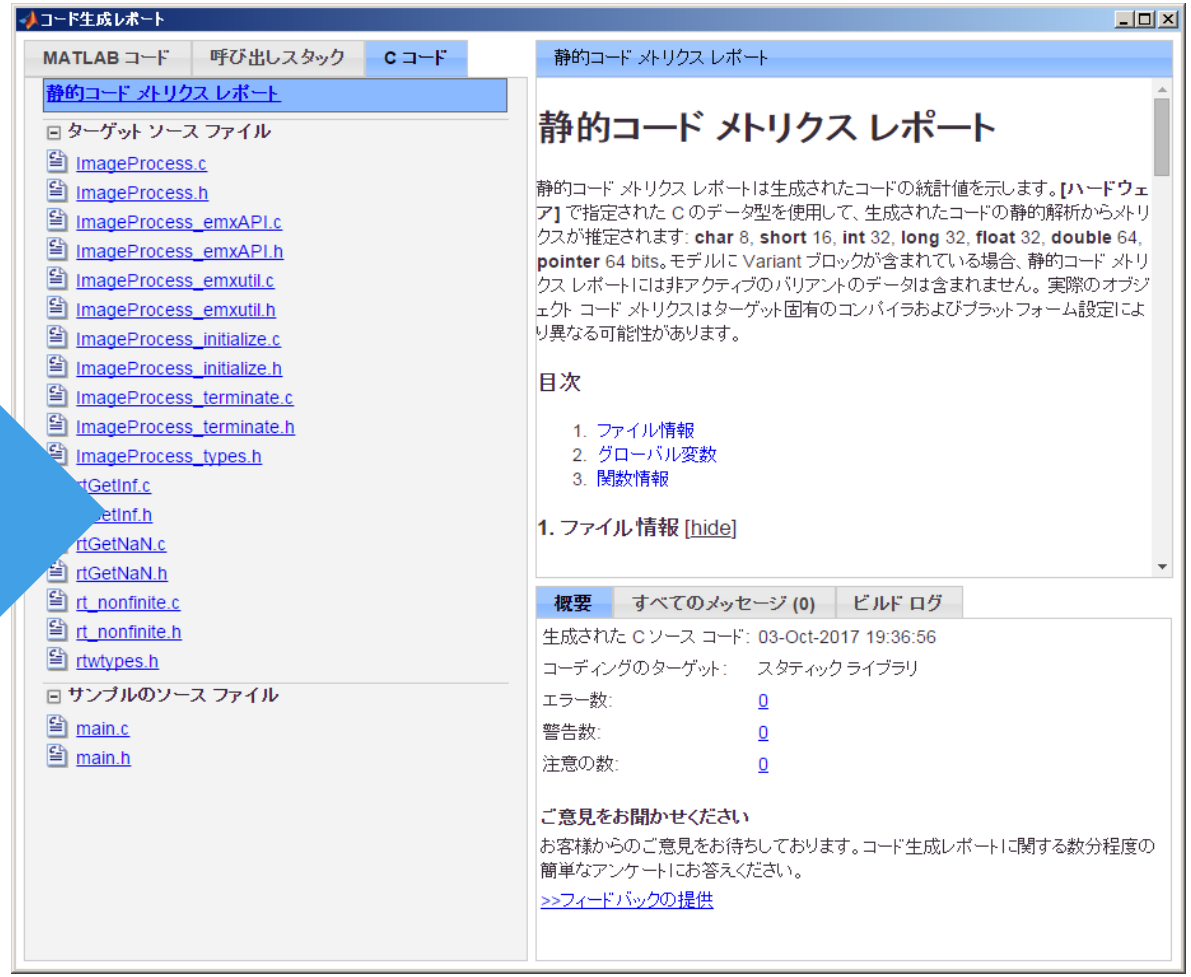


早期にアルゴリズムの性能と処理パフォーマンスの解析が可能

コードのパフォーマンス解析
コード生成レポート/静的コードメトリクス

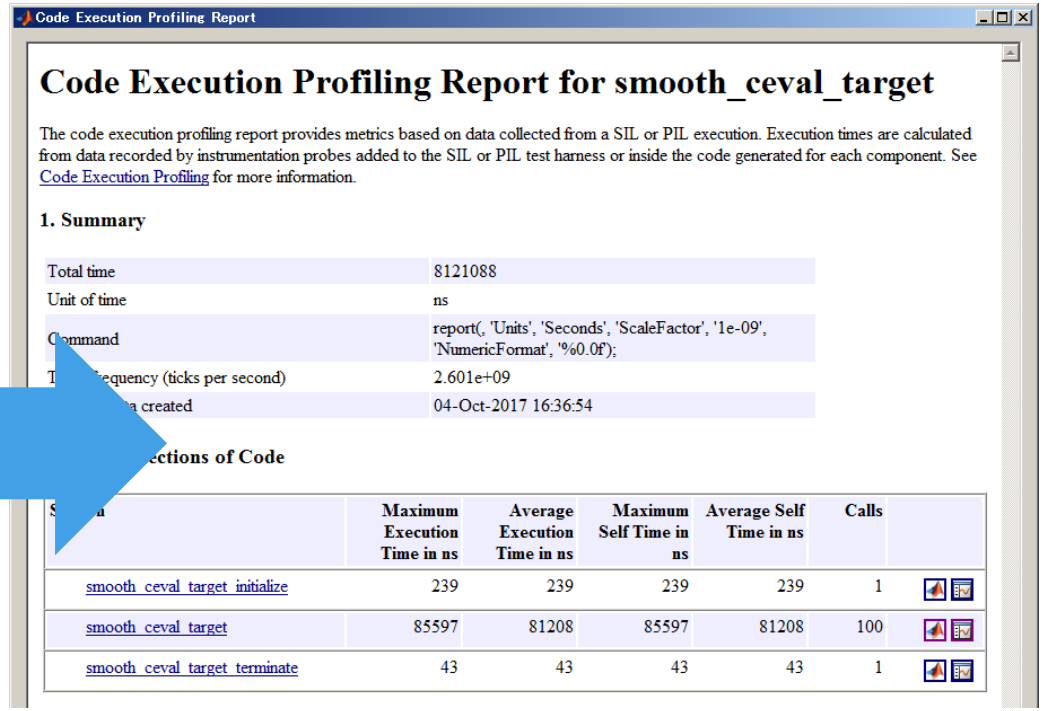
- ファイル情報
  - ファイルごとのコード行数
  - グローバル変数、サイズ（バイト）
  - スタックサイズ





等価性検証と実行時間測定
SIL/PIL検証（*Embedded Coder*機能）

- コード検証機能の種類
  - SIL(Software In the Loop) : 生成CコードをPC上で実行
  - PIL(Processor In the Loop) : 生成Cコードを実機/エミュレータ（QEMU）で実行
- コード検証の目的
  - 等価性検証 : MATLABコード ⇔ Cコード
  - 生成コードの実行時間測定



コード置換ライブラリ(Code Replacement Library=CRL)とは？

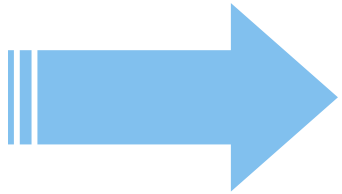
- ターゲットプロセッサ向けに最適化されたコードに置換機能 （Embedded Coderが必要）
- サポートプロセッサ：ARM Cortex-M/A, Intel x86(IPP), TI C55x/C62x/C64x/ C67xなど
- 四則演算、数学演算、信号処理演算などに対応（ターゲットプロセッサによって異なる）
  信号処理演算はSystem Objectが対応
- ユーザ登録が可能 **>> crtool**

MATLABプログラム

```
function [y1, y2, y3, y4

y1 = u1 + u2;
y2 = u1 - u2;
y3 = u1 .* u2;
y4 = sin(u3);
y5 = cos(u3);
y6 = sqrt(u3);
```

```
persistent h;
if isempty(h)
    h = dsp.FIRFilter('Numerator', fir1(63, 0.33));
end
y1 = step(h, u1);
```



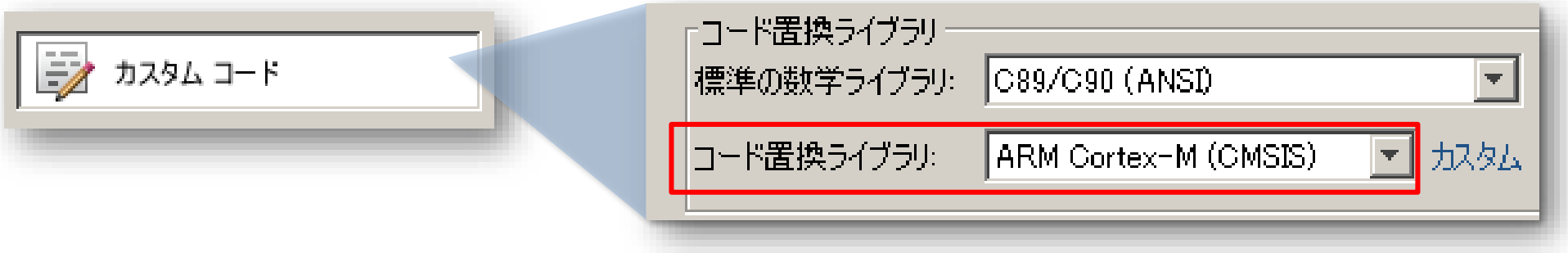
生成されたCortex-M用Cコード

```
void arm_EC_ops(const float u1[2], const floa
                float y2[2], float y3[2], flo
{
  mw_arm_add_f32(u1, u2, &b_y1[0], 2U);
  mw_arm_sub_f32(u1, u2, &y2[0], 2U);
  mw_arm_mult_f32(u1, u2, &y3[0], 2U);
  *y4 = arm_sin_f32(u3);
  *y5 = arm_cos_f32(u3);
  mw_arm_sqrt_f32(u3, y6);
```

```
/* System object Outputs function: dsp.FIRFilter */
arm_fir_f32(&b_obj->S, &U0[0], &b_y1[0], 75U);
```

コード置換ライブラリ適用方法

1. 演算（数式, System Object）、プロパティ、データ型、丸め条件を設定
（置換条件はサポートパッケージの各ドキュメントに記載）
2. MATLAB Coderの詳細設定/カスタムコード

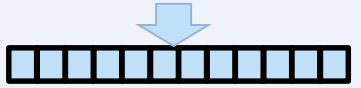


3. 置換の成功可否をレポートから確認

System Objectとは？

- 動的システムのストリーミング処理に特化したMATLABクラス
  - フィルタなどの遅延要素で必要な内部状態の保存に面倒な手続きが不要
- 共通のインターフェースを持つ（setup/reset/step/releaseメソッド）

	System Object	MATLAB関数
処理形態	ストリーミング	バッチ
メモリ消費	小	大
Simulink対応	ほぼ全て	一部
状態の管理	簡単	ユーザ設定
Cコード生成	ほぼ全て、**コード置換ライブラリ**	一部

処理イメージ	`FIR = dsp.FIRFilter;`
	 
	`for time = 1:10`
	`  out = FIR(in)`
	`end`
	 
	細切れデータを逐次処理
		
		`out = filter(b,a,in)`
		 
		全データを一度に処理

System Object詳細はMATLAB EXPO 2016講演資料参照
http://www.matlabexpo.com/jp/2016/proceedings/g3-system-object.pdf

信号処理演算のコード置換ライブラリに対応するSystem Object
CMSIS/Ne10ライブラリサポート

System Object	Cortex-M CMSIS	Cortex-A Ne10
dsp.FIRFilter	✓	✓
dsp.FIRDecimator	✓	✓
dsp.FIRInterpolator	✓	✓
dsp.LMSFilter	✓	
dsp.BiquadFilter	✓	
dsp.FFT	✓	✓
dsp.IFFT	✓	✓
dsp.Convolver	✓	
dsp.Crosscorrelator	✓	
dsp.Mean	✓	
dsp.RMS	✓	
dsp.StandardDeviation	✓	
dsp.Variance	✓	

System Object	Cortex-M CMSIS	Cortex-A Ne10
dsp.VariableBandwidthFIRFilter	✓	✓
dsp.FIRHalfbandInterpolator	✓	✓
dsp.FIRHalfbandDecimator	✓	✓
dsp.CICCompensationDecimator	✓	✓
dsp.CICCompensationInterpolator	✓	✓
dsp.DigitalDownConverter	✓	✓
dsp.DigitalUpConverter	✓	✓
dsp.SampleRateConverter	✓	✓

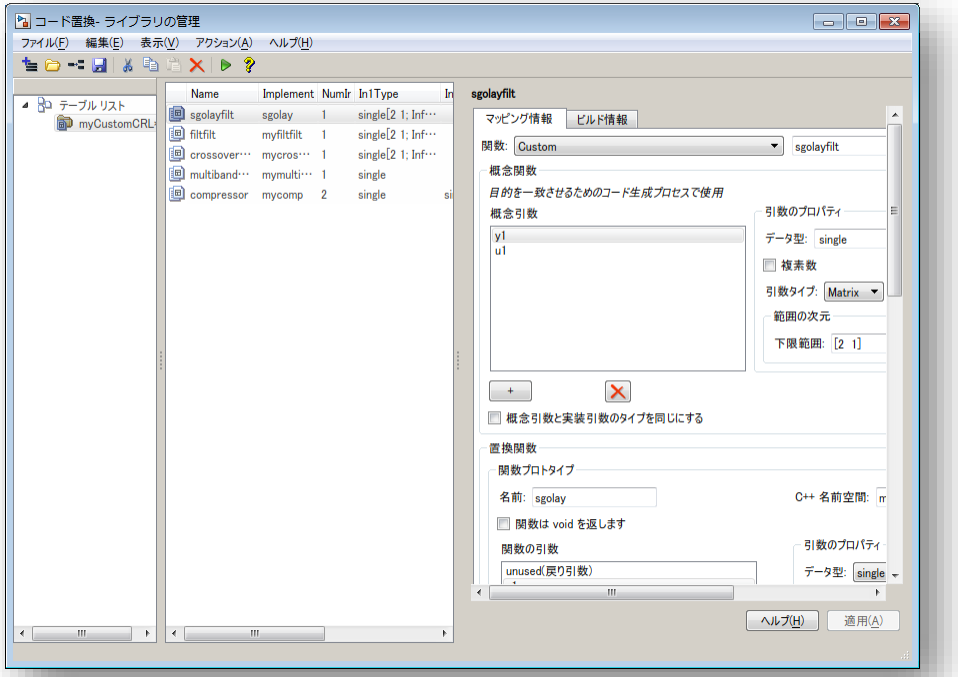
コード置換ライブラリー覧の表示

```
>> crviewer
```

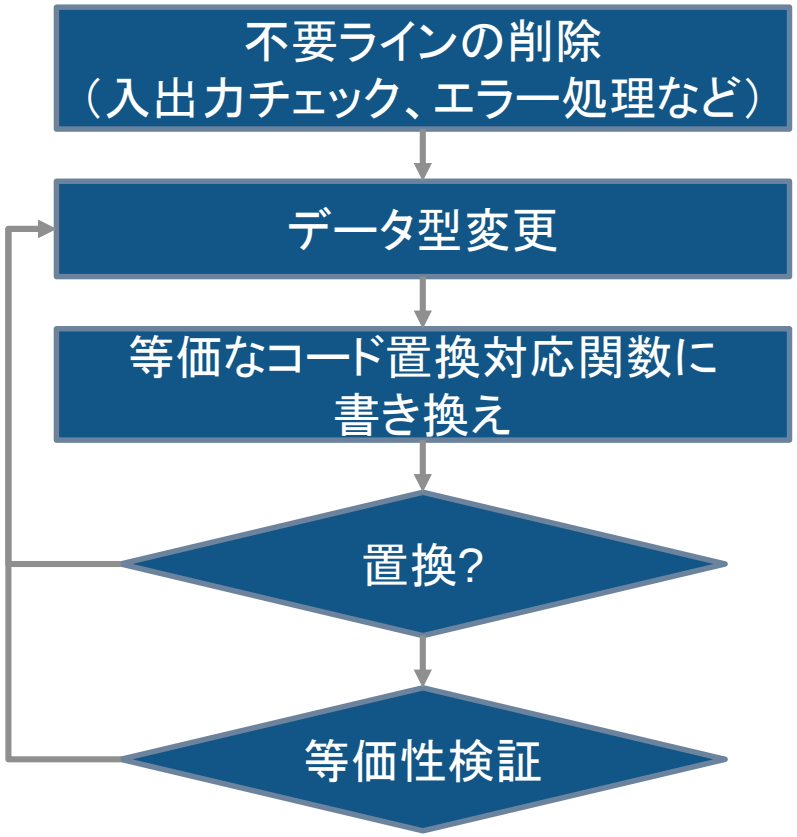
コード置換未対応の関数を対応させるには？

コード置換ライブラリのカスタマイズ
>> crtool

- MATLAB関数と対応するC関数をライブラリ登録
  - 入力/出力、データ型、次元の範囲など



MATLAB関数をカスタマイズ



まとめ

MATLAB Coderの
基本的な利用方法

MATLABコードの
コーディングテクニック

パフォーマンス解析
と最適化

MATLAB Coder利用に役立つリソース（1/2）

- Embedded Coderのオプション
  https://jp.mathworks.com/help/ecoder/gs/about-the-tutorials-1.html#buildlm
- クイックスタートガイド
  https://jp.mathworks.com/campaigns/products/offer/download_matlab-coder.html
- MATLABコードを固定小数点化するコツ（英語）
  https://jp.mathworks.com/company/newsletters/articles/best-practices-for-converting-matlab-code-to-fixed-point.html
- MATLAB and C/C++ Resources（英語）
  https://jp.mathworks.com/campaigns/portals/matlab-coder.html
- コード置換ライブラリ非対応関数をコード置換する方法
  https://jp.mathworks.com/matlabcentral/answers/384122-

MATLAB Coder利用に役立つリソース（2/2）

- System Objectとは？
  https://www.mathworks.com/help/matlab/matlab_prog/what-are-system-objects.html
  http://www.matlabexpo.com/jp/2016/proceedings/g3-system-object.pdf

- System Objectによるデバイスドライバブロックの作成（英語）
  https://jp.mathworks.com/help/supportpkg/raspberrypi/device-driver-blocks.html
- http://www.matlabexpo.com/jp/2017/proceedings/f5-matlab-coder.pdf

- トレーニングコース：MATLAB CoderによるCコード生成
  https://jp.mathworks.com/training-schedule/matlab-to-c-with-matlab-coder.html