User's Manual for Lpopc A C++ Package for Solving Multiple-Phase Optimal Control Problems Using Adaptive Radau Pseudospectral Methods version 1.0.0 published with EPL

Xue Zhichen
Wang Yujie
Wang Na
Nanjing University of Science and Technology

August 9, 2015

Disclaimer

Contents

1 Introldduction to Lpopc

1.1 What is Lpopc

Lpopc is an open source optimal control package written in C++ that using Radau Pseudospectral Methods with adaptive mesh refinement algorithm. **The algorithm employed by Lpopc is mainly based on the following literatures[2][6][7].**
Lpopc is able to deal with the the problems of the follwing form[7].Given a set of P phases (where $p = 1, \dots, P$), minimize the cost functional

$$J = \sum_{p=1}^{P} J^{(p)} = \sum_{p=1}^{P} \left[\Phi^{(p)}(\mathbf{x}^{(p)}(t_0), t_0, \mathbf{x}^{(p)}(t_f), t_f; \mathbf{q}^{(p)}) + \mathcal{L}^{(p)}(\mathbf{x}^{(p)}(t), \mathbf{u}^{(p)}(t), t; \mathbf{q}^{(p)}) dt \right] \tag{1}$$

subject to the dynamic constraint

$$\dot{\mathbf{x}}^{(p)} = \mathbf{f}^{(p)}(\mathbf{x}^{(p)}, \mathbf{u}^{(p)}, t; \mathbf{q}^{(p)}), \qquad (p = 1, \dots, P), \tag{2}$$

the boundary conditions

$$\phi_{\min} \le \phi^{(p)}(\mathbf{x}^{(p)}(t_0), t_0^{(p)}, \mathbf{x}^{(p)}(t_f), t_f^{(p)}; \mathbf{q}^{(p)}) \le \phi_{\max}, \qquad (p = 1, \dots, P), \tag{3}$$

the inequality path constraints

$$\mathbf{C}_{\min}^{(p)} \le \mathbf{C}^{(p)}(\mathbf{x}^{(p)}(t), \mathbf{u}^{(p)}(t), t; \mathbf{q}^{(p)}) \le \mathbf{C}_{\max}^{(p)}, \qquad (p = 1, \dots, P), \tag{4}$$

and the phase continuity (linkage) constraints

$$\mathbf{P}^{(s)}(\mathbf{x}^{(p_l^s)}(t_f), t_f^{(p_l^s)}; \mathbf{q}^{(p_l^s)}, \mathbf{x}^{(p_u^s)}(t_0), t_0^{(p_u^s)}; \mathbf{q}^{(p_u^s)}) = \mathbf{0}, \qquad (p_l, p_u \in [1, \dots, P], s = 1, \dots, L) \tag{5}$$

where $\mathbf{x}^{(p)}(t) \in \mathbb{R}^{n_p}$, $\mathbf{u}^{(p)}(t) \in \mathbb{R}^{m_p}$, $\mathbf{q}^{(p)} \in \mathbb{R}^{q_p}$, and $t \in \mathbb{R}$ are, respectively, the state, control, static parameters, and time in phase $p \in [1, \dots, P]$, L is the number of phases to be linked, $p_l^s \in [1, \dots, P]$, $(s = 1, \dots, L)$ are the "left" phase numbers, and $p_u^s \in [1, \dots, P]$, $(s = 1, \dots, L)$ are the "right" phase numbers.

1.2 About the Author

Xue Zhichen,Wang Yujie and Wang Na are undergraduate student studying at Nanjing University of Science and Technology.
We develop this program since August 2014,and we get a lot of help from professor Sun RuiSheng.

1.3 How you can help

The author's email address is
eddy_lpopc@163.com.

- Sending bug reports.
- Sending corrections to the documentation.
- Discussing with author about RPM algorithm.
- Tell your experience with Lpopc.
- Quoting the website of Lpopc
`http://sourceforge.net/projects/lpopc`

1.4 External Software Libraries

1.4.1 Armadillo

Armadillo C++ matrix library
Fast C++ matrix library with easy to use functions and syntax, deliberately similar to Matlab. Uses template meta-programming techniques. Also provides efficient wrappers for LAPACK, BLAS and ATLAS libraries, including high-performance versions such as Intel MKL, AMD ACML and OpenBLAS[9]
For more details, see `http://arma.sourceforge.net`
LPOPC implements matrix opertion using Armadillo for high-speed calculation.

1.4.2 BLAS and LAPACK

Note, the efficiency of Armadillo rely on the efficiency of Blas!
The standard and widely used linear algebra package which can be downloaded in source and binary form from http://www.netlib.org
It is strongly recommended to use some optinized BLAS implemetion. Examples for efficient BLAS implementations are ACML,ESSL,MKL, Atlas and OpenBLAS.The last two are open-source.
The Openblas can be found here,`http://www.openblas.net/`

1.4.3 IPOPT

IPOPT is an open-source C++ software package for large-scale nonlinear optimization,which uses an interior point method[10].
For more details,see `https://projects.coin-or.org/Ipopt/`
Lpopc convert a optimal control problem to nonlinear optimaztion problem,which is solved by Ipopt.
The major part of executing time spends in Ipopt. Modify the configuration of Ipopt to accelerate the calculation. See the manual of Ipopt.

1.4.4 spdlog

Very fast, header only, C++ logging library.Lpopc uses it print message.
See `https://github.com/gabime/spdlog`

2 Installing Lpopc

Supported Platforms

Lpopc has been successfully compiled under the follwing operating system.

- Ubuntu Linux 15.04 64Bit,using GCC 4.9.2 and Gfortran 4.9.2.
- Windows 7 64bit using Visual Studio Community 2013 and intel compiler.
- Windows 7 64bit using MinGW64 with GCC 4.9.2 and Gfortran 4.9.2.

Getting External Code

- Armadillo `http://sourceforge.net/projects/arma/?source=directory`
  Get **armadillo-5.300.4.tar.gz**
  The interface of arma is stable,so you can try newer vision.
- Ipopt `http://www.coin-or.org/download/source/Ipopt/`
  Get **Ipopt-3.12.3.tgz**
- OpenBLAS(Linux optional) `http://www.openblas.net/`
  Get **OpenBLAS 0.2.14**

2.1 Linux

IPOPT can use different Sparse Linear Solver,here,we use **MUMPS+METIS**.

1. Getting System Packages
   gcc,g++(need C++11),gfortran,wget and cmake

2. Install OpenBLAS
   Extract the downloaded source package ,enter in the folder **OpenBLAS-0.2.14** and type

   ```
   $ make
   $ make install
   ```

   Copy the folder **lib** to $(LPOPC Dir)/ThirdParty/openblas/

3. Install Armadillo
   Copy $(armadir)/ include folder to $(lpopcdir)/ ThirdParty/arma/

4. Install IPOPT
   Extract the downloaded source package.
   Get **METIS** and **MUMPS** by runing the script $(IPOPT Dir)/ThirdParty/Mumps/get.Mumps and $(IPOPT Dir)/ThirdParty/Metis/get.Metis (need **wget**).See Ipopt Manual.
   Enter in the folder **Ipopt-3.12.3** and type

   ```
   $ ./configure --with-blas="-L$(OpenBlasInatallDir)␣-lopenblas"
    --with-lapack="-L$(OpenBlasInatallDir)␣-lopenblas"
   ```

   In my computer it is

   ```
   $ ./configure --with-blas="-L/home/eddy/OpenBLAS-0.2.14␣-lopenblas"
    --with-lapack="-L/home/eddy/OpenBLAS-0.2.14␣-lopenblas"
   ```

   Then

   ```
   $ make -j(number of cores of youre CPU)
   $ make install
   ```

   Copy **include lib share** folders to $(LPOPC Dir)/ThirdParty/ipopt/

5. Inatall LPOPC

   ```
   $ cd $(LpopcDir)/build
   $ cmake ..
   $ make
   ```

2.2 Windows

2.2.1 Cygwin(MinGW,MYSY2)

Install on these Unix like environments is same with Linux,but compiling is much slower than native compiler(cl and ifort).

2.2.2 Visual Studio with ifort

The binarie from website of IPOPT doesn't work well,so you have to compile it from source.You need an Inter fortran compiler **ifort** and MKL.

1. Extract the source pacake of IPOPT to $IPOPTDIR.

2. Get **METIS** and **MUMPS** use wget or download from their official website.See IPOPT manual.

3. open Ipopt-3.12.3\Ipopt MSVisualStudio\v8-ifort\IpOpt-ifort.sln
   remove the project IpOptFss,libhsl and libhsl-no-MA57

4. modeify property of Ipopt-vc8

   - Linker->Input->Additional Dependencies
     replace with
     mkl_intel_c.lib mkl_intel_thread.lib mkl_core.lib libiomp5md.lib IpOptFor.lib CoinMumpsC.lib CoinMetis.lib CoinMumpsF90.lib
   - VC++ Directories->Library Directories
     replace with
     $(INTEL_COMPILER_PATH)\lib\ia32;
     $(INTEL_COMPILER_PATH)\mkl\ia32\lib;
     $(SolutionDir)$(Configuration)
     In my computer it's
     C:\Program Files (x86)\Intel\Composer XE\compiler\lib\ia32;
     $(SolutionDir)$(Configuration)\;
     C:\Program Files (x86)\Intel\Composer XE\mkl\lib\ia32;

5. Then compile the Ipopt.

6. Copy "include" and "lib" Folders into $lpopcdir\ThirdParty\ipopt
   make sure you have lib\Win32\Release\Ipopt-vc8.lib and IpOptFor.lib

7. Copy Ipopt-vc8.dll to the folder $(SolutionDir)$(Configuration)\

Install Armadillo

1. Copy $(armadir)\include folder to $(lpopcdir)\ThirdParty\arma

2. Modify the headfile config.hpp in armadillo_bits.
   Uncomment #define ARMA_BLAS_CAPITALS
   Comment #define ARMA_BLAS_UNDERSCORE

Compile LPOPC
Open solutionfile lpopc_msvc.sln in$(lpopcdir)\Lpopc\MSVS\liblpopc
Just build it.
If you want to try different examples or compile your problem,just replace the file HyperSensitive.cpp and hypersensitive.h,with files of your problems.
Note
If you are going to creat a new project,you need link these librarys,
mkl_intel_c.lib mkl_intel_thread.lib mkl_core.lib libiomp5md.lib Ipopt-vc8.lib liblpopc.lib

3 Interfacing Your Problem

3.1 Quick Reference

1. Creat a new headfile, declare a class inherited from **Lpopc::FunctionWrapper** implementing your optimal control problem.

```
class MyFunction:public FunctionWrapper
{
}
```

2. Declare a object of type **LpopcApplication** at the beginning of your code.

```
shared_ptr<LpopcApplication> app(new LpopcApplication(console_print));
```

3. Declare **Lpopc::Phase** and **Lpopc::Link**, input limits of your optimal control problem.
4. Declare a **Lpopc::OptimalProblem** object,add your **phases**,**linkages** and **function**.

```
        shared_ptr<FunctionWrapper> userfun(new MyFunction());

        shared_ptr<OptimalProblem> optpro(new OptimalProblem(nphase,nlink,userfun));

        optpro->AddPhase(Phase1);

        optpro->AddLinkage(Link1);
```

5. Add the **OptimalControlProblem** to **LpopcApplication**,set options,and solve the problem.

```
        app->SetOptimalControlProblem(optpro);

        app->Options()->SetStringValue(OPTION_NAME, OPTION_VALUE);

        app->SolveOptimalProblem();
```

3.2 Bounds of Var and Functions

Sepicify the limits phase by phase.
Including the headfile "LpOptimalProblem.hpp" and declare a object of `Phase` each phase using `shared_ptr<>` (C++11 needed)
The construct function of Phase accept6 arguments,Phase(index of phase,nstates,ncontrols,nparameters,npaths,nevents)
Index of phase start from 1,not 0. Then,we need to set the limits of this phase.Consder we has Phase1 object
`shared_ptr<Phase> Phase1 (new Phase(1,7,3,0,1,0));`

- Set time

```
        Phase1->SetTimeMin(t0min,tfmin);
        Phase1->SetTimeMax(t0max,tfmax);
```

- Set state one by one

```
        Phase1->SetStateMin(x0min,xmin,xfmin);
        Phase1->SetStateMax(x0max,xmax,xfmax);
```

  x0min is the minimum of initial state,xfmin is the minimum of terminal state,and xmin is the minimum of state except the initial and terminal state. The x0max,xmax,xfmax are the maximum.If the problem has more than one state,just do the same thing for every state.

- Set control one by one

```
        Phase1->SetcontrolMin(umin);
        Phase1->SetcontrolMax(umax);
```

  If the problem has more than one control,just do the same thing for every control.

- Set parameter one by one

```
        Phase1->SetparameterMin(umin);
        Phase1->SetparameterMax(umax);
```

  If the problem has more than one parameter,just do the same thing for every parameter.

- Set path bounds

```
        Phase1->SetpathMin(pathmin);
        Phase1->SetpathMax(pathmax);
```

  If the problem has more than one path,just do the same thing for every path.

- Set event bounds

```
        Phase1->SeteventMin(eventmin);
        Phase1->SeteventMax(eventmax);
```

  If the problem has more than one event,just do the same thing for every event.

- *Set duration bounds

```
        Phase1->SetDuration(durationtmin,duration);
```

  Lower and upper limits on the duration of phase $p \in [1, \dots, P]$.
  The default value is $(0, Inf)$,e.g.$t_f - t_0 \geq 0$.

Example of Setting Up a Limits of Var and Fun

Consider the following two-phase optimal control problem. In particular, suppose that *phase 1* of the problem has 3 states, 2 controls, 2 path constraints, and 5 event constraints. In addition, suppose that the lower and upper limits on the initial and terminal time in the first phase are given as

$$\begin{array}{rcccr} 0 & \leq & t_0^{(1)} & \leq & 0 \\ 50 & \leq & t_f^{(1)} & \leq & 100 \end{array}$$

Next, suppose that the lower and upper limits on the states at the *start* of the first phase are given, respectively, as

$$\begin{array}{rcccr} 1 & \leq & x_1(t_0^{(1)}) & \leq & 1 \\ -3 & \leq & x_2(t_0^{(1)}) & \leq & 0 \\ 0 & \leq & x_2(t_0^{(1)}) & \leq & 5 \end{array}$$

Similarly, suppose that the lower and upper limits on the states *during* the first phase are given, respectively, as

$$\begin{array}{rcccr} 1 & \leq & x_1(t^{(1)}) & \leq & 10 \\ -50 & \leq & x_2(t^{(1)}) & \leq & 50 \\ -20 & \leq & x_2(t^{(1)}) & \leq & 20 \end{array}$$

Finally, suppose that the lower and upper limits on the states at the *terminus* of the first phase are given, respectively, as

$$\begin{array}{rcccr} 5 & \leq & x_1(t_f^{(1)}) & \leq & 7 \\ 2 & \leq & x_2(t_f^{(1)}) & \leq & 2.5 \\ -\pi & \leq & x_2(t_f^{(1)}) & \leq & \pi \end{array}$$

Next, suppose that the lower and upper limits on the controls *during* the first phase are given, respectively, as

$$\begin{array}{rcccr} -50 & \leq & u_1(t^{(1)}) & \leq & 50 \\ -100 & \leq & u_2(t^{(1)}) & \leq & 100 \end{array}$$

Next, suppose that the lower and upper limits on the path constraints *during* the first phase are given, respectively, as

$$\begin{array}{rcccr} -10 & \leq & p_1(t^{(1)}) & \leq & 10 \\ 1 & \leq & p_2(t^{(1)}) & \leq & 1 \end{array}$$

Next, suppose that the lower and upper limits on the event constraints of the first phase are given, respectively, as

$$\begin{array}{rcccr} 0 & \leq & \phi_1^{(1)} & \leq & 1 \\ -2 & \leq & \phi_2^{(1)} & \leq & 4 \\ 8 & \leq & \phi_3^{(1)} & \leq & 20 \\ 3 & \leq & \phi_4^{(1)} & \leq & 3 \\ 10 & \leq & \phi_5^{(1)} & \leq & 10 \end{array}$$

```
       shared_ptr<Phase> phase1(new Phase(1, 3, 2, 0, 2, 5));
       phase1->SetTimeMin(0.0, 50.0);
       phase1->SetTimeMax(0., 100.);

       phase1->SetStateMin(1., 1., 5.);
       phase1->SetStateMax(1., 10., 7.);

       phase1->SetStateMin(-3., -50., 2.);
       phase1->SetStateMax(0., 50., 2.5);

       phase1->SetStateMin(0., -20., -datum::pi);//arma::datum::pi
       phase1->SetStateMax(5, 20, datum::pi);

       phase1->SetcontrolMin(-50.0);
       phase1->SetcontrolMax(50.0);

       phase1->SetcontrolMin(-100.);
       phase1->SetcontrolMax(100);

       Phase1->SetpathMin(-10.0);
       Phase1->SetpathMax(10.0);

       Phase1->SetpathMin(1.0);
       Phase1->SetpathMax(1.0);

       Phase1->SeteventMin(0.0);
       Phase1->SeteventMax(1,0);

       Phase1->SeteventMin(-2.0);
       Phase1->SeteventMax(4.0);

       Phase1->SeteventMin(8.0);
       Phase1->SeteventMax(20.0);

       Phase1->SeteventMin(3.0);
       Phase1->SeteventMax(3,0);

       Phase1->SeteventMin(10.0);
       Phase1->SeteventMax(10.0);
```

3.2.1 Linkage

Declare a object of `Link` each link using `shared_ptr<>` (C++11 needed)

```
       shared_ptr<Linkage> Link1 (new Linkage(link_insex,left_phase_index,right_phase_index));
```

Here,all three index is start from 1,consider we want to link phase 1 and phase 2 with one link function.

```
       shared_ptr<Linkage> Link1 (new Linkage(1,1,2));
       Link1->SetLinkMin(linkmin);
       Link2->SetLinkMax(linkmax);
```

The time of each phase is auto-linked by LPOPC,there is no need to set it.The $t(1)_f$ of phase 1 and $t(2)_0$ of phase 2 autolinked using $t(1)_f - t(2)_0 = 0$ inside LPOPC.

3.3 Optimal Problem Functions

The function interface in LPOPC is modified from GPOPS.In order to interface optimal control function,declare a class inherited from Lpopc::FunctionWrapper.Make a new headfile and include the headfile named "LpFunctionWrapper.h".Here is an example.

```
//File name userfun.h
#ifndef USERFUN_H
#define USERFUN_H
#include "LpFunctionWrapper.h"
using namespace Lpopc;
class UserFunction :public FunctionWrapper
{
public:
UserFunction(){}
virtual void MayerCost(SolCost&mySolcost, double& mayer);
virtual void DerivMayer(SolCost&mySolcost, rowvec& deriv_mayer);

virtual void LagrangeCost(SolCost& mySolcost, vec& langrange);
virtual void DerivLagrange(SolCost& mySolcost, mat& deriv_langrange);

virtual void DaeFunction(SolDae& mySolDae, mat& stateout, mat& pathout);
virtual void DerivDae(SolDae& mySolDae, mat& deriv_state, mat& deriv_path);

virtual void EventFunction(SolEvent& mySolEvent, vec& eventout);
virtual void DerivEvent(SolEvent& mySolEvent, mat& deriv_event);

virtual void LinkFunction(SolLink& mySolLink, vec& linkageout);
virtual void DerivLink(SolLink& mySolLink, mat& derive_link);
virtual ~UserFunction(){}
};
#endif // !USERFUN_H
```

- Cost Function
  `void MayerCost(SolCost&mySolcost, double& mayer)`
  User-defined Mayer cost functional
  `void LagrangeCost(SolCost& mySolcost, vec& langrange)`
  User-defined Lagrange cost functional
  SolCost, input argument, Here are the members of SolCost

  - `int SolCost.phase_num_`
    the index of phase,start from 1,not 0!
  - `double SolCost.initial_time_`
    the initial timme
  - `vec SolCost.initial_state_`
    a column vector of length nstate
  - `double SolCost.terminal_time_`
    the terminal time
  - `vec SolCost.terminal_state_`
    a column vector of length nstate
  - `vec SolCost.time_`
    a column vector of length N that contains the time (excluding terminal time)
  - `mat SolCost.state_`
    a matrix of size N*nstate ,contains the values of states(excluding terminal state)

- `mat SolCost.control_`
  a matrix of size N*ncontrol,cointains the values of controls(excluding terminal state)
- `mat SolCost.parameter_`
  a column vector of length q,contains the values of the static parameter.

Note N=number of LGR points which are on the interior of the time interval.
mayer, output argument
A scalar.If Mayer cost is zero,you need to set `mayer=0.0;`
lagrange, output argument
An arma::vec,column vector of size $N*1$.If Lagrange cost is zero,you need to set `langrange=zeros(t.n_elem,1);`
Example of a Cost Functional

Suppose we have a two-phase optimal control problem Suppose further that the dimension of thestate in each phase is 2 while the dimension of the control in each phase is 2. Also, suppose that the endpoint and integrand cost in phase 1 are given, respectively, as

$$\begin{aligned}\Phi^{(1)}(\mathbf{x}^{(1)}(t_0), t_0^{(1)}, \mathbf{x}^{(1)}(t_f), t_f^{(1)}) &= \mathbf{x}^T(t_f)\mathbf{S}\mathbf{x}(t_f)\\ \mathcal{L}^{(1)}(\mathbf{x}^{(1)}(t), \mathbf{u}^{(1)}(t), t) &= \mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u}\end{aligned}$$

while the endpoint and integrand in phase 2 are given, respectively, as

$$\begin{aligned}\Phi^{(2)}(\mathbf{x}^{(2)}(t_0^{(2)}), t_0^{(2)}, \mathbf{x}^{(2)}(t_f^{(2)}), t_f^{(2)}) &= \mathbf{x}^T(t_f)\mathbf{x}(t_f)\\ \mathcal{L}^{(2)}(\mathbf{x}^{(2)}(t), \mathbf{u}^{(2)}(t), t) &= \mathbf{u}^T\mathbf{R}\mathbf{u}\end{aligned}$$

Then the syntax of the above cost functional is given as follows:

```
void MayerCost(SolCost&mySolcost, double& mayer)
{
    double t0 = mySolcost.initial_time_;
    double tf = mySolcost.terminal_time_;
    vec x0 = mySolcost.initial_state_;
    vec xf = mySolcost.terminal_state_;
    mat p = mySolcost.parameter_;
    size_t iphase = mySolcost.phase_num_;
    mat Q={{5,0},{0,2}};//arma version 5.200+ need C++11
    mat R={{1,0},{0,3}};
    mat S={{1,5},{5,1}};
    if(iphase==1)
    {
        mayer=dot(xf,S*XF);
    }
    else if(iphase==2)
    {
        mayer=dot(xf,xf);
    }
}
```

```
    void LagrangeCost(SolCost& mySolcost, vec& lagrange)
    {

    mat x = mySolcost.state_;
    mat u = mySolcost.control_;
    vec t = mySolcost.time_;
    mat p = mySolcost.parameter_;
    size_t iphase = mySolcost.phase_num_;
    assert(iphase == 1 );
    if()
    {
        lagrange=dot(x,x*trans(Q))+dot(u,u*trans(Q));
    }
    else if()
```

```
        {
            lagrange=dot(u,u*trans(R));
        }

        }
```

It is noted in the above function call that the third argument in the command **dot** takes the dot product across the *rows*, thereby producing a *column vector*.

- Differential-Algebraic Equations Functions
  `void DaeFunction(SolDae& mySolDae, mat& stateout, mat& pathout)`
  User-defined differential-algebraic equations **SolDae, input argument**, Here are the members of SolCost

  - `int SolDae.phase_num_`
    the index of phase,start from 1,not 0!
  - `vec SolDae.time_`
    a column vector of length N that contains the time (excluding terminal time)
  - `mat SolDae.state_`
    a matrix of size N*nstate ,contains the values of states(excluding terminal state)
  - `mat SolDae.control_`
    a matrix of size N*ncontrol,cointains the values of controls(excluding terminal state)
  - `mat SolDae.parameter_`
    a column vector of length q,contains the values of the static parameter.

  stateout, output argument
  An arma::mat of size $N * nstate$ containing the values of the right-hand side of the n differential equations.The `stateout.col(i)` is *ith* differential eqquations.
  pathout, output argument
  An arma::mat of size $N * npaths$ containing the values of the path equations n differential equations.The `pathout.col(i)` is *ith* path equations.Leave it out,if the problem doesn't have path differencial equations
  Example of a Differential-Algebraic Equation

  Suppose we have a two-phase optimal control problem , further that the dimension of the state in each phase is 2, the dimension of the control in each phase is 2. Furthermore, suppose that there are no path constraints in phase 1 and one path constraint in phase 2. Next, suppose that the differential equations in phase 1 are given as

$$\begin{aligned}\dot{x}_1 &= -x_1^2 - x_2^2 + u_1 u_2\\ \dot{x}_2 &= -x_1 x_2 + 2(u_1 + u_2)\end{aligned}$$

  Also, suppose that the differential equations in phase 2 are given as

$$\begin{aligned}\dot{x}_1 &= \sin(x_1^2 + x_2^2) + u_1 u_2^2\\ \dot{x}_2 &= -\sin x_1 \cos x_2 + 2u_1 u_2\end{aligned}$$

  Finally, suppose that the path constraint in phase 2 is given as

$$u_1^2 + u_2^2 = 1$$

```
void DaeFunction(SolDae& mySolDae, mat& stateout, mat& pathout)
{
   vec t = mySolDae.time_;
   mat x = mySolDae.state_;
   mat u = mySolDae.contol_;
   mat p = mySolDae.parameter_;
   size_t iphase=mySolDae.phase_num_;
   if(iphase==1)
```

```
        {
            vec x1dot=-x.col(0)%x.col(0)-x.col(1)%x.col(1)
                        +u.col(01)%u.col(1);
            vec x2dot=-x.col(0)*x.col(1)+2*(u.col(0))%u.col(1);
            stateout=join_rows(x1dot,x2dot);
        }else if(iphase==2)
        {
            vec x1dot=sin(x.col(0)%x.col(0)+x.col(1)%x.col(1)) +u.col(01)%u.col(1) %u.col(1);
            vec x2dot=-sin(x.col(0))%cos(x.col(1))+2*u.col(0)%u.col(1);
            vec path=u.col(0)%u.col(0)+u.col(1)%u.col(1);
            stateout=join_rows(x1dot,x2dot);
            pathout=path;
        }

    }
```

- Event Constraint Functions
  `void EventFunction(SolEvent& mySolEvent, vec& eventout)`
  User-defined Lagrange cost functional
  SolEvent, input argument, Here are the members of SolEvent
  - `int SolEvent.phase_num_`
    the index of phase,start from 1,not 0!
  - `double SolEvent.initial_time_`
    the initial timme
  - `vec SolEvent.initial_state_`
    a column vector of length nstate
  - `double SolEvent.terminal_time_`
    the terminal time
  - `vec SolEvent.terminal_state_`
    a column vector of length nstate
  - `mat SolEvent.parameter_`
    a column vector of length q,contains the values of the static parameter.

  eventout, output argument
  An arma::vec,column vector of size $nevents * 1$ containing the values of event functions.Leave it out,if the problem doesn't have path differencial equations.
  Example of Event Constraints with Derivatives

  Suppose we have a one-phase optimal control problem that has two initial event constraints and three terminal event constraints. Suppose further that the number of states in the phase is six. Finally, let the two initial event constraints be given as

$$\begin{aligned}\phi_{01} &= x_1(t_0)^2 + x_2(t_0)^2 + x_3(t_0)^2 \\ \phi_{02} &= x_4(t_0)^2 + x_5(t_0)^2 + x_6(t_0)^2\end{aligned}$$

  while the three terminal event constraints are given as

$$\begin{aligned}\phi_{f1} &= \sin(x_1(t_f))\cos(x_2(t_f) + x_3(t_f)) \\ \phi_{f2} &= \tan(x_4^2(t_f) + x_5^2(t_f) + x_6^2(t_f)) \\ \phi_{f3} &= x_4(t_f) + x_5(t_f) + x_6(t_f)\end{aligned}$$

```
        void EventFunction(SolEvent& mySolEvent, vec& eventout)
        {
            double t0 = mySolcost.initial_time_;
            double tf = mySolcost.terminal_time_;
            vec x0 = mySolcost.initial_state_;
            vec xf = mySolcost.terminal_state_;
            eventout=zeros(5);
            eventout(0)=dot(x0.subvec(0,2),x0.subvec(0,2));
            eventout(1)=dot(x0.subvec(3,5),x0.subvec(3,5));
            eventout(2)=sin(xf(0))*cos(xf(1)+xf(2));
            eventout(3)=tan(dot(xf.subvec(3,5),xf.subvec(3,5)));
            eventout(4)=xf(3)+xf(4)+xf(5);
        }
```

- Linkage Constraint Functions

  `void LinkFunction(SolLink& mySolLink, vec& linkageout)`
  SolLink, input argument, Here are the members of SolLink

  - `int SolLink.left_phase_num_`
    the index of left phase
  - `int SolLink.right_phase_num_`
    the index of right phase
  - `size_t SolLink.ipair`
    the index of linkage,start from 1,not 0!
  - `vec SolLink.left_state_`
    a column vector of length nstate of left phase
  - `vec SolLink.right_state_`
    a column vector of length nstate pf right phase
  - `mat SolLink.left_parameter_`
    a column vector of length q,contains the values of the static parameter in the left phase
  - `mat SolLink.right_parameter_`
    a column vector of length q,contains the values of the static parameter in the right phase

  Example of Linkage Constraint

  Suppose we have a multiple phase optimal control problem with a simple link between the phases, i.e. the state of the end of the phase is equal to the state at the beginning of the next phase.

$$\mathbf{P} = x^l(t_f) - x^r(t_0)$$

```
        void LinkFunction(SolLink& mySolLink, vec& linkageout)
        {
            vec x0_right=mySolLink.right_state_;
            vec xf_left=mySolLink.left_state_;

            links=xf_eft-x0_right;
        }
```

3.4 Derivative

The user has two choices for the computation of the derivative of the objective function gradient and the constraint Jacobian for use within the NLP solver.Since Ipopt acceptes exact Hessian,Lpopc uses sparse finite-difference methods[6](4.4) for computation of the derivative[5].Set option **"first-derive"** with **"finite-difference"**(default) or **analytic**

3.4.1 Finite-difference Method for Jacobian(Default)

Lpopc uses **forward** finite difference method to calculat the derivative of each function. The default value of perturbation h is 10^{-6}.User can reset it with option "**finite-difference-tol**"

3.4.2 Analytic Method for Jacobian

Analytic differentiation has the advantage that it is fast and accurate,howerer, it is complex for the user to code.LPOPC Provides an option for checking the user defined derivatives using finite-difference approximation(consume time). Set option **"analytic-derive-check"** to **"yes"**(default value is **"no"**) to start checking user defined derivatives.

- Derivative of Mayer Function

```
void DerivMayer(SolCost&mySolcost, rowvec& deriv_mayer);
```

deriv_mayer is an arma::rowvec of size $1 \times (2n+2+q)$,define the partial derivatives of the Mayer cost with respect to the initial state, initial time, final state, final time, and finally the parameters

$$\text{deriv\_mayer} = \begin{bmatrix} \frac{\partial \Phi}{\partial \mathbf{x}(t_0)} & \frac{\partial \Phi}{\partial t_0} & \frac{\partial \Phi}{\partial \mathbf{x}(t_f)} & \frac{\partial \Phi}{\partial t_f} & \frac{\partial \Phi}{\partial p} \end{bmatrix}$$

where

$$\begin{aligned} \frac{\partial \Phi}{\partial \mathbf{x}(t_0)} &\in \mathbb{R}^{1\times n} \\ \frac{\partial \Phi}{\partial t_0} &\in \mathbb{R} \\ \frac{\partial \Phi}{\partial \mathbf{x}(t_f)} &\in \mathbb{R}^{1\times n} \\ \frac{\partial \Phi}{\partial t_f} &\in \mathbb{R} \\ \frac{\partial \Phi}{\partial \mathbf{p}} &\in \mathbb{R}^{1\times q} \end{aligned} \tag{6}$$

- Derivative of Lagrange Function

```
void DerivLagrange(SolCost& mySolcost, mat& deriv_lagrange);
```

deriv_lagrange is an arma::mat of size $N \times (n+m+q+1)$, defines the partial derivatives of the Lagrange cost with respect to the state, control, parameters, and time at each of the N LGR points:

$$\text{deriv\_lagrange} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \mathbf{x}} & \frac{\partial \mathcal{L}}{\partial \mathbf{u}} & \frac{\partial \mathcal{L}}{\partial t} & \frac{\partial \mathcal{L}}{\partial p} \end{bmatrix}$$

where

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \mathbf{x}} &\in \mathbb{R}^{N\times n} \\ \frac{\partial \mathcal{L}}{\partial \mathbf{u}} &\in \mathbb{R}^{N\times m} \\ \frac{\partial \mathcal{L}}{\partial t} &= \mathbb{R}^{N\times 1} \\ \frac{\partial \mathcal{L}}{\partial \mathbf{p}} &= \mathbb{R}^{N\times q} \end{aligned} \tag{7}$$

Example of a Cost Functional with Derivatives
Suppose we have a two-phase optimal control problem t, further that the dimension of the state in each phase is 2 while the dimension of the control in each phase is 2. Also, suppose that the endpoint and integrand cost in phase 1 are given, respectively, as

$$\begin{aligned} \Phi^{(1)}(\mathbf{x}^{(1)}(t_0), t_0^{(1)}, \mathbf{x}^{(1)}(t_f), t_f^{(1)}) &= \mathbf{x}^T(t_f)\mathbf{S}\mathbf{x}(t_f) \\ \mathcal{L}^{(1)}(\mathbf{x}^{(1)}(t), \mathbf{u}^{(1)}(t), t) &= \mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u} \end{aligned}$$

while the endpoint and integrand in phase 2 are given, respectively, as

$$\begin{aligned}\Phi^{(2)}(\mathbf{x}^{(2)}(t_0^{(2)}), t_0^{(2)}, \mathbf{x}^{(2)}(t_f^{(2)}), t_f^{(2)}) &= \mathbf{x}^T(t_f)\mathbf{x}(t_f) \\ \mathcal{L}^{(2)}(\mathbf{x}^{(2)}(t), \mathbf{u}^{(2)}(t), t) &= \mathbf{u}^T\mathbf{R}\mathbf{u}\end{aligned}$$

```
void DerivMayer(SolCost&mySolcost, rowvec& deriv_mayer)
{
    double t0 = mySolcost.initial_time_;
    double tf = mySolcost.terminal_time_;
    vec x0 = mySolcost.initial_state_;
    vec xf = mySolcost.terminal_state_;
    mat p = mySolcost.parameter_;
    size_t iphase = mySolcost.phase_num_;
    eriv_mayer = zeros(1, x0.n_elem + 1 + xf.n_elem + 1 + p.n_elem);

    mat Q={{5,0},{0,2}};//arma version 5.200+ need C++11
    mat R={{1,0},{0,3}};
    mat S={{1,5},{5,1}};
    if(iphase==1)
    {
        deriv_mayer.subvec(3,4)=trans(xf)*S;
    }else if(iphase==2)
    {
        deriv_mayer.subvec(3,4)=trans(xf);
    }
}
void DerivLagrange(SolCost& mySolcost, mat& deriv_lagrange)
{
    mat x = mySolcost.state_;
    mat u = mySolcost.control_;
    vec t = mySolcost.time_;
    mat p = mySolcost.parameter_;
    size_t iphase = mySolcost.phase_num_;
    deriv_lagrange = zeros(x.n_rows, x.n_cols + u.n_cols+t.n_cols+p.n_elem);
    if(iphase==1)
    {
        deriv_lagrange.cols(0,1)=x*trans(Q);
        deriv_lagrange.cols(2,3)=u*trans(R);
    }else if(iphase==2)
    {
        deriv_lagrange.cols(2,3)=u*trans(R);
    }
}
```

- Derivative of DAE&Path Function

```
void DerivDae(SolDae& mySolDae, mat& deriv_state, mat& deriv_path);
```

deriv_state is an arma::mat of size $N(n) \times (n + m + q + 1)$
deriv_path is an arma::mat of size $N(c) \times (n + m + q + 1)$

$$\text{derive\_state} = \begin{bmatrix} \dfrac{\partial f_1}{\partial \mathbf{x}} & \dfrac{\partial f_1}{\partial \mathbf{u}} & \dfrac{\partial f_1}{\partial t} & \dfrac{\partial f_1}{\partial \mathbf{p}} \\ \vdots & \vdots & \vdots & \vdots \\ \dfrac{\partial f_2}{\partial \mathbf{x}} & \dfrac{\partial f_2}{\partial \mathbf{u}} & \dfrac{\partial f_2}{\partial t} & \dfrac{\partial f_2}{\partial \mathbf{p}} \\ \vdots & \vdots & \vdots & \vdots \\ \dfrac{\partial f_n}{\partial \mathbf{x}} & \dfrac{\partial f_n}{\partial \mathbf{u}} & \dfrac{\partial f_n}{\partial t} & \dfrac{\partial f_n}{\partial \mathbf{p}} \end{bmatrix}$$

$$\text{derive\_path} = \begin{bmatrix} \dfrac{\partial C_1}{\partial \mathbf{x}} & \dfrac{\partial C_1}{\partial \mathbf{u}} & \dfrac{\partial C_1}{\partial t} & \dfrac{\partial C_1}{\partial \mathbf{p}} \\ \vdots & \vdots & \vdots & \vdots \\ \dfrac{\partial C_r}{\partial \mathbf{x}} & \dfrac{\partial C_r}{\partial \mathbf{u}} & \dfrac{\partial C_r}{\partial t} & \dfrac{\partial C_r}{\partial \mathbf{p}} \end{bmatrix}$$

where f_i, $(i = 1, \ldots, n)$ is the right-hand side of the i^{th} differential equation, and C_j, $(j = 1, \ldots, r)$ is the j^{th} path constraint. Each of the elements of deriv_state have the following sizes:

$$\begin{array}{rcll} \dfrac{\partial f_i}{\partial \mathbf{x}} & \in & \mathbb{R}^{N\times n}, & (i = 1, \ldots, n) \\ \dfrac{\partial f_i}{\partial \mathbf{u}} & \in & \mathbb{R}^{N\times m}, & (i = 1, \ldots, n) \\ \dfrac{\partial f_i}{\partial t} & \in & \mathbb{R}^{N\times 1}, & (i = 1, \ldots, n) \\ \dfrac{\partial f_i}{\partial \mathbf{p}} & \in & \mathbb{R}^{N\times q}, & (i = 1, \ldots, n) \\ \dfrac{\partial C_i}{\partial \mathbf{x}} & \in & \mathbb{R}^{N\times n}, & (i = 1, \ldots, r) \\ \dfrac{\partial C_i}{\partial \mathbf{u}} & \in & \mathbb{R}^{N\times m}, & (i = 1, \ldots, r) \\ \dfrac{\partial C_i}{\partial t} & \in & \mathbb{R}^{N\times 1}, & (i = 1, \ldots, r) \\ \dfrac{\partial C_i}{\partial \mathbf{p}} & \in & \mathbb{R}^{N\times q}, & (i = 1, \ldots, r) \end{array} \tag{8}$$

Example of a Differential-Algebraic Equation with Derivatives

Suppose we have a two-phase optimal control problem , further that the dimension of the state in each phase is 2, the dimension of the control in each phase is 2. Furthermore, suppose that there are no path constraints in phase 1 and one path constraint in phase 2. Next, suppose that the differential equations in phase 1 are given as

$$\begin{array}{rcl} \dot{x}_1 & = & -x_1^2 - x_2^2 + u_1 u_2 \\ \dot{x}_2 & = & -x_1 x_2 + 2(u_1 + u_2) \end{array}$$

Also, suppose that the differential equations in phase 2 are given as

$$\begin{array}{rcl} \dot{x}_1 & = & \sin(x_1^2 + x_2^2) + u_1 u_2^2 \\ \dot{x}_2 & = & -\sin x_1 \cos x_2 + 2u_1 u_2 \end{array}$$

Finally, suppose that the path constraint in phase 2 is given as

$$u_1^2 + u_2^2 = 1$$

```
void DerivDae(SolDae& mySolDae, mat& deriv_state, mat& deriv_path)
{
   vec t = mySolDae.time_;
   mat x = mySolDae.state_;
   mat u = mySolDae.contol_;
   mat p = mySolDae.parameter_;
   size_t iphase=mySolDae.phase_num_;
   if(iphase==1)
   {
      vec df1_dx1 = -2*x.col(0);
      vec df1_dx2 = -2*x.col(1);
      vec df1_du1 = u.col(1);
      vec df1_du2 = u.col(0);
      vec df2_dx1 = -x.col(0);
      vec df2_dx2 = -x.col(1);
      vec df2_du1 = 2*ones(size(t));
      vec df2_du2 = 2*ones(size(t));
      deriv_state=zeros(df1_dx1.nrow+df2_dx1.nrow,x.ncol+u.ncol+1+p.nrow);
      deriv_state.col(0)=join_vert(df1_dx1,df2_dx1);
      deriv_state.col(1)=join_vert(df1_dx2,df2_dx2);
      deriv_state.col(2)=join_vert(df1_du1,df2_du1);
      deriv_state.col(3)=join_vert(df1_du2,df2_du2);

   }else if(iphase==2)
   {
      vec df1_dx1 = 2*x.col(0)%cos(x.col(0)%x.col(0)+ x.col(1)%x.col(1));
      vec df1_dx2 = 2*x.col(1)%cos(x.col(0)%x.col(0)+ x.col(1)%x.col(1));
      vec df1_du1 = u.col(1)%u.col(1);
      vec df1_du2 = 2*u.col(0) %u.col(1);
      vec df2_dx1 = -cos(x.col(0)) %cos(x.col(1));
      vec df2_dx2 = sin(x.col(0)).*sin(x.col(1));
      vec df2_du1 = 2*u.col(1);
      vec df2_du2 = 2*u.col(0);
      vec dpath_du1 = 2*u.col(0);
      vec dpath_du2 = 2*u.col(1);

      deriv_state=zeros(df1_dx1.nrow+df2_dx1.nrow,
                        x.ncol+u.ncol+1+p.nrow);
      deriv_state.col(0)=join_vert(df1_dx1,df2_dx1);
      deriv_state.col(1)=join_vert(df1_dx2,df2_dx2);
      deriv_state.col(2)=join_vert(df1_du1,df2_du1);
      deriv_state.col(3)=join_vert(df1_du2,df2_du2);

      deriv_path=zeros(dpath_du1.nrow,x.ncol+u.ncol+1+p.nrow);
      deriv_path.col(2)=dpath_du1;
      deriv_path.col(3)=dpath_du2;
   }
}
```

- Derivative of Event Function

```
void DerivEvent(SolEvent& mySolEvent, mat& deriv_event);
```

deriv_event is an arma::mat of size $e \times (2n + 2 + q)$

$$\text{deriv\_event} = \begin{bmatrix} \frac{\partial \phi_1}{\partial \mathbf{x}(t_0)}, & \frac{\partial \phi_1}{\partial t_0}, & \frac{\partial \phi_1}{\partial \mathbf{x}(t_f)}, & \frac{\partial \phi_1}{\partial t_f}, & \frac{\partial \phi_1}{\partial p} \\ \vdots, & \vdots, & \vdots, & \vdots, & \vdots \\ \frac{\partial \phi_e}{\partial \mathbf{x}(t_0)}, & \frac{\partial \phi_e}{\partial t_0}, & \frac{\partial \phi_e}{\partial \mathbf{x}(t_f)}, & \frac{\partial \phi_e}{\partial t_f}, & \frac{\partial \phi_e}{\partial p} \end{bmatrix}$$

where ϕ_i, $(i = 1, \ldots, e)$ is the i^{th} event constraint. The sizes of each of the entries in deriv_event are as follows:

$$\begin{array}{rcl} \frac{\partial \phi_i}{\partial \mathbf{x}(t_0)} & \in & \mathbb{R}^{1\times n} \\ \frac{\partial \phi_i}{\partial t_0} & \in & \mathbb{R} \\ \frac{\partial \phi_i}{\partial \mathbf{x}(t_f)} & \in & \mathbb{R}^{1\times n} \quad , \quad (i = 1, \ldots, e) \\ \frac{\partial \phi_i}{\partial t_f} & \in & \mathbb{R} \\ \frac{\partial \Phi}{\partial \mathbf{p}} & \in & \mathbb{R}^{1\times q} \end{array} \tag{9}$$

Example of Event Constraints with Derivatives

Suppose we have a one-phase optimal control problem that has two initial event constraints and three terminal event constraints. Suppose further that the number of states in the phase is six. Finally, let the two initial event constraints be given as

$$\begin{array}{rcl} \phi_{01} & = & x_1(t_0)^2 + x_2(t_0)^2 + x_3(t_0)^2 \\ \phi_{02} & = & x_4(t_0)^2 + x_5(t_0)^2 + x_6(t_0)^2 \end{array}$$

while the three terminal event constraints are given as

$$\begin{array}{rcl} \phi_{f1} & = & \sin(x_1(t_f))\cos(x_2(t_f) + x_3(t_f)) \\ \phi_{f2} & = & \tan(x_4^2(t_f) + x_5^2(t_f) + x_6^2(t_f)) \\ \phi_{f3} & = & x_4(t_f) + x_5(t_f) + x_6(t_f) \end{array}$$

```
void DerivEvent(SolEvent& mySolEvent, mat& deriv_event)
{
    double t0 = mySolcost.initial_time_;
    double tf = mySolcost.terminal_time_;
    vec x0 = mySolcost.initial_state_;
    vec xf = mySolcost.terminal_state_;

    rowvec dei1_dx10 = 2*trans(x0.subvec(0,2));

    rowvec dei2_dx20 = 2*trans(x0.subvec(3,5));

    rowvec def1_dx1f(3);

    def1_dx1f(0)=cos(xf(0))*cos(xf(1)+xf(2));

    def1_dx1f(1)=-sin(xf(0))*sin(xf(1)+xf(2));

    def1_dx1f(2)=-sin(xf(0))*sin(xf(1)+xf(2));

    rowvec def2_dx2f = 2*trans(xf.subvec(3:5))/(trans((cos(dot(xf.subvec(3,5),
    xf.subvec(3,5)))))%trans((cos(dot(xf.subvec(3,5),xf.subvec(3,5))))));

    rowvec def3_dx2f = ones(1,3);
    deriv_event=zeros(3,x0.ncol+1+xf.ncol+1);
```

```
        rowvwc direv_row1=zeros(3,x0.ncol+1+xf.ncol+1);
        rowvwc direv_row2=zeros(3,x0.ncol+1+xf.ncol+1);
        rowvwc direv_row3=zeros(3,x0.ncol+1+xf.ncol+1);

        direv_row1.subvec(0,2)=dei1_dx10;
        direv_row1.subvec(7,9)=def1_dx1f;

        direv_row2.subvec(3,5)=dei2_dx20;
        direv_row2.subvec(10,12)=def2_dx2f;

        direv_row3.subvec(10,12)=def3_dx2f;

        deriv_event.row(1)=direv_row1;
        deriv_event.row(2)=direv_row2;
        deriv_event.row(3)=deriv_row3;
    }
```

- Derivative of Linkage Function

```
    void DerivLink(SolLink& mySolLink, mat& derive_link);
```

derive_link is an arma::mat of size $l \times (n^l + q^l + n^r + q^r)$ where l is the number of linkages in the constraint, n^l is the number of states in the left phase, q^l is the number of parameters in the left phase, n^r is the number of states in the right phase, and q^r is the number of parameters in the right phase.

$$\text{derive\_link} = \begin{bmatrix} \dfrac{\partial \mathbf{P}_1}{\partial \mathbf{x}^l(t_f)}, & \dfrac{\partial \mathbf{P}_1}{\partial p^l}, & \dfrac{\partial \mathbf{P}_1}{\partial \mathbf{x}^r(t_0)}, & \dfrac{\partial \mathbf{P}_1}{\partial p^r} \\ \vdots, & \vdots, & \vdots, & \vdots \\ \dfrac{\partial \mathbf{P}_l}{\partial \mathbf{x}^l(t_f)}, & \dfrac{\partial \mathbf{P}_l}{\partial p^l}, & \dfrac{\partial \mathbf{P}_l}{\partial \mathbf{x}^r(t_0)}, & \dfrac{\partial \mathbf{P}_l}{\partial p^r} \end{bmatrix}$$

where $\mathbf{P}_i$, $(i = 1, \ldots, l)$ is the i^{th} linkage constraint. It is important to provide all the derivatives in the correct order *even if* they are zero.

Example of Linkage Constraint with Derivatives

Suppose we have a multiple phase optimal control problem with a simple link between the phases, i.e. the state of the end of the phase is equal to the state at the beginning of the next phase.

$$\mathbf{P} = x^l(t_f) - x^r(t_0)$$

```
        void DerivLink(SolLink& mySolLink, mat& derive_link)
        {
            vec x0_right=mySolLink.right_state_;
            vec xf_left=mySolLink.left_state_;

            derive_link=zeros(xf_left.n_elem,xf_left.n_elem+x0_right.n_elem);

            derive_link.cols(0, xf_left.n_elem - 1) = - eye<mat>(xf_left.n_elem,
             xf_left.n_elem);
            derive_link.cols(xf_left.n_elem, xf_left.n_elem + x0_right.n_elem - 1)
            = eye<mat>(x0_right.n_elem, x0_right.n_elem);
        }
```

Here n is the number of states, m is the number of controls, q is the number of parameters, c is the number of path constraints,e is the number of event constraints and N is the number of LGR points in the phase.

3.4.3 Hessian

Set option **"hessian-appromation"** to **"exact"**,LPOPC calculates Hessian matrix with sparse finite difference[6], to**"limited-memory"**(defalut) LPOPC won't provide hessian message for IPOPT.
Note
Using exact hessian will accelerate the calculation in IPOPT a lot in theroy.But when the functions of optimal problem is very complex, it will cnsume a lot time.If so,set **"hessian-appromation"** to**"limited-memory"** may be the better choice.

3.4.4 Function Dependencies

The NLP associated with thee Radau collocatio method has a sparse structure,blocks of the constraint Jacobian and Lagrangin Hessian are dependent upon whether a particular NLP function depends upon a particular NLP variable[5].LPOPC detect the sparse struct automatically using **sparse-NaN** methods[6].

3.5 Initial Guess

Set the initial Guess phase by phase.Here is phase 1 as an example.

- Set time

```
        Phase1->SetTimeGuess(t0);
        Phase1->SetTimeGuess(tf);
```

- Set state one by one

```
        Phase1->SetStateGuess(1, x10);//count from 1
        Phase1->SetStateGuess(1, x1f);//count from 1
        Phase1->SetStateGuess(2, x20);//count from 1
        Phase1->SetStateGuess(2, x2f);//count from 1
```

  The first paramemter is the index of the state,it's 1 mean you are setting x_1.You should set a guess value for every time point.Here the time points are t_0 and t_f,so,we just set x_0 and x_f.

- Set control one by one

```
        Phase1->SetControlGuess(1, -1);
        Phase1->SetControlGuess(1, 1);
```

  The control is same with state.If the problem has more than one control,just do the same thing for every control.

- Set parameter one by one

```
        Phase1->SetparameterGuess(u0);
        Phase1->SetparameterGuess(uf);
```

  If the problem has more than one parameter,just do the same thing for every parameter.

Example of Specifying Initial Guess
Suppose we have a problem that has two states and two controls in phase 1 while it has two states and one control in phase 2. Furthermore, suppose that we choose four time points for the guess in phase 1.

```
        Phase1->SetTimeGuess(0);
        Phase1->SetTimeGuess(1);
        Phase1->SetTimeGuess(3);
        Phase1->SetTimeGuess(5);

        Phase1->SetStateGuess(1,1.27);
        Phase1->SetStateGuess(1,3.1);
        Phase1->SetStateGuess(1,5.8);
        Phase1->SetStateGuess(1,9.6);

        Phase1->SetStateGuess(2,-4.2);
        Phase1->SetStateGuess(2,-9.6);
        Phase1->SetStateGuess(2,8.5);
        Phase1->SetStateGuess(2,-9.6);

        Phase1->SetControlGuess(1,0.1);
        Phase1->SetControlGuess(1,0.3);
        Phase1->SetControlGuess(1,0.5);
        Phase1->SetControlGuess(1,0.7);
```

3.6 Mesh Refinement

LPOPC implements the "**ph**" mesh refinement algorithm[4].
Note,the solution error is about the differencial funnction,the error of path function is not calculated.
User can set the **Nmax**(default value is 16) and **Nmin**(default value is 4) used in "**ph**" algorithm.
The "**hp**" mesh refinement algorithm will be added in next version.

Set First Grid*

- mesh points a monotonically increasing row vector of length M_p, $(p \in [1, \ldots, P])$, where each entry in the vector is on the domain $[-1, +1]$, that contains a set of mesh points for the initial run of LPOPC. If the user does not have an estimate of the mesh point locations, this field should be left blank.
  Default is $(-1, 1)$
- nodes perinterval a row vector of length $M_p - 1$, $(p \in [1, \ldots, P])$, where each entry in the vector is a positive integer that contains the number of collocation points in each mesh interval for the initial run of LPOPC. If the user does not have an estimate of the number of collocation points in each mesh interval, this entry should be left blank.
  Default is 10.

Example for Setting First Grid

Suppose that we choose to initialize LPOPC with a mesh consisting of six mesh points $(-1, -0.75, -0.5, 0, 0.5, 0.75, +1)$ with the 2, 4, 4, 3, and 2 collocation points in the first through fifth mesh intervals, respectively.

```
      Phase1->SetMeshPoints(-1);
      Phase1->SetMeshPoints(-0.75);
      Phase1->SetMeshPoints(-0.5);
      Phase1->SetMeshPoints(0.0);
      Phase1->SetMeshPoints(0.75);
      Phase1->SetMeshPoints(1);

      Phase1->SetNodesPerInterval(2)
      Phase1->SetNodesPerInterval(4)
      Phase1->SetNodesPerInterval(4)
      Phase1->SetNodesPerInterval(3)
```

```
    Phase1->SetNodesPerInterval(2)
```

Note In this vision,do not set the number of nodes beyond $[Nmin, Nmax]$.

3.7 Scaling*

The auto scale procedure is based on the scaling algorithm in[2] and still under test.Set the option **"auto-scale"** to **"yes"** to try it(default is **"no"**).
Warninng
Scaling by hand is the best choice.The automatic scaling procedure is by no means fullproof, it works well on many problems,it fails on some problems.Auto-scale is slower than manual scale.

3.8 Solve Problem

Set all information LPOPC needed,using `app->SolveOptimalProblem();` to solve your problem. If the problem is solved successfully,the result will be saved in several fils.

3.9 Output File

If user's optimal control problem is solvedLPOPC will save the result with ascii txt format.The following fils will be overwritten erery timeLPOPC is called.

- lpopc-main-msg.txt, main output file of LPOPC.
- timeX,the X is the index of phase,start from 1,the time points of phase X,the number of elements is $ntime$ =the number of LGR points+1.
- stateX,the X is the index of phase,start from 1,the state of phase X,every column is a state,containing $ntime$ state points corresponding to every time point.
- controlX,the X is the index of phase,start from 1,the control of phase X,same with stateX.
- parameterX,the X is the index of phase,start from 1,the parameter of phase X,same with stateX.
- costate1X,the X is the index of phase,start from 1,the costate of phase X,same with stateX.
- HamiltonianX,the X is the index of phase,start from 1,the Hamiltonian of phase X,same with stateX.
- grid-XIpopt-out.txt,this is the msg IPOPT returned when LPOPC solves mesh X+1,the X is start from 0.
- *lpopc-debug-msg.txt,when start with debug mode,this file traces which function inside LPOPC is called and when the function is called.This message is very helpful to find bugs.

3.10 Options Reference

The options of LPOPC can be classified in three types,Number(double),Integer(int) and String(std::string). Set option using the code below

```
  app->Options()->SetNumberValue(option-name,option-value);

  app->Options()->SetIntegerValue(option-name,option-value);

  app->Options()->SetStringValue(option-name,option-value);
```

Here,the option-name is a std::string.

- Ipopt-tol
  Type Number
  Default 10^{-6}

- first-derive
  Type String
  Value "finite-difference","analytic"
  Default "finite-difference"

- finite-difference-tol Type Number
  Default 10^{-6}

- analytic-derive-check Type String
  Value "yes" "no"
  Defalut "no"

- analytic-derive-check-tol
  Type Number
  Default 10^{-6}

- hessian-approximation
  Type String
  Value "exact","limited-memory"
  Default "limited-memory"

- mesh-refine-methods
  Type String
  Value "ph"
  Default "ph"

- max-grid-num
  Type Integer
  Default 10

- desired-relative-error
  Type Number
  Default 10^{-6}

- Nmax
  Type Integer
  Default 16

- Nmin
  Type Integer
  Default 4

- auto-scale(testing)
  Type String
  Value "yes","no"
  Default "no"

4 Examples

4.1 Hyper-Sensitive Problem

Consider the following optimal control problem[8].This problem is used to test the mesh refinement algorithm in[4]. Minimize the cost functional

$$J = \frac{1}{2}\int_0^{t_f} \left(x^2 + u^2\right) dt \tag{10}$$

subject to the dynamic constraint

$$\dot{x} = -x^3 + u \tag{11}$$

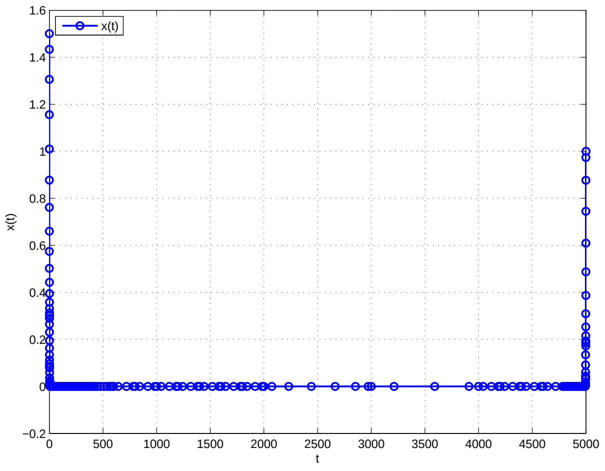
and the boundary conditions

$$\begin{array}{rcl} x(0) & = & 1.5 \\ x(t_f) & = & 1 \end{array} \tag{12}$$

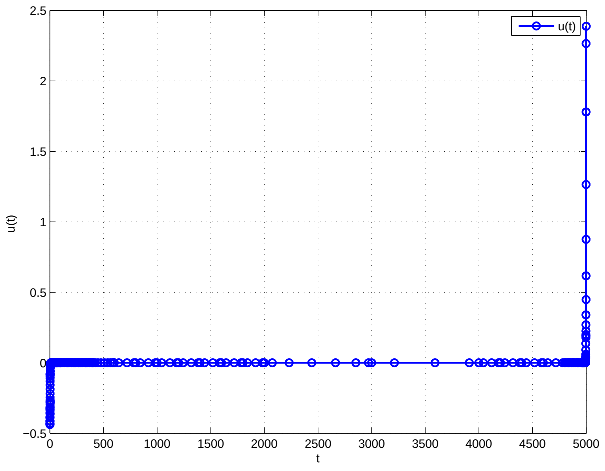
with $t_f = 5000$.
The LPOPC code that solves this problem is shown below.

```
// Copyright (C) 2014-2015 Xue Zhichen, Wang Yujie,Wang Na
// All Rights Reserved.
// This file is a part of LPOPC , published under the Eclipse Public License.
// Author:Xue Zhichen 7/15 2015 11:11
// Email:eddy_lpopc@163.com
#ifndef HYPERSENSITIVE_H
#define HYPERSENSITIVE_H
#include "LpFunctionWrapper.h"
using namespace Lpopc;
class HyperSensitiveFunction :public FunctionWrapper
{
public:
    HyperSensitiveFunction(){}
    virtual void MayerCost(SolCost&mySolcost, double& mayer);
    virtual void DerivMayer(SolCost&mySolcost, rowvec& deriv_mayer);

    virtual void LagrangeCost(SolCost& mySolcost, vec& langrange);
    virtual void DerivLagrange(SolCost& mySolcost, mat& deriv_langrange);

    virtual void DaeFunction(SolDae& mySolDae, mat& stateout, mat& pathout);
    virtual void DerivDae(SolDae& mySolDae, mat& deriv_state, mat& deriv_path);

    virtual void EventFunction(SolEvent& mySolEvent, vec& eventout);
    virtual void DerivEvent(SolEvent& mySolEvent, mat& deriv_event);

    virtual void LinkFunction(SolLink& mySolLink, vec& linkageout);
    virtual void DerivLink(SolLink& mySolLink, mat& derive_link);
    virtual ~HyperSensitiveFunction(){}
};


#endif // !HYPERSENSITIVE_H
```

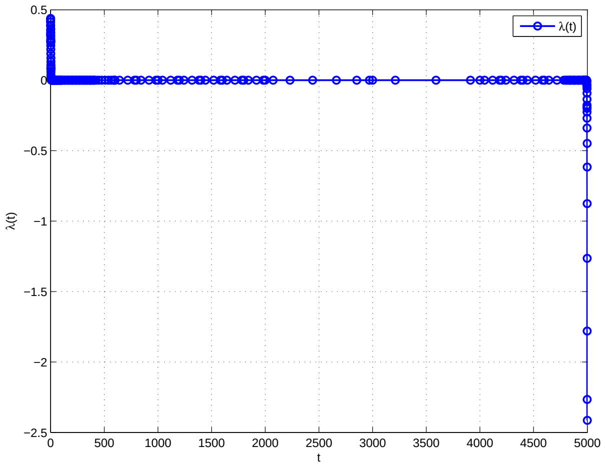
```
// Copyright (C) 2014-2015 Xue Zhichen, Wang Yujie,Wang Na
// All Rights Reserved.
// This file is a part of LPOPC , published under the Eclipse Public License.
// Author:Xue Zhichen 7/14 2015 13:16
// Email:eddy_lpopc@163.com

///only need including two headfils
#include "hypersensitive.h"
#include "LpLpopcApplication.hpp"


////////////////////////////////////////////////////////////////////////////
using namespace Lpopc;
int main(void)
{

    double t0 = 0.0, tf = 5000, x0 = 1.5, xf = 1;
    double xmin = -10, xmax = 10, umin = -10, umax = 10;


    //Step 1
    shared_ptr<LpopcApplication> app(new LpopcApplication(console_print));

    //Step2
    shared_ptr<Phase> Phase1(new Phase(1, 1, 1, 0, 0, 0));
    //Set bounds
    Phase1->SetTimeMin(t0, tf);
    Phase1->SetTimeMax(t0, tf);
    Phase1->SetStateMin(x0, xmin, xf);
    Phase1->SetStateMax(x0, xmax, xf);
    Phase1->SetcontrolMin(umin);
    Phase1->SetcontrolMax(umax);

    ///Set guess
    Phase1->SetTimeGuess(t0);
    Phase1->SetTimeGuess(tf);
    Phase1->SetStateGuess(1, x0);//count from 1
    Phase1->SetStateGuess(1, xf);//count from 1
    Phase1->SetControlGuess(1, -1);
    Phase1->SetControlGuess(1, 1);

    //Step3
    shared_ptr<FunctionWrapper> userfun(new HyperSensitiveFunction());//declare optimal problem function
    shared_ptr<OptimalProblem> optpro(new OptimalProblem(1, 0, userfun));
    optpro->AddPhase(Phase1);


    LP_DBG_START_FUN("main")
    try
    {
        //Step 4
        app->SetOptimalControlProblem(optpro);
        app->Options()->SetStringValue("hessian-approximation", "exact"); //use sparse-finite hessian
        app->Options()->SetStringValue("first-derive", "analytic");//use analytic derive

        app->Options()->SetIntegerValue("max-grid-num", 20); //maximum of mesh refinement

        app->SolveOptimalProblem();// Solve the problem
    }
    catch (LpopcException & e)
    {
        std::cout << "got␣an␣error" << std::endl;
    }
    catch (...)
    {
        std::cout << "got␣an␣error" << std::endl;
    }

    return 0;
}


void HyperSensitiveFunction::MayerCost(SolCost&mySolcost, double& mayer)
{
    vec x0 = mySolcost.initial_state_;
    vec xf = mySolcost.terminal_state_;
    double t0 = mySolcost.initial_time_;
    double tf = mySolcost.terminal_time_;
    mat p = mySolcost.parameter_;
    int iphase = mySolcost.phase_num_;
    assert(iphase == 1 );
    mayer= 0.0;
}

void HyperSensitiveFunction::DerivMayer(SolCost&mySolcost, rowvec& deriv_mayer)
{
    double t0 = mySolcost.initial_time_;
    double tf = mySolcost.terminal_time_;
    vec x0 = mySolcost.initial_state_;
    vec xf = mySolcost.terminal_state_;
    mat p = mySolcost.parameter_;
    size_t iphase = mySolcost.phase_num_;
    assert(iphase == 1 );
    deriv_mayer = zeros(1, x0.n_elem + 1 + xf.n_elem + 1 + p.n_elem);
}

void HyperSensitiveFunction::LagrangeCost(SolCost& mySolcost, vec& langrange)
{

    mat x = mySolcost.state_;
    mat u = mySolcost.control_;
    vec t = mySolcost.time_;
    mat p = mySolcost.parameter_;
    size_t iphase = mySolcost.phase_num_;
    assert(iphase == 1 );
    langrange =0.5*( x%(x) + u%(u));

}

void HyperSensitiveFunction::DerivLagrange(SolCost& mySolcost, mat& deriv_langrange)
{
    mat x = mySolcost.state_;
    mat u = mySolcost.control_;
    vec t = mySolcost.time_;
    mat p = mySolcost.parameter_;
    size_t iphase = mySolcost.phase_num_;
    deriv_langrange = zeros(x.n_rows, x.n_cols + u.n_cols+t.n_cols+p.n_elem);
    deriv_langrange.cols(0, x.n_cols-1) = x;
    deriv_langrange.cols(x.n_cols, x.n_cols + u.n_cols - 1) = u;
}

void HyperSensitiveFunction::DaeFunction(SolDae& mySolDae, mat& stateout, mat& pathout)
{
    vec t = mySolDae.time_;
    mat x = mySolDae.state_;
    mat u = mySolDae.contol_;
    mat p = mySolDae.parameter_;
    size_t iphase = mySolDae.phase_num_;
    assert(iphase == 1 );
    stateout = -x%x%x + u;

}

void HyperSensitiveFunction::DerivDae(SolDae& mySolDae, mat& deriv_state, mat& deriv_path)
{
    vec t = mySolDae.time_;
    mat x = mySolDae.state_;
    mat u = mySolDae.contol_;
    mat p=mySolDae.parameter_;

    mat df_dx = -3*(x%x);
    mat df_du = ones(u.n_rows, u.n_cols);
    mat df_dt = zeros(t.n_rows, t.n_cols);

    deriv_state = zeros(df_dx.n_rows, df_dx.n_cols + df_du.n_cols + df_dt.n_cols);
    deriv_state.cols(0, df_dx.n_cols - 1) = df_dx;
    deriv_state.cols(df_dx.n_cols, df_dx.n_cols + df_du.n_cols - 1) = df_du;
    deriv_state.cols(df_dx.n_cols + df_du.n_cols, deriv_state.n_cols - 1) = df_dt;
}
void HyperSensitiveFunction::EventFunction(SolEvent& mySolEvent, vec& eventout)
{
}

void HyperSensitiveFunction::DerivEvent(SolEvent& mySolEvent, mat& deriv_event)
{

}

void HyperSensitiveFunction::LinkFunction(SolLink& mySolLink, vec& linkageout)
{
}

void HyperSensitiveFunction::DerivLink(SolLink& mySolLink, mat& derive_link)
{
}
```



(a) State, $x(t)$, vs. t.



(b) Control, $u(t)$, vs. t.



(c) Costate, $\lambda(t)$, vs. t.

4.2 Bryson-DenHam Problem

Consider the following optimal control problem.This problem is known in the literature as the Bryson-Denham problem[3]. Minimize the cost functional

$$J = x_3(t_f) \tag{13}$$

subject to the dynamic constraints

$$\begin{aligned} \dot{x_1} &= x_2 \\ \dot{x_2} &= u \\ \dot{x_3} &= \tfrac{1}{2}u^2 \end{aligned} \tag{14}$$

the path constraint

$$0 \le x_1(t) \le 1/9 \tag{15}$$

and the boundary conditions

$$\begin{aligned} x_1(0) &= 0 \\ x_2(0) &= 1 \\ x_3(0) &= 0 \\ x_1(t_f) &= 0 \\ x_2(t_f) &= -1 \end{aligned} \tag{16}$$

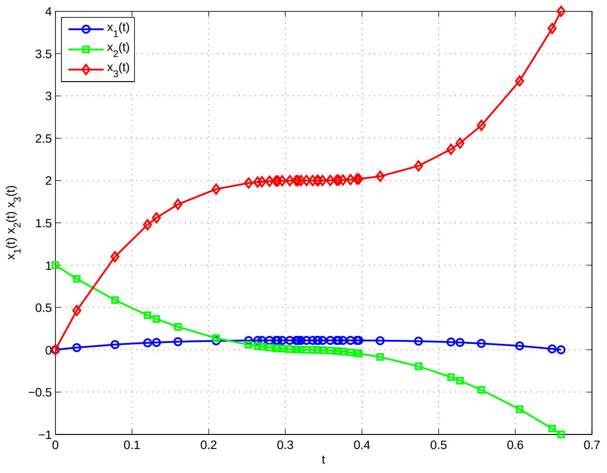
The LPOPC code that solves this problem is shown below.

```
// Copyright (C) 2014-2015 Xue Zhichen, Wang Yujie,Wang Na
// All Rights Reserved.
// This file is a part of LPOPC , published under the Eclipse Public License.
// Author:Xue Zhichen 7/15 2015 11:11
// Email:eddy_lpopc@163.com
#ifndef LPOPC_BRYSON_DENHAM_HPP
#define LPOPC_BRYSON_DENHAM_HPP
#include "LpFunctionWrapper.h"
using namespace Lpopc;
class BrysonDenhamFunction :public FunctionWrapper
{
public:
    BrysonDenhamFunction(){}
    virtual void MayerCost(SolCost&mySolcost, double& mayer);
    virtual void DerivMayer(SolCost&mySolcost, rowvec& deriv_mayer);

    virtual void LagrangeCost(SolCost& mySolcost, vec& langrange);
    virtual void DerivLagrange(SolCost& mySolcost, mat& deriv_langrange);

    virtual void DaeFunction(SolDae& mySolDae, mat& stateout, mat& pathout);
    virtual void DerivDae(SolDae& mySolDae, mat& deriv_state, mat& deriv_path);

    virtual void EventFunction(SolEvent& mySolEvent, vec& eventout);
    virtual void DerivEvent(SolEvent& mySolEvent, mat& deriv_event);

    virtual void LinkFunction(SolLink& mySolLink, vec& linkageout);
    virtual void DerivLink(SolLink& mySolLink, mat& derive_link);
    virtual ~BrysonDenhamFunction(){}
};

#endif // !LPOPC_BRYSON_DENHAM_HPP
```

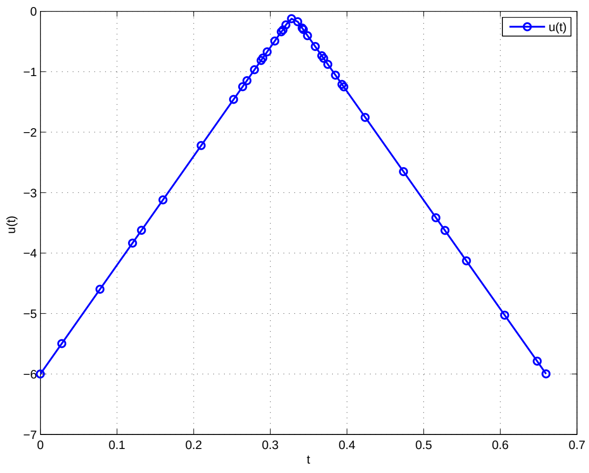
```
// Copyright (C) 2014-2015 Xue Zhichen, Wang Yujie,Wang Na
// All Rights Reserved.
// This file is a part of LPOPC , published under the Eclipse Public License.
// Author:Xue Zhichen 7/15 2015 11:18
// Email:eddy_lpopc@163.com
```

```
#include "BrysonDenham.h"
#include "LpLpopcApplication.hpp"
int main()
{
    double x10 = 0;
    double x20 = -10;
    double x30 = 0;
    double x1f = 0;
    double x2f = -1;
    double x1min = 0;
    double x1max = 1.0 / 9;
    double x2min = -10;
    double x2max = 10;
    double x3min = -10;
    double x3max = 10;
    //Step1
    shared_ptr<LpopcApplication> app(new LpopcApplication(console_print));
    //Step2
    shared_ptr<Phase> phase1(new Phase(1, 3, 1, 0, 0, 5));
    phase1->SetTimeMin(0.0, 0.0);
    phase1->SetTimeMax(0, 50);

    phase1->SetStateMin(0, 0, 0);
    phase1->SetStateMax(1.0/9.0, 1.0/9.0, 1.0/9.0);

    phase1->SetStateMin(-10, -10, -10);
    phase1->SetStateMax(10, 10, 10);

    phase1->SetStateMin(-10, -10, -10);
    phase1->SetStateMax(10, 10, 10);

    phase1->SetcontrolMin(-10);
    phase1->SetcontrolMax(10);

    phase1->SetTimeGuess(0.0);
    phase1->SetTimeGuess(1.0);

    phase1->SeteventMin(0);
    phase1->SeteventMin(1);
    phase1->SeteventMin(0);
    phase1->SeteventMin(0);
    phase1->SeteventMin(-1);

    phase1->SeteventMax(0);
    phase1->SeteventMax(1);
    phase1->SeteventMax(0);
    phase1->SeteventMax(0);
    phase1->SeteventMax(-1);

    phase1->SetStateGuess(1, 0);// count from 1
    phase1->SetStateGuess(1, 0);

    phase1->SetStateGuess(2, 1.0);
    phase1->SetStateGuess(2, -1.0);

    phase1->SetStateGuess(3, 0.0);
    phase1->SetStateGuess(3, 0.0);

    phase1->SetControlGuess(1, 0.0);
    phase1->SetControlGuess(1, 0.0);
    //Step3
    shared_ptr<FunctionWrapper> userfun(new BrysonDenhamFunction());
    shared_ptr<OptimalProblem> optpro(new OptimalProblem(1, 0, userfun));
    optpro->AddPhase(phase1);

        try
    {
        //Step4
        app->SetOptimalControlProblem(optpro);
        app->Options()->SetStringValue("hessian-approximation", "exact");
        app->Options()->SetIntegerValue("max-grid-num", 20);
        app->SolveOptimalProblem();
    }
    catch (LpopcException & e)
    {
        std::cout << "got␣an␣error" << std::endl;
    }
    catch (std::logic_error& e)
    {
        std::cout << "got␣an␣error" << std::endl;
    }
    catch (std::runtime_error& e)
    {
        std::cout << "got␣an␣error" << std::endl;
    }
    catch (std::bad_alloc& e)
    {
        std::cout << "got␣an␣error" << std::endl;
    }
    return 0;
}

void BrysonDenhamFunction::MayerCost(SolCost&mySolcost, double& mayer)
{
    vec xf = mySolcost.terminal_state_;
    mayer = xf(2);
}

void BrysonDenhamFunction::LagrangeCost(SolCost& mySolcost, vec& langrange)
{
    vec t = mySolcost.time_;
    langrange = zeros(t.n_elem);

}

void BrysonDenhamFunction::DaeFunction(SolDae& mySolDae, mat& stateout, mat& pathout)
{
    vec t = mySolDae.time_;
    mat x = mySolDae.state_;
    mat u = mySolDae.contol_;
    mat p = mySolDae.parameter_;
    size_t iphase = mySolDae.phase_num_;
    assert(iphase == 1);
    stateout = zeros(x.n_rows, x.n_cols);
    stateout.col(0) = x.col(1);
    stateout.col(1) = u;
    stateout.col(2) = 0.5*(u%u);

}
```

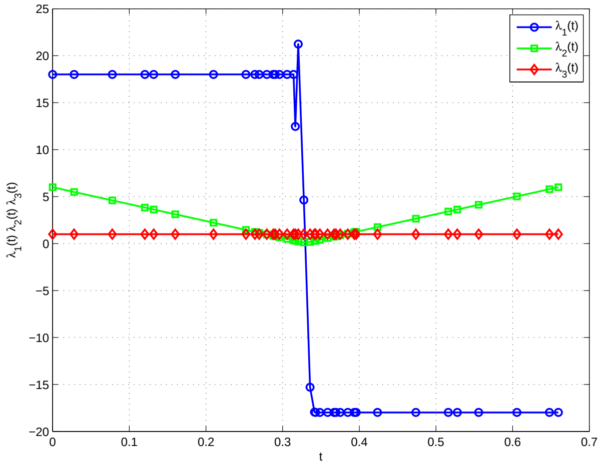
```
void BrysonDenhamFunction::DerivMayer(SolCost&mySolcost, rowvec& deriv_mayer)
{
}

void BrysonDenhamFunction::DerivLagrange(SolCost& mySolcost, mat& deriv_langrange)
{
}

void BrysonDenhamFunction::DerivDae(SolDae& mySolDae, mat& deriv_state, mat& deriv_path)
{
}

void BrysonDenhamFunction::EventFunction(SolEvent& mySolEvent, vec& eventout)
{
    vec x0 = mySolEvent.initial_state_;
    vec xf = mySolEvent.terminal_state_;
    eventout = zeros(5);
    double x10 = x0(0);
    double x20 = x0(1);
    double x30 = x0(2);
    double x1f = xf(0);
    double x2f = xf(1);
    eventout(0) = x10;
    eventout(1) = x20;
    eventout(2) = x30;
    eventout(3) = x1f;
    eventout(4) = x2f;
}

void BrysonDenhamFunction::DerivEvent(SolEvent& mySolEvent, mat& deriv_event)
{
}

void BrysonDenhamFunction::LinkFunction(SolLink& mySolLink, vec& linkageout)
{
}

void BrysonDenhamFunction::DerivLink(SolLink& mySolLink, mat& derive_link)
{
}
```



(a) State vs. Time.



(b) Control vs. Time.



(c) Costate vs. Time.

4.3 Launch

This problem consists of the launch of a space vechicl.See[1]for a full description of the problem.
Only a brief description is given here.The flight of the rocket can be divided into four phases,with dry masses ejected from the vehicle at the end of phases 1,2 and 3.The final times of phase 1,2 and 3 are fixed,while the final time of phase 4 is free. The optimal control problem is then to find the control, u, that minimizes the cost function

$$J = -m^{(4)}(t_f) \tag{17}$$

The equations of motion for a non-lifting point mass in flight over a spherical rotating planet are expressed in Cartesian Earth centered inertial (ECI) coordinates as

$$\begin{aligned} \dot{\mathbf{r}} &= v \\ \dot{\mathbf{v}} &= -\frac{\mu}{\|\mathbf{r}\|^3} r + \frac{T}{m} u + \frac{D}{m} \\ \dot{m} &= -\frac{T}{g_0 I_{sp}} \end{aligned} \tag{18}$$

where $r(t) = \begin{bmatrix} x(t) & y(t) & z(t) \end{bmatrix}^T$ is the position, $v = \begin{bmatrix} v_x(t) & v_y(t) & v_z(t) \end{bmatrix}^T$ is the Cartesian ECI velocity, μ is the gravitational parameter, T is the vacuum thrust, m is the mass, g_0 is the acceleration due to gravity at sea level, I_{sp} is the specific impulse of the engine, $u = \begin{bmatrix} u_x & u_y & u_z \end{bmatrix}^T$ is the thrust direction, and $D = \begin{bmatrix} D_x & D_y & D_z \end{bmatrix}^T$ is the drag force. The drag force is defined as

$$D = -\frac{1}{2} C_D A_{ref} \rho \|v_{rel}\| v_{rel} \tag{19}$$

where C_D is the drag coefficient, A_{ref} is the reference area, ρ is the atmospheric density, and v_{rel} is the Earth relative velocity, where v_{rel} is given as

$$v_{rel} = v - \boldsymbol{\omega} \times r \tag{20}$$

where $\boldsymbol{\omega}$ is the angular velocity of the Earth relative to inertial space. The atmospheric density is modeled as the exponential function

$$\rho = \rho_0 \exp[-h/h_0] \tag{21}$$

where ρ_0 is the atmospheric density at sea level, $h = \|r\| - R_e$ is the altitude, R_e is the equatorial radius of the Earth, and h_0 is the density scale height.

The launch vehicle starts on the ground at rest (relative to the Earth) at time t_0, so that the ECI initial conditions are

$$\begin{aligned} r(t_0) &= r_0 = \begin{bmatrix} 5605.2 & 0 & 3043.4 \end{bmatrix}^T \quad \text{km} \\ v(t_0) &= v_0 = \begin{bmatrix} 0 & 0.4076 & 0 \end{bmatrix}^T \quad \text{km/s} \\ m(t_0) &= m_0 = 301454 \quad \text{kg} \end{aligned} \tag{22}$$

which corresponds to the Cape Canaveral launch site. The terminal constraints define the target geosynchronous transfer orbit (GTO), which is defined in orbital elements as

$$\begin{aligned} a_f &= 24361.14\,\text{km}, \\ e_f &= 0.7308, \\ i_f &= 28.5\,\text{deg}, \\ \Omega_f &= 269.8\,\text{deg}, \\ \omega_f &= 130.5\,\text{deg} \end{aligned} \tag{23}$$

A path constraint is imposed on the control to guarantee that the control vector is unit length, so that

$$|u| = 1 \tag{24}$$

The following linkage constraints force the position and velocity to be continuous and also account for the mass ejections, as

$$\begin{array}{rcl} r^{(p)}(t_f) - r^{(p+1)}(t_0) & = & 0, \\ v^{(p)}(t_f) - v^{(p+1)}(t_0) & = & 0, \qquad (p = 1, \ldots, 3) \\ m^{(p)}(t_f) - m^{(p)}_{dry} - m^{(p+1)}(t_0) & = & 0 \end{array} \tag{25}$$

where the superscript (p) represents the phase number.

The Lpopc code that solves this problem is shown below.

```
// Copyright (C) 2014-2015 Xue Zhichen, Wang Yujie,Wang Na
// All Rights Reserved.
// This file is a part of LPOPC , published under the Eclipse Public License.
// Author:Xue Zhichen 7/14 2015 13:15
// Email:eddy_lpopc@163.com
#ifndef LAUNCH_H
#define LAUNCH_H
#include "LpFunctionWrapper.h"
using namespace Lpopc;
class LaunchFunction:public FunctionWrapper
{
public:
    LaunchFunction(){}
    virtual void MayerCost(SolCost&mySolcost,double& mayer);
    virtual void DerivMayer(SolCost&mySolcost,rowvec& deriv_mayer);

    virtual void LagrangeCost(SolCost& mySolcost,vec& langrange);
    virtual void DerivLagrange(SolCost& mySolcost,mat& deriv_langrange);

    virtual void DaeFunction(SolDae& mySolDae,mat& stateout,mat& pathout);
    virtual void DerivDae(SolDae& mySolDae,mat& deriv_state,mat& deriv_path);

    virtual void EventFunction(SolEvent& mySolEvent,vec& eventout);
    virtual void DerivEvent(SolEvent& mySolEvent,mat& deriv_event);

    virtual void LinkFunction(SolLink& mySolLink,vec& linkageout);
    virtual void DerivLink(SolLink& mySolLink,mat& derive_link);
    virtual ~LaunchFunction(){}
};


#endif // !LAUNCH_H
```

```
// Copyright (C) 2014-2015 Xue Zhichen, Wang Yujie,Wang Na
// All Rights Reserved.
// This file is a part of LPOPC , published under the Eclipse Public License.
// Author:Xue Zhichen 7/14 2015 13:16
// Email:eddy_lpopc@163.com
//
#include "Launch.hpp"
#include "LpLpopcApplication.hpp"

const double PI= datum::pi;
using namespace Lpopc;
 void Launchoe2rv( vec& oe,double mu,vec& ri,vec& vi);
 void Launchrv2oe(vec& oe, vec rv, vec vv, double mu);
 void DLaunchrv2oe(const vec& rv,const vec& vv,double mu,double Re,mat& doe);
 double earthRadius       = 6378145;
    double gravParam        = 3.986012e14;
   double initialMass        = 301454;
   double earthRotRate       = 7.29211585e-5;
   double seaLevelDensity    = 1.225;
   double densityScaleHeight = 7200;
   double g0                 = 9.80665;

   struct struct_scales
   {
       double length;
       double speed;
       double time;
       double acceleration;
       double mass;
       double force;
       double area;
       double volume;
       double density;
       double gravparam;
   }scales={
        earthRadius,
       sqrt(gravParam/scales.length),
       scales.length/scales.speed,
        scales.speed/scales.time,
        initialMass,
       scales.mass*scales.acceleration,
        scales.length*scales.length,
       scales.area*scales.length,
       scales.mass/scales.volume,
       scales.acceleration*scales.length*scales.length
   };

   double omega=earthRotRate*scales.time;

   struct CONSTANTS_struct
   {
       double omega_matrix[9];
       double mu;
       double cd;
       double sa;
       double rho0;
       double H;
       double Re;
       double g0;
       double thrust_srb;
       double thrust_first;
       double thrust_second;
       double ISP_srb;
       double ISP_first;
       double ISP_second;
   }CONSTANTS={
       {0,1*omega,0,-1*omega,0,0,0,0,0},
       gravParam/scales.gravparam,
       0.5,
       4*PI/scales.area,
       seaLevelDensity/scales.density,
       densityScaleHeight/scales.length,
       earthRadius/scales.length,g0/scales.acceleration
   };

int main(void)
{
```

```
double lat0 = 28.5*PI/180;           // Geocentric Latitude of Cape Canaveral
double x0 = CONSTANTS.Re* cos (lat0); // x component of initial position
double z0 = CONSTANTS.Re* sin (lat0);  // z component of initial position
double  y0 = 0;

vec r0(3, 1);
r0(0) = x0; r0(1) = y0;r0(2) =z0;

mat omega_matrix(CONSTANTS.omega_matrix,3,3);

mat v0 = omega_matrix*r0;

double bt_srb = 75.2/scales.time;
double bt_first = 261.0/scales.time;
double bt_second = 700.0/scales.time;

double t0 = 0.0/scales.time;
double t1 = 75.2/scales.time;
double t2 = 150.4/scales.time;
double t3 = 261/scales.time;
double t4 = 961/scales.time;

double m_tot_srb   = 19290/scales.mass;
double m_prop_srb  = 17010/scales.mass;
double m_dry_srb   = m_tot_srb-m_prop_srb;
double m_tot_first = 104380/scales.mass;
double m_prop_first = 95550/scales.mass;
double m_dry_first = m_tot_first-m_prop_first;
double m_tot_second = 19300/scales.mass;
double m_prop_second = 16820/scales.mass;
double m_dry_second = m_tot_second-m_prop_second;
double m_payload   = 4164/scales.mass;
double thrust_srb  = 628500/scales.force;
double thrust_first = 1083100/scales.force;
double thrust_second = 110094/scales.force;
double mdot_srb    = m_prop_srb/bt_srb;
double ISP_srb     = thrust_srb/(CONSTANTS.g0*mdot_srb);
double mdot_first  = m_prop_first/bt_first;
double ISP_first   = thrust_first/(CONSTANTS.g0*mdot_first);
double mdot_second = m_prop_second/bt_second;
double ISP_second   = thrust_second/(CONSTANTS.g0*mdot_second);

double af = 24361140/scales.length;
double ef = 0.7308;
double incf = 28.5*PI/180;
double Omf = 269.8*PI/180;
double omf = 130.5*PI/180;
double nuguess = 0;
double cosincf = cos(incf);
double cosOmf = cos(Omf);
double cosomf = cos(omf);

vec oe_matrix(6);
oe_matrix[0] = af;
oe_matrix[1] = ef;
oe_matrix[2] = incf;
oe_matrix[3] = Omf;
oe_matrix[4] = omf;
oe_matrix[5] = nuguess;
vec vout,rout;
Launchoe2rv(oe_matrix,CONSTANTS.mu,rout,vout);
```

```
double  m10 = m_payload+m_tot_second+m_tot_first+9*m_tot_srb;
double m1f = m10-(6*mdot_srb+mdot_first)*t1;
double m20 = m1f-6*m_dry_srb;
double m2f = m20-(3*mdot_srb+mdot_first)*(t2-t1);
double m30 = m2f-3*m_dry_srb;
double m3f = m30-mdot_first*(t3-t2);
double m40 = m3f-m_dry_first;
double m4f = m_payload;

CONSTANTS.thrust_srb  = thrust_srb;
CONSTANTS.thrust_first = thrust_first;
CONSTANTS.thrust_second = thrust_second;
CONSTANTS.ISP_srb     = ISP_srb;
CONSTANTS.ISP_first   = ISP_first;
CONSTANTS.ISP_second  = ISP_second;

double rmin = -2*CONSTANTS.Re;
double rmax = -rmin;
double vmin = -10000/scales.speed;
double vmax = -vmin;
//Step 1
shared_ptr<LpopcApplication> app(new LpopcApplication(console_print));

//Step2
shared_ptr<Phase> Phase1 (new Phase(1,7,3,0,1,0));
Phase1->SetTimeMin(t0,t1);
Phase1->SetTimeMax(t0,t1);
Phase1->SetStateMin(r0(0),rmin,rmin);
Phase1->SetStateMax(r0(0),rmax,rmax);

Phase1->SetStateMin(r0(1),rmin,rmin);
Phase1->SetStateMax(r0(1),rmax,rmax);

Phase1->SetStateMin(r0(2),rmin,rmin);
Phase1->SetStateMax(r0(2),rmax,rmax);

Phase1->SetStateMin(v0(0),vmin,vmin);
Phase1->SetStateMax(v0(0),vmax,vmax);

Phase1->SetStateMin(v0(1),vmin,vmin);
Phase1->SetStateMax(v0(1),vmax,vmax);

Phase1->SetStateMin(v0(2),vmin,vmin);
Phase1->SetStateMax(v0(2),vmax,vmax);

Phase1->SetStateMin(m10,m1f,m1f);
Phase1->SetStateMax(m10,m10,m10);

Phase1->SetcontrolMin(-1);
Phase1->SetcontrolMax(1);

Phase1->SetcontrolMin(-1);
Phase1->SetcontrolMax(1);

Phase1->SetcontrolMin(-1);
Phase1->SetcontrolMax(1);
//No parameter

Phase1->SetpathMin(1);
Phase1->SetpathMax(1);
```

```
Phase1->SetTimeGuess(t0);
Phase1->SetTimeGuess(t1);

Phase1->SetStateGuess(1,r0(0));
Phase1->SetStateGuess(1,r0(0));

Phase1->SetStateGuess(2,r0(1));
Phase1->SetStateGuess(2,r0(1));

Phase1->SetStateGuess(3,r0(2));
Phase1->SetStateGuess(3,r0(2));

Phase1->SetStateGuess(4,v0(0));
Phase1->SetStateGuess(4,v0(0));

Phase1->SetStateGuess(5,v0(1));
Phase1->SetStateGuess(5,v0(1));

Phase1->SetStateGuess(6,v0(2));
Phase1->SetStateGuess(6,v0(2));

Phase1->SetStateGuess(7,m10);
Phase1->SetStateGuess(7,m1f);

Phase1->SetControlGuess(1,0);
Phase1->SetControlGuess(1,0);

Phase1->SetControlGuess(2,1);
Phase1->SetControlGuess(2,1);

Phase1->SetControlGuess(3,0);
Phase1->SetControlGuess(3,0);
//Phase 2
shared_ptr<Phase> Phase2 (new Phase(2,7,3,0,1,0));
Phase2->SetTimeMin(t1,t2);
Phase2->SetTimeMax(t1,t2);

Phase2->SetStateMin(rmin,rmin,rmin);
Phase2->SetStateMax(rmax,rmax,rmax);

Phase2->SetStateMin(rmin,rmin,rmin);
Phase2->SetStateMax(rmax,rmax,rmax);

Phase2->SetStateMin(rmin,rmin,rmin);
Phase2->SetStateMax(rmax,rmax,rmax);

Phase2->SetStateMin(vmin,vmin,vmin);
Phase2->SetStateMax(vmax,vmax,vmax);

Phase2->SetStateMin(vmin,vmin,vmin);
Phase2->SetStateMax(vmax,vmax,vmax);

Phase2->SetStateMin(vmin,vmin,vmin);
Phase2->SetStateMax(vmax,vmax,vmax);

Phase2->SetStateMin(m2f,m2f,m2f);
Phase2->SetStateMax(m20,m20,m20);

Phase2->SetcontrolMin(-1);
Phase2->SetcontrolMax(1);
```

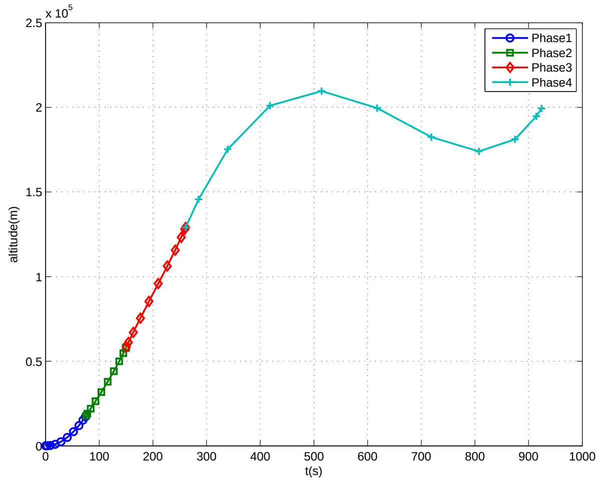
```
Phase2->SetcontrolMin(-1);
Phase2->SetcontrolMax(1);

Phase2->SetcontrolMin(-1);
Phase2->SetcontrolMax(1);
//No parameter

Phase2->SetpathMin(1);
Phase2->SetpathMax(1);

Phase2->SetTimeGuess(t1);
Phase2->SetTimeGuess(t2);

Phase2->SetStateGuess(1,r0(0));
Phase2->SetStateGuess(1,r0(0));

Phase2->SetStateGuess(2,r0(1));
Phase2->SetStateGuess(2,r0(1));

Phase2->SetStateGuess(3,r0(2));
Phase2->SetStateGuess(3,r0(2));

Phase2->SetStateGuess(4,v0(0));
Phase2->SetStateGuess(4,v0(0));

Phase2->SetStateGuess(5,v0(1));
Phase2->SetStateGuess(5,v0(1));

Phase2->SetStateGuess(6,v0(2));
Phase2->SetStateGuess(6,v0(2));

Phase2->SetStateGuess(7,m20);
Phase2->SetStateGuess(7,m2f);

Phase2->SetControlGuess(1,0);
Phase2->SetControlGuess(1,0);

Phase2->SetControlGuess(2,1);
Phase2->SetControlGuess(2,1);

Phase2->SetControlGuess(3,0);
Phase2->SetControlGuess(3,0);


shared_ptr<Phase> Phase3 (new Phase(3,7,3,0,1,0));
Phase3->SetTimeMin(t2,t3);
Phase3->SetTimeMax(t2,t3);

Phase3->SetStateMin(rmin,rmin,rmin);
Phase3->SetStateMax(rmax,rmax,rmax);

Phase3->SetStateMin(rmin,rmin,rmin);
Phase3->SetStateMax(rmax,rmax,rmax);

Phase3->SetStateMin(rmin,rmin,rmin);
Phase3->SetStateMax(rmax,rmax,rmax);

Phase3->SetStateMin(vmin,vmin,vmin);
Phase3->SetStateMax(vmax,vmax,vmax);

Phase3->SetStateMin(vmin,vmin,vmin);
```

```
Phase3->SetStateMax(vmax,vmax,vmax);

Phase3->SetStateMin(vmin,vmin,vmin);
Phase3->SetStateMax(vmax,vmax,vmax);

Phase3->SetStateMin(m3f,m3f,m3f);
Phase3->SetStateMax(m30,m30,m30);

Phase3->SetcontrolMin(-1);
Phase3->SetcontrolMax(1);

Phase3->SetcontrolMin(-1);
Phase3->SetcontrolMax(1);

Phase3->SetcontrolMin(-1);
Phase3->SetcontrolMax(1);
//No parameter

Phase3->SetpathMin(1);
Phase3->SetpathMax(1);

Phase3->SetTimeGuess(t2);
Phase3->SetTimeGuess(t3);

Phase3->SetStateGuess(1,rout(0));
Phase3->SetStateGuess(1,rout(0));

Phase3->SetStateGuess(2,rout(1));
Phase3->SetStateGuess(2,rout(1));

Phase3->SetStateGuess(3,rout(2));
Phase3->SetStateGuess(3,rout(2));

Phase3->SetStateGuess(4,vout(0));
Phase3->SetStateGuess(4,vout(0));

Phase3->SetStateGuess(5,vout(1));
Phase3->SetStateGuess(5,vout(1));

Phase3->SetStateGuess(6,vout(2));
Phase3->SetStateGuess(6,vout(2));

Phase3->SetStateGuess(7,m30);
Phase3->SetStateGuess(7,m3f);

Phase3->SetControlGuess(1,0);
Phase3->SetControlGuess(1,0);

Phase3->SetControlGuess(2,1);
Phase3->SetControlGuess(2,1);

Phase3->SetControlGuess(3,0);
Phase3->SetControlGuess(3,0);

shared_ptr<Phase> Phase4 (new Phase(4,7,3,0,1,5));
Phase4->SetTimeMin(t3,t3);
Phase4->SetTimeMax(t3,t4);

Phase4->SetStateMin(rmin,rmin,rmin);
Phase4->SetStateMax(rmax,rmax,rmax);
```

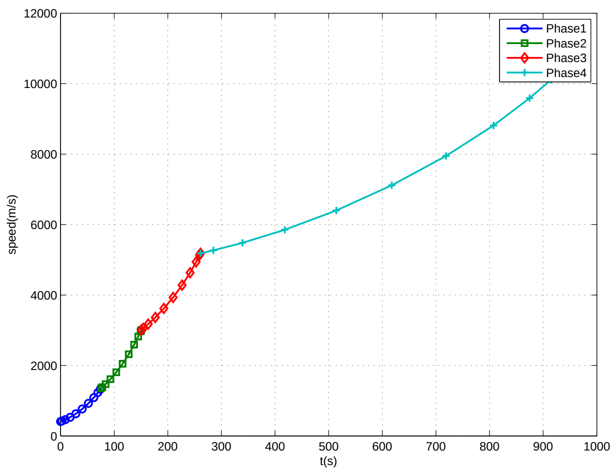
```
Phase4->SetStateMin(rmin,rmin,rmin);
Phase4->SetStateMax(rmax,rmax,rmax);

Phase4->SetStateMin(rmin,rmin,rmin);
Phase4->SetStateMax(rmax,rmax,rmax);

Phase4->SetStateMin(vmin,vmin,vmin);
Phase4->SetStateMax(vmax,vmax,vmax);

Phase4->SetStateMin(vmin,vmin,vmin);
Phase4->SetStateMax(vmax,vmax,vmax);

Phase4->SetStateMin(vmin,vmin,vmin);
Phase4->SetStateMax(vmax,vmax,vmax);

Phase4->SetStateMin(m4f,m4f,m4f);
Phase4->SetStateMax(m40,m40,m40);

Phase4->SetcontrolMin(-1);
Phase4->SetcontrolMax(1);

Phase4->SetcontrolMin(-1);
Phase4->SetcontrolMax(1);

Phase4->SetcontrolMin(-1);
Phase4->SetcontrolMax(1);
//No parameter

Phase4->SetpathMin(1);
Phase4->SetpathMax(1);

Phase4->SeteventMin(af);
Phase4->SeteventMin(ef);
Phase4->SeteventMin(incf);
Phase4->SeteventMin(Omf);
Phase4->SeteventMin(omf);

Phase4->SeteventMax(af);
Phase4->SeteventMax(ef);
Phase4->SeteventMax(incf);
Phase4->SeteventMax(Omf);
Phase4->SeteventMax(omf);

Phase4->SetTimeGuess(t3);
Phase4->SetTimeGuess(t4);

Phase4->SetStateGuess(1,rout(0));
Phase4->SetStateGuess(1,rout(0));

Phase4->SetStateGuess(2,rout(1));
Phase4->SetStateGuess(2,rout(1));

Phase4->SetStateGuess(3,rout(2));
Phase4->SetStateGuess(3,rout(2));

Phase4->SetStateGuess(4,vout(0));
Phase4->SetStateGuess(4,vout(0));

Phase4->SetStateGuess(5,vout(1));
Phase4->SetStateGuess(5,vout(1));
```

```
Phase4->SetStateGuess(6,vout(2));
Phase4->SetStateGuess(6,vout(2));

Phase4->SetStateGuess(7,m40);
Phase4->SetStateGuess(7,m4f);

Phase4->SetControlGuess(1,0);
Phase4->SetControlGuess(1,0);

Phase4->SetControlGuess(2,1);
Phase4->SetControlGuess(2,1);

Phase4->SetControlGuess(3,0);
Phase4->SetControlGuess(3,0);

shared_ptr<Linkage> Link1 (new Linkage(1,1,2));
Link1->SetLinkMin(0);
Link1->SetLinkMin(0);
Link1->SetLinkMin(0);

Link1->SetLinkMin(0);
Link1->SetLinkMin(0);
Link1->SetLinkMin(0);
Link1->SetLinkMin(-6*m_dry_srb);

Link1->SetLinkMax(0);
Link1->SetLinkMax(0);
Link1->SetLinkMax(0);

Link1->SetLinkMax(0);
Link1->SetLinkMax(0);
Link1->SetLinkMax(0);
Link1->SetLinkMax(-6*m_dry_srb);

shared_ptr<Linkage> Link2 (new Linkage(2,2,3));

Link2->SetLinkMin(0);
Link2->SetLinkMin(0);
Link2->SetLinkMin(0);

Link2->SetLinkMin(0);
Link2->SetLinkMin(0);
Link2->SetLinkMin(0);
Link2->SetLinkMin(-3*m_dry_srb);

Link2->SetLinkMax(0);
Link2->SetLinkMax(0);
Link2->SetLinkMax(0);

Link2->SetLinkMax(0);
Link2->SetLinkMax(0);
Link2->SetLinkMax(0);
Link2->SetLinkMax(-3*m_dry_srb);

shared_ptr<Linkage> Link3(new Linkage(3,3,4));
Link3->SetLinkMin(0);
Link3->SetLinkMin(0);
Link3->SetLinkMin(0);

Link3->SetLinkMin(0);
Link3->SetLinkMin(0);
```

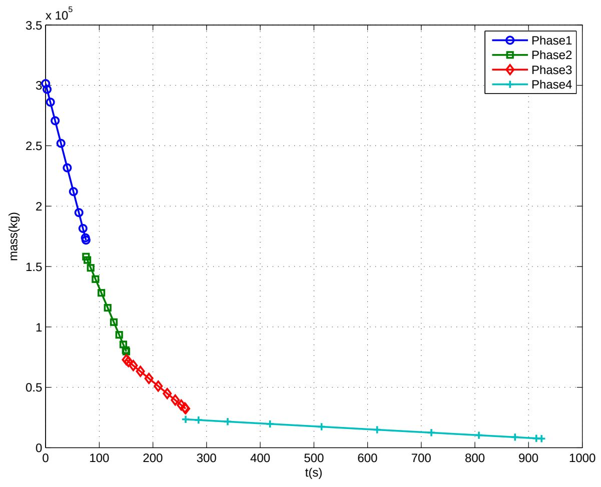
```
    Link3->SetLinkMin(0);
    Link3->SetLinkMin(-m_dry_first);

    Link3->SetLinkMax(0);
    Link3->SetLinkMax(0);
    Link3->SetLinkMax(0);

    Link3->SetLinkMax(0);
    Link3->SetLinkMax(0);
    Link3->SetLinkMax(0);
    Link3->SetLinkMax(-m_dry_first);

    //Step3
    shared_ptr<FunctionWrapper> userfun(new LaunchFunction());
    shared_ptr<OptimalProblem> optpro(new OptimalProblem(4,3,userfun));
    optpro->AddPhase(Phase1);
    optpro->AddPhase(Phase2);
    optpro->AddPhase(Phase3);
    optpro->AddPhase(Phase4);

    optpro->AddLinkage(Link1);
    optpro->AddLinkage(Link2);
    optpro->AddLinkage(Link3);


        try
        {
            //Step 4
            app->SetOptimalControlProblem(optpro);
            app->SolveOptimalProblem();
        }
        catch (LpopcException & e)
        {
            std::cout << "got␣an␣error" << std::endl;
        }
        catch (...)
        {
            std::cout << "got␣an␣error" << std::endl;
        }


    return 0;
}

void Launchoe2rv( vec& oe,double mu, vec& ri, vec& vi )
{
    assert(oe.n_elem==6);
    double a=oe(0);
    double e=oe(1);
    double i=oe(2);
    double Om=oe(3);
    double om=oe(4);
    double nu=oe(5);
    double p = a*(1-e*e);
    double r = p/(1+e*cos(nu));

    vec rv(3);
    rv[0] = r*cos(nu);
    rv[1] = r*sin(nu);
    rv[2] = 0;
    vec vv(3);
    vv[0] = -sin(nu);
    vv[1] = e + cos(nu);
    vv[2] = 0;
     vv*= sqrt(mu/p);

    double cO = cos(Om);
    double sO = sin(Om);
    double co = cos(om);
    double so = sin(om);
    double ci = cos(i);
    double si = sin(i);

    mat m_R;
    m_R << cO*co - sO*so*ci << -cO*so - sO*co*ci << sO*si << endr
        << sO*co + cO*so*ci << -sO*so + cO*co*ci << -cO*si << endr
        << so*si << co*si << ci << endr;
    /*m_R(3, 3, cO*co - sO*so*ci, sO*co + cO*so*ci, so*si,
               -cO*so-sO*co*ci,-sO*so+cO*co*ci,co*si,
               sO*si,                -cO*si,                ci);*/
    ri = m_R*rv;
    vi = m_R*vv;
}

void Launchrv2oe( vec& oe,vec rv,vec vv,double mu )
{
    vec K(3);
    K[0] = 0.0; K[1] = 0.0; K[2] = 1.0;
    vec hv=cross(rv,vv);
    vec nv=cross(K,hv);
    double n=sqrt(dot(nv,nv));
    double h2=dot(hv,hv);
    double v2=dot(vv,vv);
    double r=sqrt(dot(rv,rv));

    vec ev=rv*(v2-mu/r)-vv*dot(rv,vv);
    ev*=(1.0/mu);
    double p=h2/mu;
     double e=sqrt(dot(ev,ev));//eccentricity
     double a=p/(1-e*e);//semimajor axis
     double i = acos(hv(2)/sqrt(h2));//inclination
     double Om1=acos(nv(0)/n);//RAAN
     double eps=datum::eps;
     if (nv(1)<0-eps)
     {
         Om1=2*PI-Om1;
     }
     double Om2=acos(dot(nv,ev)/n/e);

     if (ev(2)<0)
     {
         Om2=2*PI-Om2;
     }

     double nu=acos(dot(ev,rv)/e/r);
     if (dot(rv,vv)<0)
     {
         nu=2*PI-nu;
     }
     oe(0)=a;
     oe(1)=e;
     oe(2)=i;
     oe(3)=Om1;
     oe(4)=Om2;
     oe(5)=nu;
}



void LaunchFunction::MayerCost( SolCost&mySolcost,double& mayer )
{
    mat xf=mySolcost.terminal_state_;
    if (mySolcost.phase_num_==4)
    {
        mayer=-xf(6);
    }else
    {
        mayer=0.0;
    }
}

void LaunchFunction::DerivMayer( SolCost&mySolcost,rowvec& deriv_mayer )
{
}

void LaunchFunction::LagrangeCost( SolCost& mySolcost,vec& langrange )
{
    vec t=mySolcost.time_;
    langrange=zeros(t.n_elem,1);
}

void LaunchFunction::DerivLagrange( SolCost& mySolcost,mat& deriv_langrange )
{
}

void LaunchFunction::DaeFunction( SolDae& mySolDae,mat& stateout,mat& pathout )
{
    vec t=mySolDae.time_;
    mat x=mySolDae.state_;
    mat u=mySolDae.contol_;

    int iphase=mySolDae.phase_num_;
    mat r=x.cols(0,2);
    mat v=x.cols(3,5);
    vec m=x.col(6);

    mat rad = sqrt(sum(r % r, 1));
    mat omega_matrix(CONSTANTS.omega_matrix, 3, 3);
    mat omegacrossr=r*trans(omega_matrix);
    mat vrel=v-omegacrossr;
    mat speedrel=sqrt(sum(vrel%(vrel),1));
    mat altitude=rad-CONSTANTS.Re;
    mat ret=-altitude/CONSTANTS.H;
    mat rho=exp (ret)*CONSTANTS.rho0;
    mat bc = rho / (m * 2)*(CONSTANTS.sa*CONSTANTS.cd);
    mat bcspeed=bc%(speedrel);
    mat bcspeedmat=repmat(bcspeed,1,3);
    mat Drag=bcspeedmat*(-1.0);
    Drag=Drag%vrel;//% means elem *
    mat muoverradcubed=ones(rad.n_rows,rad.n_cols)*CONSTANTS.mu;
    muoverradcubed = muoverradcubed / (pow(rad, 3));
    mat muoverradcubedMat=repmat(muoverradcubed,1,3);
    mat grav =-muoverradcubedMat %(r);
    mat T_tot,mdot;
    if (iphase==1)
    {
        mat T_srb,T_first,m1dot,m2dot;
        T_srb = ones(t.n_elem,1)*(6*CONSTANTS.thrust_srb);
        T_first =ones(t.n_elem,1)* (CONSTANTS.thrust_first);
        T_tot = T_srb+T_first;
        m1dot=zeros(T_tot.n_rows,T_tot.n_cols);
        m2dot=m1dot;
        m1dot -=T_srb/(CONSTANTS.g0*CONSTANTS.ISP_srb);
        m2dot -=T_first/(CONSTANTS.g0*CONSTANTS.ISP_first);
        mdot = m1dot+m2dot;

    }else if (iphase==2)
    {
        mat T_srb,T_first,m1dot,m2dot;
        T_srb = ones(t.n_elem,1)*(3*CONSTANTS.thrust_srb);
        T_first =ones(t.n_elem,1)* (CONSTANTS.thrust_first);
        T_tot = T_srb+T_first;
        m1dot=zeros(T_tot.n_rows,T_tot.n_cols);
        m2dot=m1dot;
        m1dot -=T_srb/(CONSTANTS.g0*CONSTANTS.ISP_srb);
        m2dot -=T_first/(CONSTANTS.g0*CONSTANTS.ISP_first);
        mdot = m1dot+m2dot;
    }else if( iphase==3){
        mat T_first;
        T_first =ones(t.n_elem,1)* CONSTANTS.thrust_first;
        T_tot = T_first;
        mdot=zeros(T_tot.n_rows,T_tot.n_cols);
        mdot -=T_first/(CONSTANTS.g0*CONSTANTS.ISP_first);
    }else if (iphase==4)
    {   vec T_second;
        T_second = ones(t.n_elem,1)*CONSTANTS.thrust_second;
        T_tot=T_second;
        mdot=zeros(T_tot.n_rows,T_tot.n_cols);
        mdot -=T_second/(CONSTANTS.g0*CONSTANTS.ISP_second);
    }

    pathout = sum(u%(u),1);
    mat Toverm = T_tot/(m);
    mat Tovermmat = repmat(Toverm,1,3);

    mat thrust = Tovermmat%(u);

    mat rdot = v;
    mat vdot = thrust+Drag+grav;
    stateout = zeros(t.n_elem, x.n_cols);// allocate memroy!!!!!
    stateout.cols(0,rdot.n_cols-1)=rdot;
    stateout.cols(rdot.n_cols,rdot.n_cols+vdot.n_cols-1)=vdot;
    stateout.col(rdot.n_cols+vdot.n_cols)=mdot;
    }

void LaunchFunction::DerivDae( SolDae& mySolDae,mat& deriv_state,mat& deriv_path )
{
}

void LaunchFunction::EventFunction( SolEvent& mySolEvent,vec& eventout )
{
    vec xf=mySolEvent.terminal_state_;
    int iphase=mySolEvent.phase_num_;
    if (iphase==4)
    {
        vec oe=zeros(6,1);
        Launchrv2oe(oe, xf.subvec(0, 2), xf.subvec(3, 5), CONSTANTS.mu);
        eventout=oe(span(0,4));
    }
}

void LaunchFunction::DerivEvent( SolEvent& mySolEvent,mat& deriv_event )
{
}

void LaunchFunction::LinkFunction( SolLink& mySolLink,vec& linkageout )
{
    vec x0_right=mySolLink.right_state_;
    vec xf_left=mySolLink.left_state_;
    linkageout=x0_right-xf_left;
}

void LaunchFunction::DerivLink( SolLink& mySolLink,mat& derive_link )
{
}
```



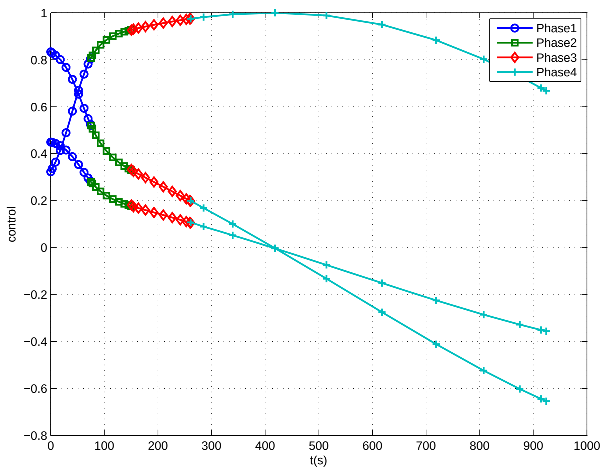
(a) Altitude vs. Time.



(b) Speed vs. Time.



(c) Mass vs. Time.



(d) Control vs. Time.

References

[1] David Benson. *A Gauss pseudospectral transcription for optimal control.* PhD thesis, Massachusetts Institute of Technology, 2005.

[2] John T. Betts. *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming.* Cambridge University Press, New York, NY, USA, 2nd edition, 2009.

[3] Arthur E BRYSON, JR, Mukund N Desai, and William C Hoffman. Energy-state approximation in performance optimization of supersonicaircraft. *Journal of Aircraft*, 6(6):481–488, 1969.

[4] Michael A Patterson, William W Hager, and Anil V Rao. A ph mesh refinement method for optimal control. *Optimal Control Applications and Methods*, 2014.

[5] Michael A Patterson and Anil Rao. Exploiting sparsity in direct collocation pseudospectral methods for solving optimal control problems. *Journal of Spacecraft and Rockets*, 49(2):354–377, 2012.

[6] Michael A. Patterson and Anil V. Rao. Gpops-ii: A matlab software for solving multiple-phase optimal control problems using hp-adaptive gaussian quadrature collocation methods and sparse nonlinear programming. *ACM Trans. Math. Softw.*, 41(1):1:1–1:37, October 2014.

[7] Anil V Rao, David A Benson, Christopher Darby, Michael A Patterson, Camila Francolin, Ilyssa Sanders, and Geoffrey T Huntington. Algorithm 902: Gpops, a matlab software for solving multiple-phase optimal control problems using the gauss pseudospectral method. *ACM Transactions on Mathematical Software (TOMS)*, 37(2):22, 2010.

[8] Anil V Rao and Kenneth D Mease. Eigenvector approximate dichotomic basis method for solving hyper-sensitive optimal control problems. *Optimal Control Applications and Methods*, 21(1):1–19, 2000.

[9] Conrad Sanderson. Armadillo: An open source c++ linear algebra library for fast prototyping and computationally intensive experiments. 2010.

[10] Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1):25–57, 2006.