Trip Optimizer

Development of a Driver Assistance System for Locomotives Using MATLAB

8 May 2017

MathWorks Automotive Conference | May 9th 2017
GE Proprietary

Agenda

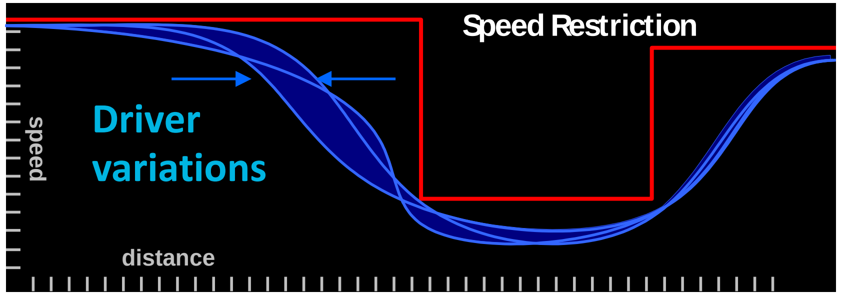
1. Trip Optimizer Overview
2. Trip Optimizer and MATLAB
3. Integrating an external optimization library into MATLAB Code Generation Toolset

Trip Optimizer Overview

GE's Trip Optimizer
"Fuel conscious cruise control for trains"

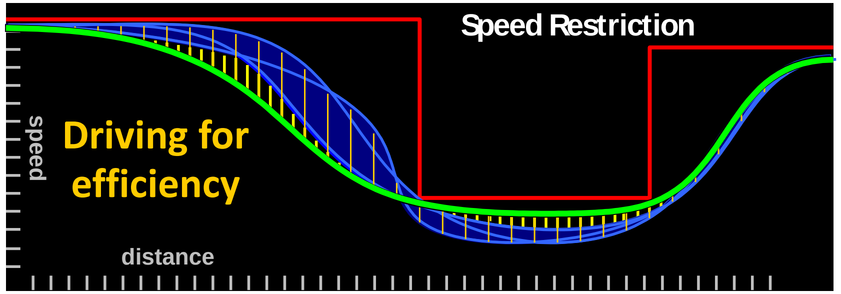
Train and driver variations result in:

- Less than optimal fuel use
- High emissions
- Trip variations
- Wear and tear



Trip Optimizer:

- **Looks over the entire route for fuel savings opportunities**
- **Then controls the throttle to the plan**
  - Saves fuel
  - Reduces emissions
  - Reduces equipment wear and tear
  - Consistent trips improve scheduling



Trip Optimizer Deployment/Operation

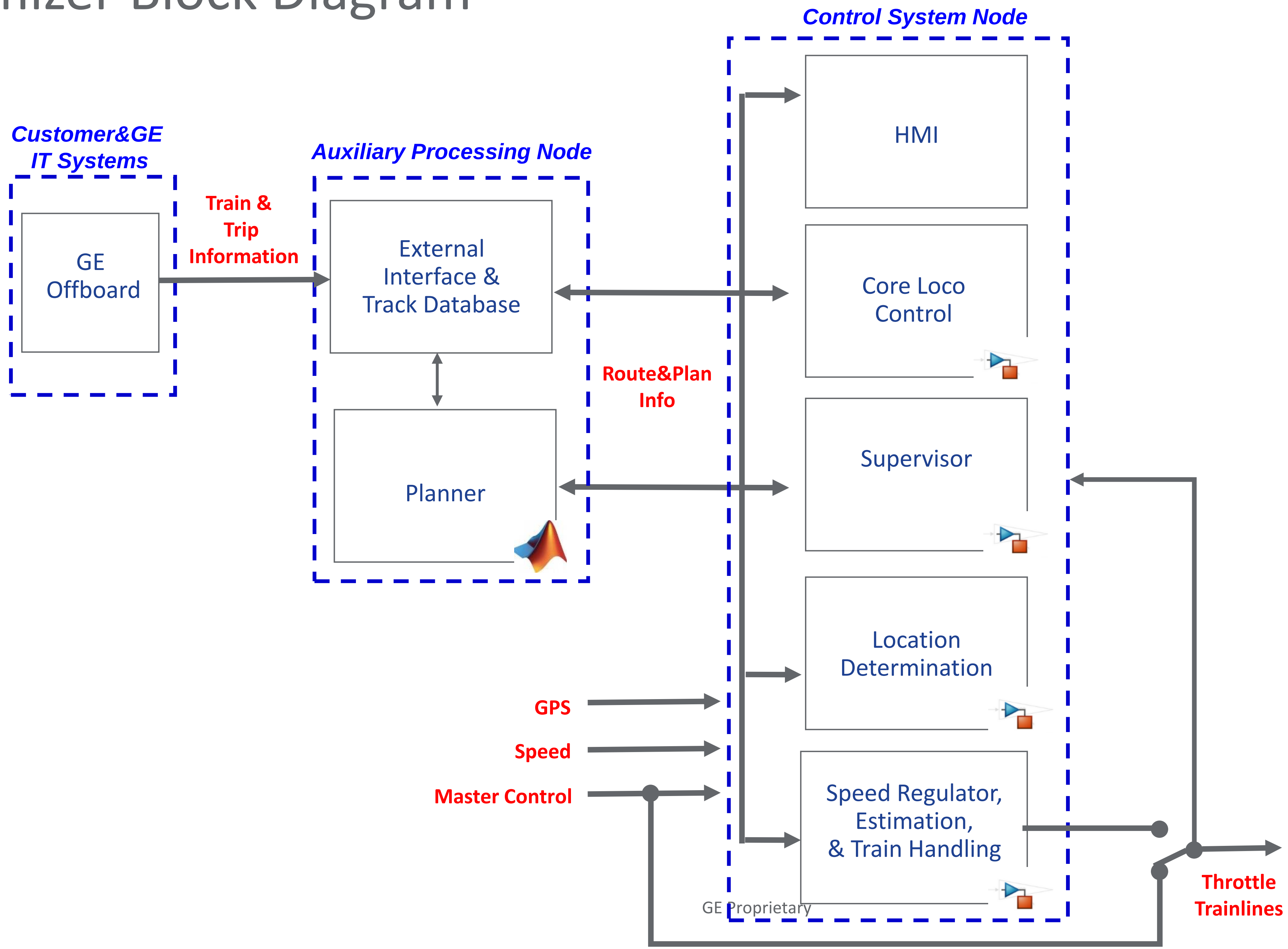
- 8,000 systems installed world wide
- 60,000 miles of mapped track
- 216M miles of auto operation
- 73M miles of auto in 2016
- 1.7M auto miles per week
- 142,000 gallons of diesel fuel saved per week

Trip Optimizer and MATLAB

Converging Technologies

- Research and development into Trip Optimizer began close to the time MathWorks began rolling out automatic code generation from Simulink
- Trip Optimizer team leveraged this technology to quickly produce proof of concept simulations
- Automatic code generation for embedded targets allowed accelerated transition from simulation environment to on locomotive demos
- Simulink and MATLAB now embedded in the core elements of the product

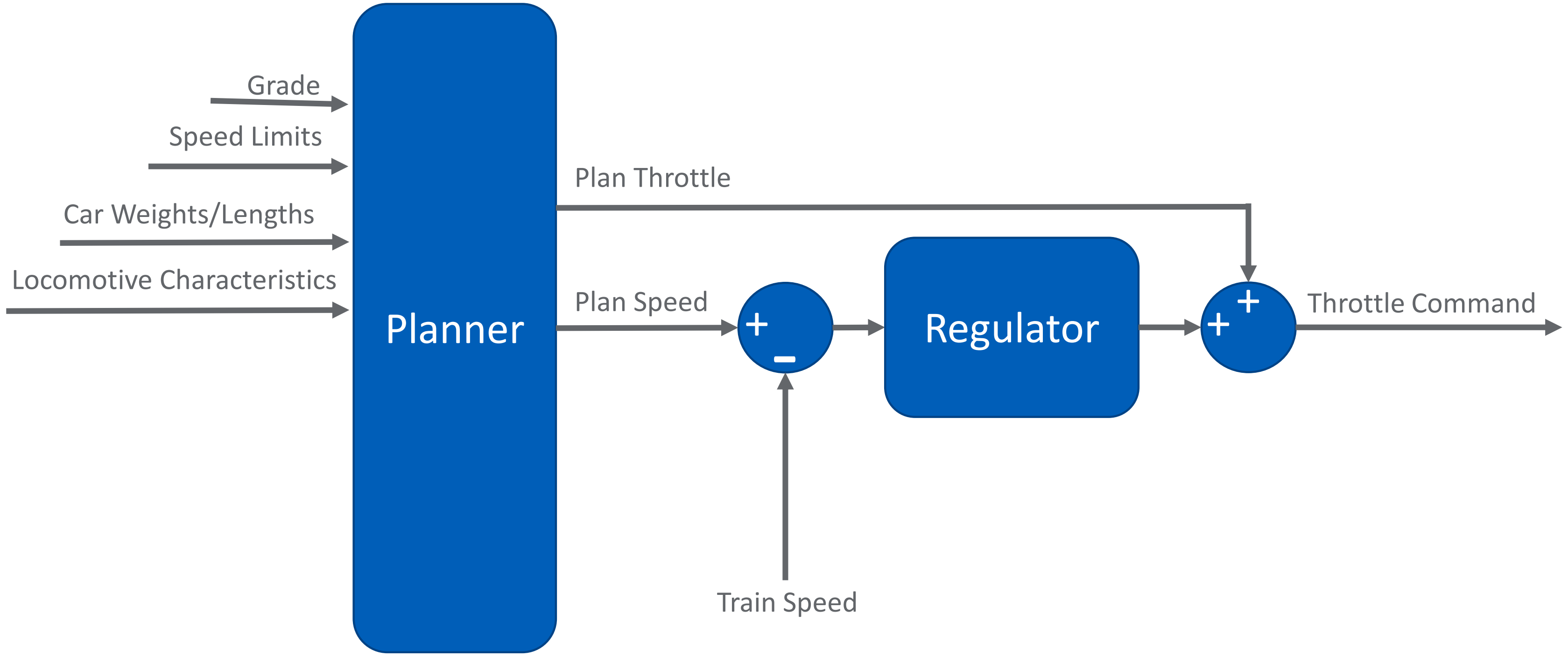
Trip Optimizer Block Diagram



Integrating an External Optimization Library into the MATLAB Code Generation Toolset

```
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0 1.6109693e+001 1.12e+001 5.28e-001   0.0 0.00e+000    -  0.00e+000 0.00e+00
0   0
   1 1.7410406e+001 7.49e-001 2.25e+001  -0.3 7.97e-001    -  3.19e-001 1.00e+00
0f  1
   2 1.8001613e+001 7.52e-003 4.96e+000  -0.3 5.60e-002   2.0 9.97e-001 1.00e+00
0h  1
   3 1.71                                                               1.00e+00
0f  1
   4 1.69                                                               1.00e+00
0h  1
   5 1.7003411e+001 2.16e-002 8.42e-003  -2.9 7.03e-002    -  9.68e-001 1.00e+00
0h  1
   6 1.7013974e+001 2.03e-004 8.65e-005  -4.5 6.22e-003    -  1.00e+000 1.00e+00
0h  1
   7 1.7014017e+001 2.76e-007 2.18e-007 -10.3 1.43e-004    -  9.99e-001 1.00e+00
0h  1
   8 1.7014017e+001 2.13e-014 2.29e-014 -11.0 1.04e-007    -  1.00e+000 1.00e+00
0h  1

Number of Iterations....: 8

                                   (scaled)                 (unscaled)
Objective...............:  1.7014017140224134e+001   1.7014017140224134e+001
Dual infeasibility......:  2.2928101314633036e-014   2.2928101314633036e-014
Constraint violation....:  2.1316282072803006e-014   2.1316282072803006e-014
Complementarity.........:  1.0023967333275279e-011   1.0023967333275279e-011
Overall NLP error.......:  1.0023967333275279e-011   1.0023967333275279e-011
```

Trip Optimizer Planner



- Fuel optimal plan generated for entire route at time of trip initialization
- Trip plan adjusted to account for changes in conditions along the route
- Plan speed is reference for speed regulator
- Plan throttle is 100% feed forward term on speed regulator output

Why Integration?

m-code "Pre-Process" — AutoCode_EntryPoint_1()

m-code "Initial Guess" — AutoCode_EntryPoint_2()

m-code "Build Inputs" — Hand Code

Invoke external executable — Hand Code

m-code "Transform Output" — Hand Code

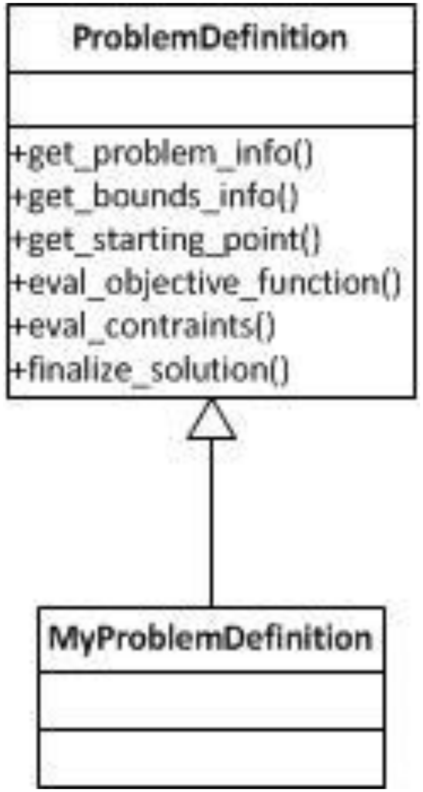
m-code "Post-Process" — AutoCode_EntryPoint_3()

Single interface point with hand-code

m-code "Do It" — AutoCode_EntryPoint()

Desktop environment equivalent to embedded target

The Optimization Library C++ Interface



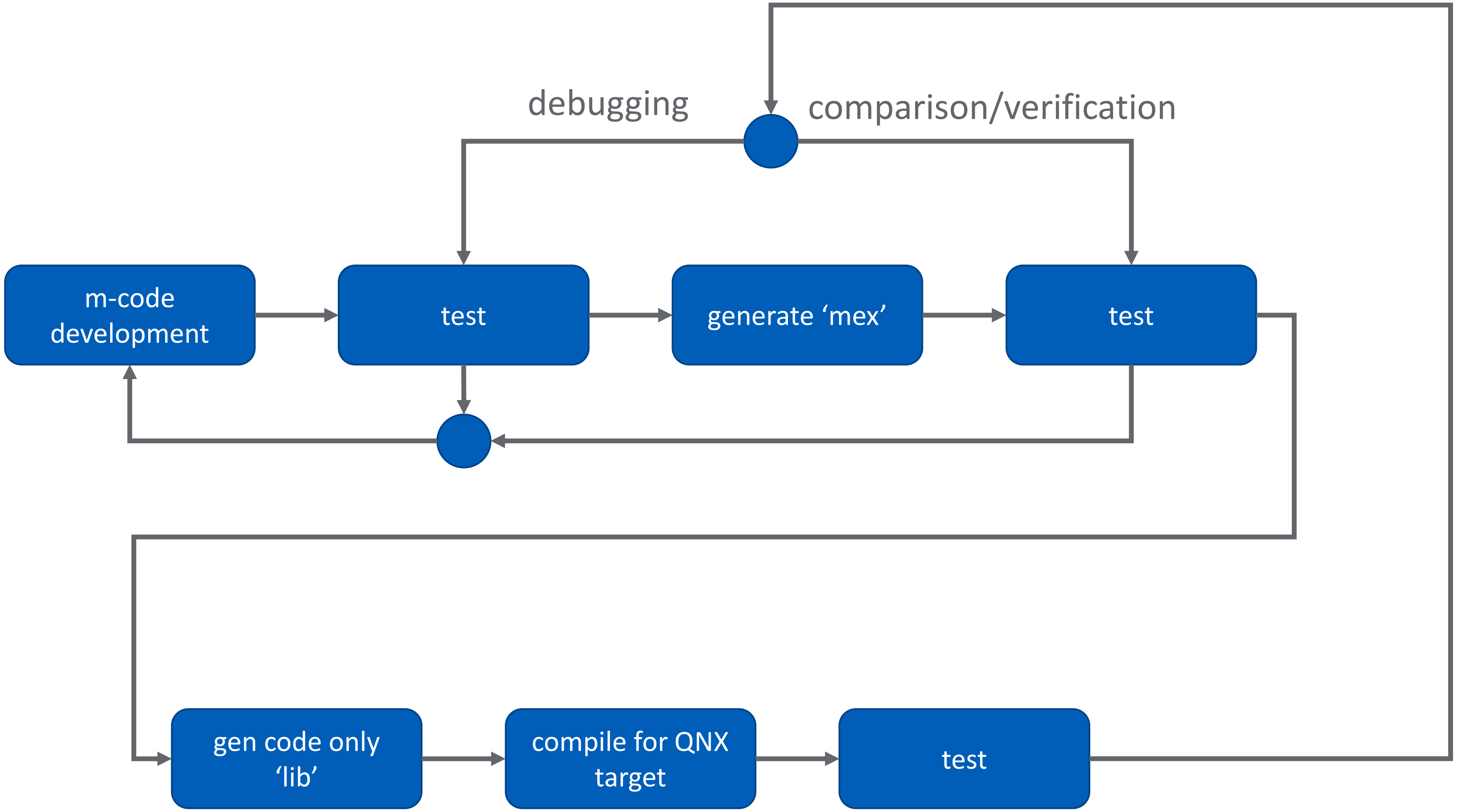

```
SmartPtr<ProblemDefinition> my_problem = new MyProblemDefinition();

SmartPtr<OptimizationApplication> app = ApplicationFactory();

app->Initialize();

app->Optimize(my_problem);
```

How We Did It – ceval + minGW + addLinkObjects

```
function [ProfileOut, errCode, cpuTime, niters] = CallOpt_ceval(OptSpec, Mesh, espec, ProfileOut)

    [OptData, Constants] = PopulateInitData_ceval(OptSpec, Mesh, espec, ProfileOut);

    x = zeros(1, OptData.nvars);
    niters = int32(0);
    errCode = int32(0);
    cpuTime = double(0);

    coder.cinclude('ceval_optimizer.h');
    coder.ceval('ceval_optimizer', coder.rref(OptData), coder.wref(x), coder.wref(errCode), coder.wref(niters), coder.wref(cpuTime));

    ProfileOut  = Post_Process_ceval(Mesh, Constants, x, ProfileOut);
end
```

```
coder.updateBuildInfo('addLinkObjects','liboptimizer.a',ipoptLibPath,'',true,true);
coder.updateBuildInfo('addLinkObjects','libgfortran.a',mingwLibPath,'',true,true);
coder.updateBuildInfo('addLinkObjects','libquadmath.a',mingwLibPath,'',true,true);
coder.updateBuildInfo('addLinkObjects','libstdc++.a',mingwLibPath,'',true,true);
```

Planner - Improved Development Lifecycle



- Each time the model is run on the target all inputs are written to a text file which can be read into MATLAB to recreate the scenario exactly
- Failures from field can be brought back to MATLAB environment for debugging/algorithm enhancement

Conclusions

- Integrating external code into code generation process can enable parity between MATLAB development environment and embedded execution target
- Increased productivity – Development and debugging
- Defects found earlier in life cycle