CGPOPS: A C++ Software for Solving Multiple-Phase Optimal Control Problems Using Adaptive Gaussian Quadrature Collocation and Sparse Nonlinear Programming

Eric Chin

Useful terminal commands:

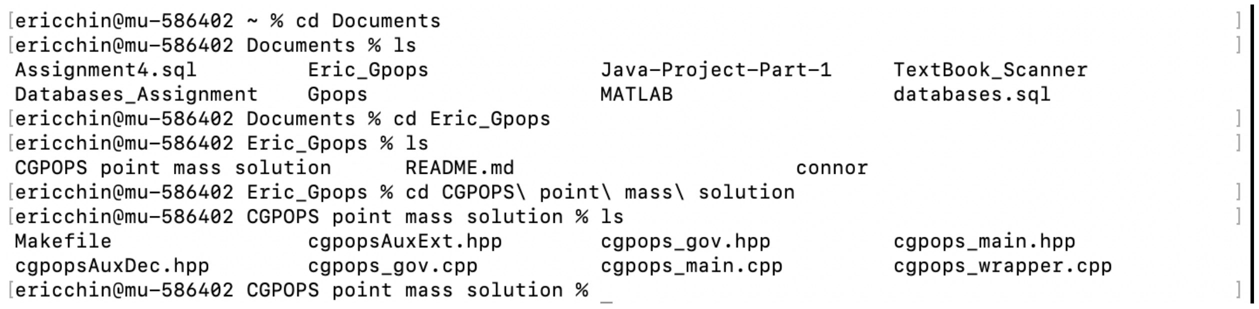
cd - change directory
vim - opens and edits the files if not using a IDE
make - compiles all the files in a directory
./run_cgpops - runs the CGPOPS software
wq - exits a file and saves it
ls - shows all the files in the current directory

High level How to run CGPOPS:

To run any file in CGPOPS you need to first go to the directory of the file that you want to open. The directory must have the run_cgpops file inorder to run and usually that file is found in the file's directory. Use the "make" command in that directory to compile all the files in the directory. If there are no errors, use the "./run_cgpops" command to use the CGPOPS software.

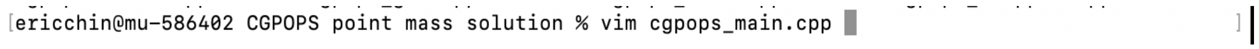
How to use Vim through the terminal

To get the files to run you have to get to the right directory. To this you open your terminal and use "cd" and the directory name to navigate to where your files are stored.
-For this example I would cd into Documents



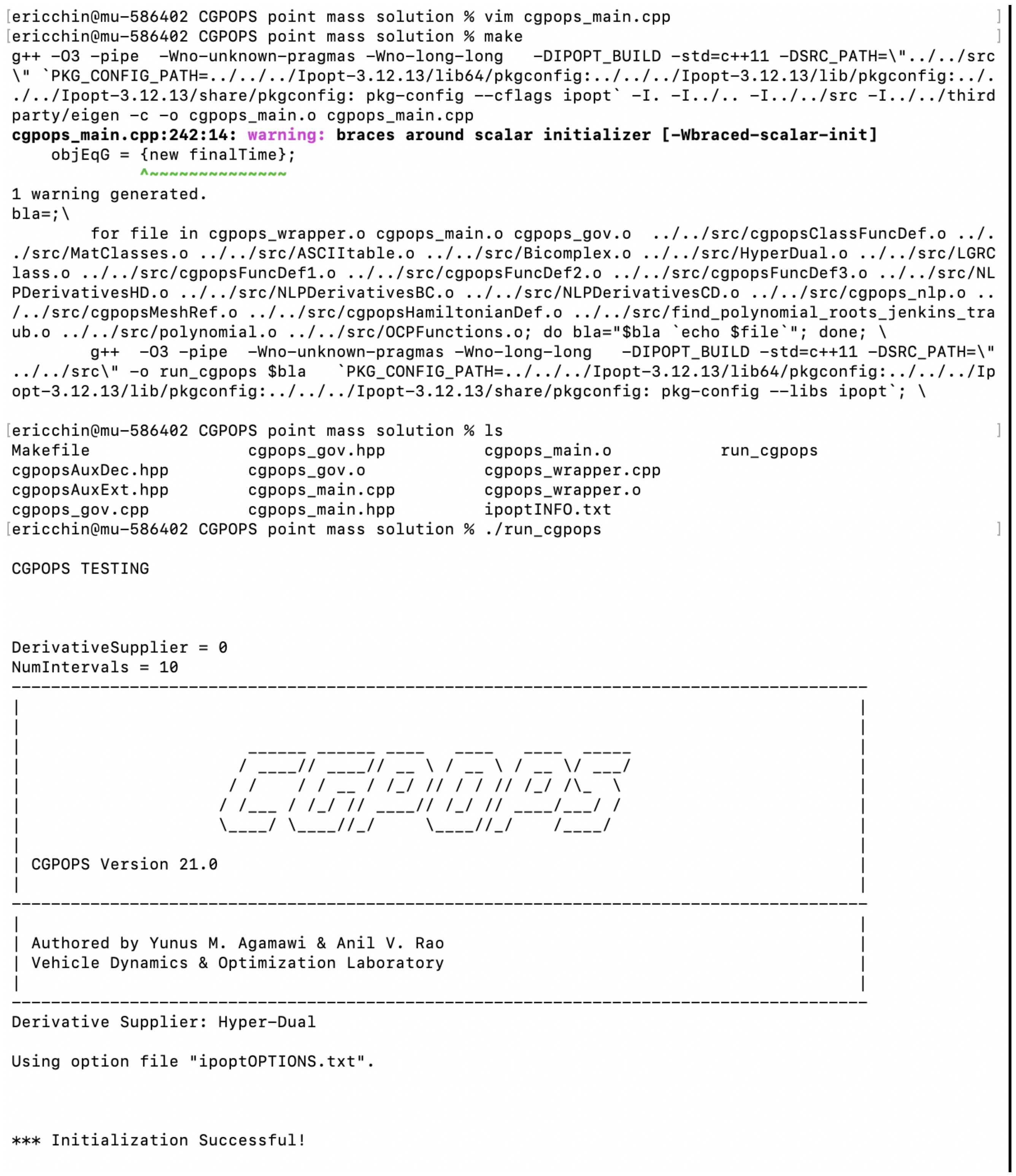
```
ericchin@mu-586402 ~ % cd Documents
ericchin@mu-586402 Documents % ls
 Assignment4.sql         Eric_Gpops              Java-Project-Part-1     TextBook_Scanner
 Databases_Assignment    Gpops                   MATLAB                  databases.sql
ericchin@mu-586402 Documents % cd Eric_Gpops
ericchin@mu-586402 Eric_Gpops % ls
 CGPOPS point mass solution      README.md                       connor
ericchin@mu-586402 Eric_Gpops % cd CGPOPS\ point\ mass\ solution
ericchin@mu-586402 CGPOPS point mass solution % ls
 Makefile                cgpopsAuxExt.hpp        cgpops_gov.hpp          cgpops_main.hpp
 cgpopsAuxDec.hpp        cgpops_gov.cpp          cgpops_main.cpp         cgpops_wrapper.cpp
ericchin@mu-586402 CGPOPS point mass solution % _
```

After you get into the directory that has all the files you can use vim and the file name to open and edit the files.
-For example if you wanted to edit the cgpops_main.cpp file I would use "vim cgpops_main.cpp".



```
ericchin@mu-586402 CGPOPS point mass solution % vim cgpops_main.cpp
```

Inorder to run this programn in the CGPOPS point mass solution directory you would use the command "make" and it should compile all the files in the directory. A new file should be created called "./run_cgpops" you can then type in the terminal "./run_cgpops" and it should run.
Note: every time that you edit your code or make changes you will have to do the "make" command inorder to update the code.



```
[ericchin@mu-586402 CGPOPS point mass solution % vim cgpops_main.cpp
[ericchin@mu-586402 CGPOPS point mass solution % make
g++ -O3 -pipe  -Wno-unknown-pragmas -Wno-long-long   -DIPOPT_BUILD -std=c++11 -DSRC_PATH=\"../../src
\" `PKG_CONFIG_PATH=../../../Ipopt-3.12.13/lib64/pkgconfig:../../../Ipopt-3.12.13/lib/pkgconfig:../.
./../Ipopt-3.12.13/share/pkgconfig: pkg-config --cflags ipopt` -I. -I../.. -I../../src -I../../third
party/eigen -c -o cgpops_main.o cgpops_main.cpp
cgpops_main.cpp:242:14: warning: braces around scalar initializer [-Wbraced-scalar-init]
    objEqG = {new finalTime};
             ^~~~~~~~~~~~~~~
1 warning generated.
bla=;\
        for file in cgpops_wrapper.o cgpops_main.o cgpops_gov.o  ../../src/cgpopsClassFuncDef.o ../.
./src/MatClasses.o ../../src/ASCIItable.o ../../src/Bicomplex.o ../../src/HyperDual.o ../../src/LGRC
lass.o ../../src/cgpopsFuncDef1.o ../../src/cgpopsFuncDef2.o ../../src/cgpopsFuncDef3.o ../../src/NL
PDerivativesHD.o ../../src/NLPDerivativesBC.o ../../src/NLPDerivativesCD.o ../../src/cgpops_nlp.o ..
/../src/cgpopsMeshRef.o ../../src/cgpopsHamiltonianDef.o ../../src/find_polynomial_roots_jenkins_tra
ub.o ../../src/polynomial.o ../../src/OCPFunctions.o; do bla="$bla `echo $file`"; done; \
        g++  -O3 -pipe  -Wno-unknown-pragmas -Wno-long-long   -DIPOPT_BUILD -std=c++11 -DSRC_PATH=\"
../../src\" -o run_cgpops $bla   `PKG_CONFIG_PATH=../../../Ipopt-3.12.13/lib64/pkgconfig:../../../Ip
opt-3.12.13/lib/pkgconfig:../../../Ipopt-3.12.13/share/pkgconfig: pkg-config --libs ipopt`; \

[ericchin@mu-586402 CGPOPS point mass solution % ls
Makefile                cgpops_gov.hpp          cgpops_main.o           run_cgpops
cgpopsAuxDec.hpp        cgpops_gov.o            cgpops_wrapper.cpp
cgpopsAuxExt.hpp        cgpops_main.cpp         cgpops_wrapper.o
cgpops_gov.cpp          cgpops_main.hpp         ipoptINFO.txt
[ericchin@mu-586402 CGPOPS point mass solution % ./run_cgpops

CGPOPS TESTING



DerivativeSupplier = 0
NumIntervals = 10
---------------------------------------------------------------------------------------
|                                                                                     |
|                                                                                     |
|                       ______ ______ ____   ____   ____   _____                      |
|                      / ____// ____// __ \ / __ \ / __ \/ ___/                       |
|                     / /    / / __ / /_/ // / / // /_/ /\_  \                        |
|                    / /___ / /_/ // ____// /_/ // ____/___/ /                        |
|                    \____/ \____//_/     \____//_/    /____/                         |
|                                                                                     |
| CGPOPS Version 21.0                                                                 |
|                                                                                     |
---------------------------------------------------------------------------------------
|                                                                                     |
| Authored by Yunus M. Agamawi & Anil V. Rao                                          |
| Vehicle Dynamics & Optimization Laboratory                                          |
|                                                                                     |
---------------------------------------------------------------------------------------
Derivative Supplier: Hyper-Dual

Using option file "ipoptOPTIONS.txt".



*** Initialization Successful!
```

Note: My run had an error because my personal computer does not have all the dependencie.

<u>Files:</u>

These are the main files that you will need to run the CGPOPS software.

Makefile- This is the file that compiles everything in the directory. Nothing needs to be changed for this file.
cgpopsAuxDec.hpp- This is where you create your Auxiliary data and set it to its values

cgpopsAuxExt.hpp- Externs the Auxiliary data from cgpopsAuxDec.hpp so that all files can use the values

cgpops_gov.hpp- This file is used to define your objects. You can create the prototypes for the different functions and objects that you are solving for here. This file is mainly used to name different objects and give objects inheritance from the CGPOPS library ie. (PathContraintEQ, or ObjectEQ).

```
class xDot : public OrdinaryDifferentialEq
{
public:
    virtual void eval_eq(double& lhs, double* x, double* u, double& t, double* s,
                         double* p);
    virtual void eval_eq(HyperDual& lhs, HyperDual* x, HyperDual* u, HyperDual& t,
                         HyperDual* s, HyperDual* p);
    virtual void eval_eq(Bicomplex& lhs, Bicomplex* x, Bicomplex* u, Bicomplex& t,
                         Bicomplex* s, Bicomplex* p);
private:
    template <class T> void eq_def(T& lhs, T* x, T* u, T& t, T* s, T* p);
};
```

A breakdown of this code: I am creating a class called xDot. It is a public class that extends the OrdinaryDifferentialEq class that is in the CGPOPS library (This basically gives it a OrdinaryDifferentialEq label so that CGPOPS can identify it). In the public section I am creating my default constructor which takes the type class double for lhs, x, u, t, s, and p. In the two lines below the default constructor there are two overloaded constructors, this means that the class of xDot can either take the default constructor parameters as doubles or it can take the overloaded constructor with the type of hyperdual.

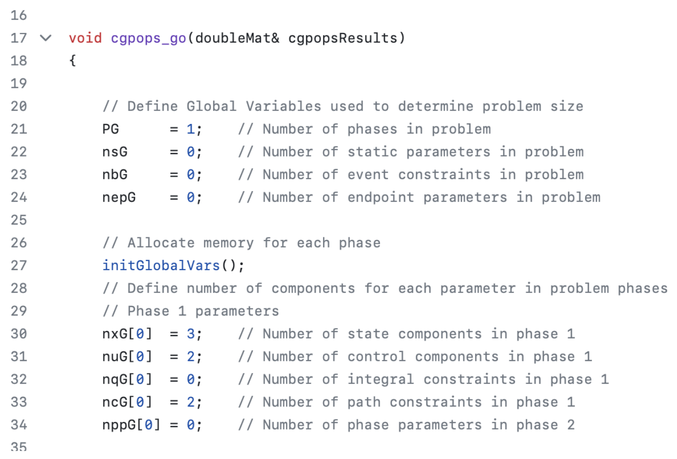
cgpops_gov.cpp- This file is used to add the data into the functions created in the cgpops_gov.hpp file.

```
template <class T> void xDot::eq_def(T& lhs, T* x, T* u, T& t, T* s, T* p)
{
    lhs = cos(theta)*v;
    //lhs = 5;
}
```

In this example I am trying to set the xDot variable to cos(theta)*v. Inorder to save the value you set it equal to lhs and you can use any of the parameters in the parenthesis as variables in the equation. As Well as any global variables that you define ie. (cos, theta, or v) which are all defined in global scope at the top of this file.

```
#include "cgpops_gov.hpp"

#define x1      x[0]
#define y       x[1]
#define theta   x[2]

#define v       u[0]
#define u2      u[1]
#define u_pen   q[0][0]
```

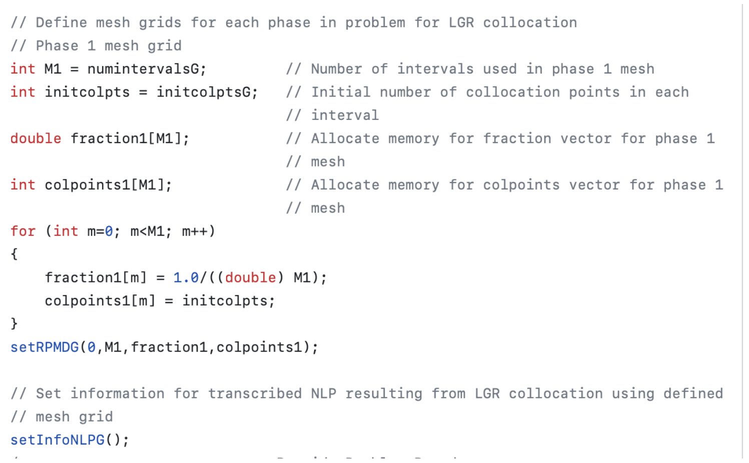
Cgpops_main.cpp- This file is used for multiple things; the first thing is to define the global variables for both the phases and the constraints.



```

void cgpops_go(doubleMat& cgpopsResults)
{

    // Define Global Variables used to determine problem size
    PG      = 1;    // Number of phases in problem
    nsG     = 0;    // Number of static parameters in problem
    nbG     = 0;    // Number of event constraints in problem
    nepG    = 0;    // Number of endpoint parameters in problem

    // Allocate memory for each phase
    initGlobalVars();
    // Define number of components for each parameter in problem phases
    // Phase 1 parameters
    nxG[0]  = 3;    // Number of state components in phase 1
    nuG[0]  = 2;    // Number of control components in phase 1
    nqG[0]  = 0;    // Number of integral constraints in phase 1
    ncG[0]  = 2;    // Number of path constraints in phase 1
    nppG[0] = 0;    // Number of phase parameters in phase 2

```

The global variables set out the optimization problem and vary problem to problem.

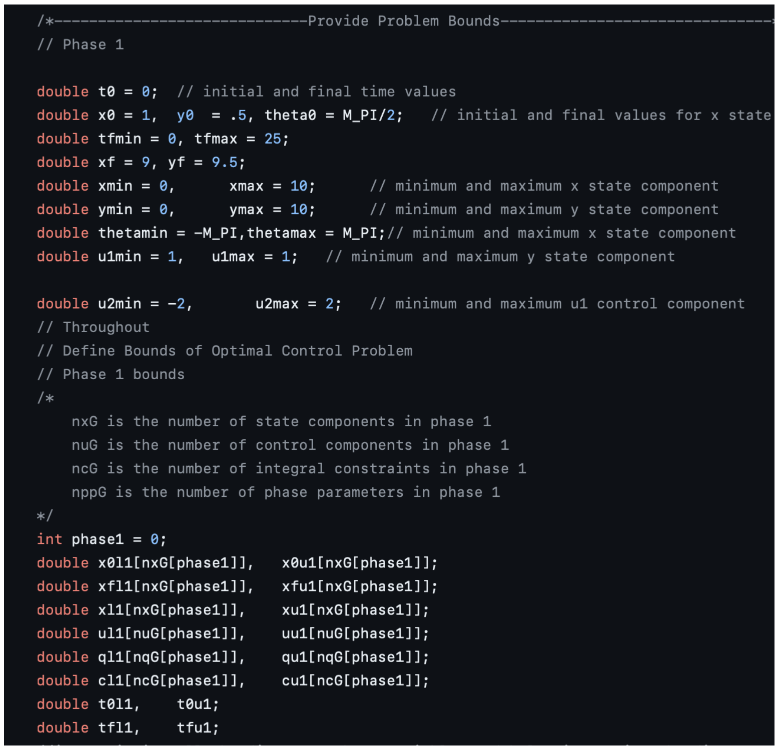
The next part in the cgpops_main file is used when setting up the initial variables for the mesh grids in the phases



```
// Define mesh grids for each phase in problem for LGR collocation
// Phase 1 mesh grid
int M1 = numintervalsG;         // Number of intervals used in phase 1 mesh
int initcolpts = initcolptsG;   // Initial number of collocation points in each
                                // interval
double fraction1[M1];           // Allocate memory for fraction vector for phase 1
                                // mesh
int colpoints1[M1];             // Allocate memory for colpoints vector for phase 1
                                // mesh
for (int m=0; m<M1; m++)
{
    fraction1[m] = 1.0/((double) M1);
    colpoints1[m] = initcolpts;
}
setRPMDG(0,M1,fraction1,colpoints1);

// Set information for transcribed NLP resulting from LGR collocation using defined
// mesh grid
setInfoNLPG();
```

This is used to free up memory in the grid and is used for memory storage later on. This should not be changed too much.

The “provide problem bounds” section is used to set the optimum control variables for the problem. These variables are dependent on what variables are in the optimal control problem. There is a list of usable variable names in the code.

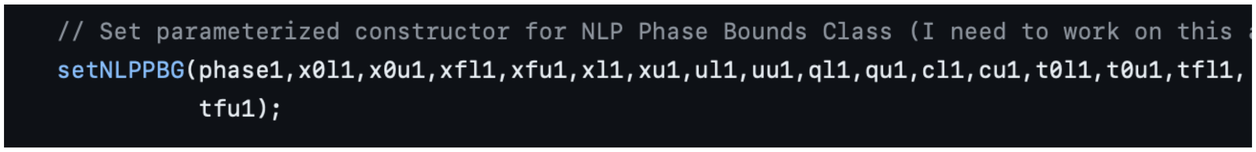


```
/*---------------------------Provide Problem Bounds------------------------------
// Phase 1

double t0 = 0;  // initial and final time values
double x0 = 1,  y0  = .5, theta0 = M_PI/2;   // initial and final values for x state
double tfmin = 0, tfmax = 25;
double xf = 9, yf = 9.5;
double xmin = 0,      xmax = 10;      // minimum and maximum x state component
double ymin = 0,      ymax = 10;      // minimum and maximum y state component
double thetamin = -M_PI,thetamax = M_PI;// minimum and maximum x state component
double u1min = 1,   u1max = 1;   // minimum and maximum y state component

double u2min = -2,       u2max = 2;   // minimum and maximum u1 control component
// Throughout
// Define Bounds of Optimal Control Problem
// Phase 1 bounds
/*
    nxG is the number of state components in phase 1
    nuG is the number of control components in phase 1
    ncG is the number of integral constraints in phase 1
    nppG is the number of phase parameters in phase 1
*/
int phase1 = 0;
double x0l1[nxG[phase1]],   x0u1[nxG[phase1]];
double xfl1[nxG[phase1]],   xfu1[nxG[phase1]];
double xl1[nxG[phase1]],    xu1[nxG[phase1]];
double ul1[nuG[phase1]],    uu1[nuG[phase1]];
double ql1[nqG[phase1]],    qu1[nqG[phase1]];
double cl1[ncG[phase1]],    cu1[ncG[phase1]];
double t0l1,    t0u1;
double tfl1,    tfu1;
```

The first part is setting the initial variables that are going to be used in the optimization software. In the second part after the block code it is instantiating the phases and assigning the variables to their phases and components.

lhs0 - value for endpoint function to be computed
x0 - pointer to array of initial state values in each phase
xf - pointer to array of final state values in each phase
q - pointer to array of integral values in each phase
t0 - pointer to array of initial time value in each phase
tf - pointer to array of final time value in each phase
s - array of static parameter values in problem
e - array of endpoint parameter values in problem
lhs - value for phase function to be computed
x - array of state values at point in phase
u - array of control values at point in phase
t - value of time at point in phase
s - array of static parameter values in problem
p - array of phase point parameter values at point in phase

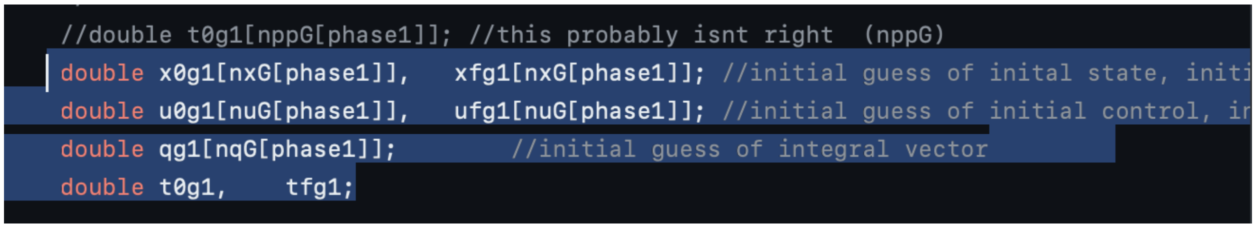
For creating the bounds and setting up the NLP bound class you can use any of the variables above. They must be the same because CGPOPS creates these variables within the software.



```
    // Set parameterized constructor for NLP Phase Bounds Class (I need to work on this 
    setNLPPBG(phase1,x0l1,x0u1,xfl1,xfu1,xl1,xu1,ul1,uu1,ql1,qu1,cl1,cu1,t0l1,t0u1,tfl1,
              tfu1);
```

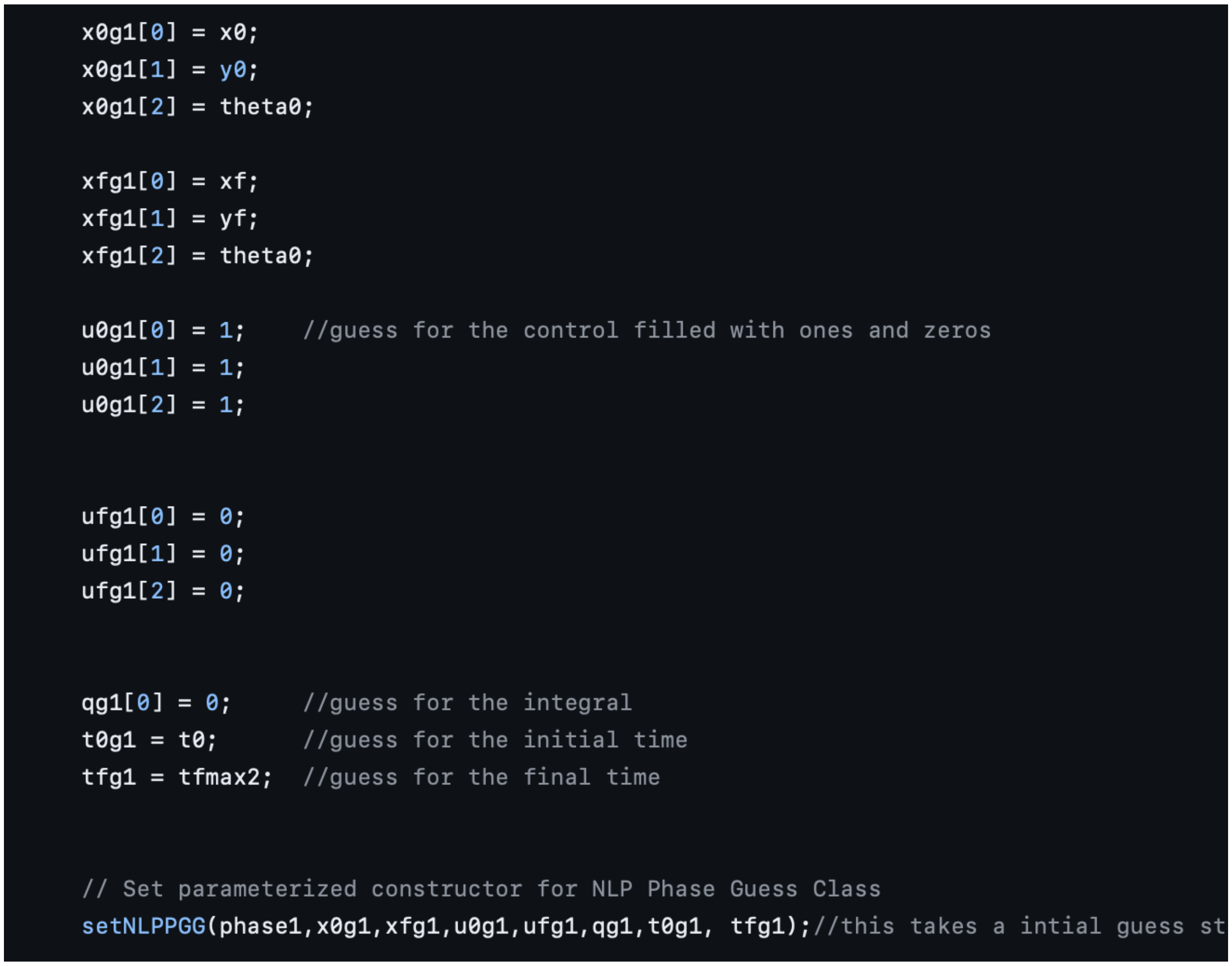
This sets the NLP and should match the variables that you created above.

This is for the guess class and sets the variables that you are going to use inside of what phase it is going to be in.

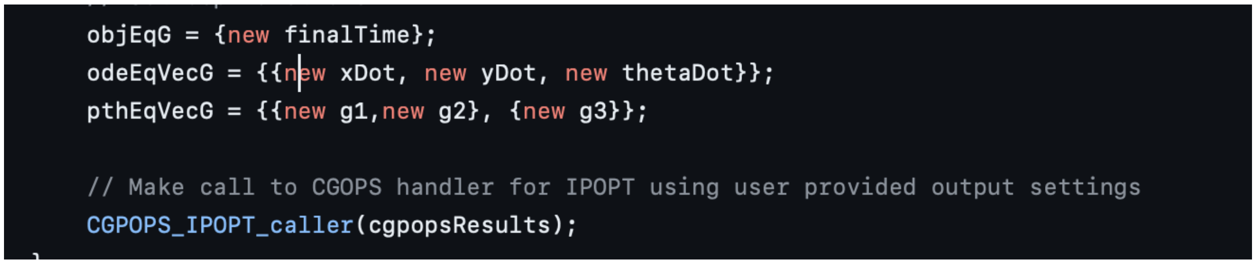


```
    //double t0g1[nppG[phase1]]; //this probably isnt right  (nppG)
    double x0g1[nxG[phase1]],   xfg1[nxG[phase1]]; //initial guess of inital state, initi
    double u0g1[nuG[phase1]],   ufg1[nuG[phase1]]; //initial guess of initial control, in
    double qg1[nqG[phase1]];         //initial guess of integral vector
    double t0g1,    tfg1;
```

This is the guess variable creation and putting the variables into the guess object. The guess variables must match what is inside the constructor. You can change the constructor to what the problem requires.



```
    x0g1[0] = x0;
    x0g1[1] = y0;
    x0g1[2] = theta0;

    xfg1[0] = xf;
    xfg1[1] = yf;
    xfg1[2] = theta0;

    u0g1[0] = 1;    //guess for the control filled with ones and zeros
    u0g1[1] = 1;
    u0g1[2] = 1;


    ufg1[0] = 0;
    ufg1[1] = 0;
    ufg1[2] = 0;


    qg1[0] = 0;     //guess for the integral
    t0g1 = t0;      //guess for the initial time
    tfg1 = tfmax2;  //guess for the final time


    // Set parameterized constructor for NLP Phase Guess Class
    setNLPPGG(phase1,x0g1,xfg1,u0g1,ufg1,qg1,t0g1, tfg1);//this takes a intial guess st
```

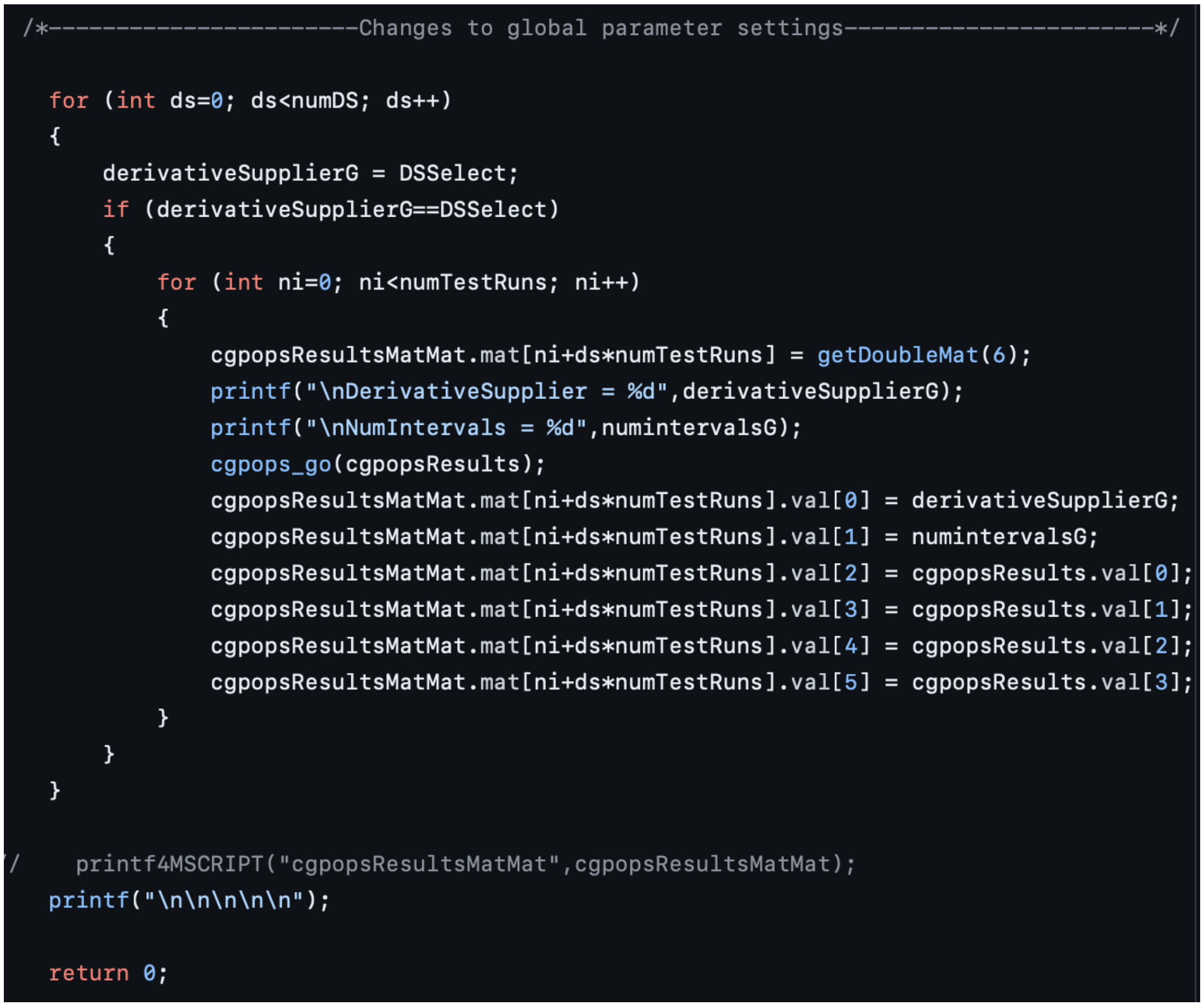


```
    objEqG = {new finalTime};
    odeEqVecG = {{new xDot, new yDot, new thetaDot}};
    pthEqVecG = {{new g1,new g2}, {new g3}};

    // Make call to CGOPS handler for IPOPT using user provided output settings
    CGPOPS_IPOPT_caller(cgpopsResults);
```

This should be the last part of the code. It shows what you are calling and what variables you are solving for. For example the object eqg is taking a final time, and the odeEQVECG is talking in the xDot, yDot, and thetaDot.
The last part of the code is the CGPOPS IPOPT caller that uses everything that was stored into the cgpopresults and uses the linear solver inside of ipopt. This usually stays the same.

Cgpops_wrapper.cpp- The wrapper is the file that creates the mesh settings. You can think of it was a translator from matlab to C++ for the problem mesh. You can change the variable numbers in this file but the functions should usually be the same

This last part of the wapper.cpp is what actually runs when the cgpops_wrapper.cpp is called. I used this code to check what was happening and to see what steps were not running using print statements..



```
/*----------------------Changes to global parameter settings----------------------*/

   for (int ds=0; ds<numDS; ds++)
   {
       derivativeSupplierG = DSSelect;
       if (derivativeSupplierG==DSSelect)
       {
           for (int ni=0; ni<numTestRuns; ni++)
           {
               cgpopsResultsMatMat.mat[ni+ds*numTestRuns] = getDoubleMat(6);
               printf("\nDerivativeSupplier = %d",derivativeSupplierG);
               printf("\nNumIntervals = %d",numintervalsG);
               cgpops_go(cgpopsResults);
               cgpopsResultsMatMat.mat[ni+ds*numTestRuns].val[0] = derivativeSupplierG;
               cgpopsResultsMatMat.mat[ni+ds*numTestRuns].val[1] = numintervalsG;
               cgpopsResultsMatMat.mat[ni+ds*numTestRuns].val[2] = cgpopsResults.val[0];
               cgpopsResultsMatMat.mat[ni+ds*numTestRuns].val[3] = cgpopsResults.val[1];
               cgpopsResultsMatMat.mat[ni+ds*numTestRuns].val[4] = cgpopsResults.val[2];
               cgpopsResultsMatMat.mat[ni+ds*numTestRuns].val[5] = cgpopsResults.val[3];
           }
       }
   }

/    printf4MSCRIPT("cgpopsResultsMatMat",cgpopsResultsMatMat);
   printf("\n\n\n\n\n");

   return 0;
```