POPT

Pseudospectral OPTimiser: A software for multi-phase trajectory optimisation

Preface

This document serves as the user's guide for the open-source MATLAB software *Pseudospectral OPTimiser* (*POPT*). *POPT* employs the Legendre-Gauss-Radau (LGR) collocation method to solve multiple-phase trajectory optimisation problems and has been designed to work with the NLP solver IPOPT.

Licensing Information

POPT is distributed under the MIT License. Users of the software must abide by the terms of the license.

Installation

POPT has been confirmed to be executable on MATLAB versions R2014a and beyond. The software can be used on any machine that runs Microsoft Windows 32-bit or 64-bit, Linux 64-bit or Mac OS-X 64-bit operating systems. MATLAB mex files for IPOPT for each of these operating systems is provided with the software. The installation procedure is:

1. Clone or download the source code from `github.com/InderpreetMetla/popt`.
2. Add the top-level folder to the MATLAB path or run the script `poptpathsetup.m`.

Multiple-Phase Trajectory Optimisation Problem Formulation

$\mathcal{POPT}$ is able to solve multiple-phase trajectory optimisation problems that take the following form. Let $p \in \{1, \dots, P\}$ be a single phase from a set of P phases. The aim of the problem is to determine the state, $\boldsymbol{x}^{(p)}\left(t^{(p)}\right) \in \mathbb{R}^{N_{\boldsymbol{x}}^{(p)}}$, the control, $\boldsymbol{u}^{(p)}\left(t^{(p)}\right) \in \mathbb{R}^{N_{\boldsymbol{u}}^{(p)}}$, the initial time $t_0^{(p)} \in \mathbb{R}$ and the final time $t_f^{(p)} \in \mathbb{R}$ that minimise the cost function

$$J = \sum_{p=1}^{P} \left[\phi^{(p)} \left(t_0^{(p)}, t_f^{(p)}, \boldsymbol{x}^{(p)}\left(t_0^{(p)}\right), \boldsymbol{x}^{(p)}\left(t_f^{(p)}\right) \right) + \int_{t_0^{(p)}}^{t_f^{(p)}} g^{(p)}\left(t, \boldsymbol{x}^{(p)}(t), \boldsymbol{u}^{(p)}(t)\right) dt \right] \quad (8.1a)$$

subject to

$$\dot{\boldsymbol{x}}^{(p)}(t^{(p)}) = \boldsymbol{f}^{(p)}\left(t^{(p)}, \boldsymbol{x}^{(p)}, \boldsymbol{u}^{(p)}\right) \quad \textbf{Dynamic Equality Constraints}$$

$$\boldsymbol{c}_L^{(p)} \leq \boldsymbol{c}^{(p)}\left(t^{(p)}, \boldsymbol{x}^{(p)}, \boldsymbol{u}^{(p)}\right) \leq \boldsymbol{c}_U^{(p)} \quad \textbf{Inequality Path Constraints}$$

$$\boldsymbol{b}_L^{(p)} \leq \boldsymbol{b}^{(p)}\left(t_0^{(p)}, t_f^{(p)}, \boldsymbol{x}^{(p)}, \boldsymbol{x}^{(p)}\right) \leq \boldsymbol{b}_U^{(p)} \quad \textbf{Boundary Constraints} \quad (8.1b)$$

$$\boldsymbol{\delta}_L^{(\psi)} \leq \boldsymbol{\delta}^{(\psi)}\begin{pmatrix} t_0^{(\psi+1)}, t_f^{(\psi)}, \boldsymbol{x}^{(\psi+1)}\left(t_0^{(\psi+1)}\right), \boldsymbol{x}^{(\psi)}\left(t_f^{(\psi)}\right), \dots \\ \boldsymbol{u}^{(\psi+1)}\left(t_0^{(\psi+1)}\right), \boldsymbol{u}^{(\psi)}\left(t_f^{(\psi)}\right) \end{pmatrix} \leq \boldsymbol{\delta}_U^{(\psi)} \quad \textbf{Phase Linkage Constraints}$$

Note that $N_{\boldsymbol{x}}^{(p)}$ is the number of states in the pth phase and $N_{\boldsymbol{u}}^{(p)}$ is the number of controls in the pth phase. Also note that $\psi \in \{1, \dots, P-1\}$, which restricts the phase linkages to be sequential. The functions have the following dimensions: $\phi^{(p)} \in \mathbb{R}$, $g^{(p)} \in \mathbb{R}$, $\boldsymbol{f}^{(p)} \in \mathbb{R}^{N_{\boldsymbol{x}}^{(p)}}$, $\boldsymbol{c}^{(p)} \in \mathbb{R}^{N_{\boldsymbol{c}}^{(p)}}$, $\boldsymbol{b}^{(p)} \in \mathbb{R}^{N_{\boldsymbol{b}}^{(p)}}$ and $\boldsymbol{\delta}^{(\psi)} \in \mathbb{R}^{N_{\boldsymbol{\delta}}^{(\psi)}}$, where $N_{\boldsymbol{c}}^{(p)}$ and $N_{\boldsymbol{b}}^{(p)}$ are the number of path constraints and boundary conditions in phase p, and $N_{\boldsymbol{\delta}}^{(\psi)}$ is the number of linkage constraints across link ψ.

Constructing a Problem

The call to $\mathcal{POPT}$ is deceptively simple. A problem is passed and solved by the software using the syntax

```
[solution, plotdata] = popt(problem),
```

where `problem` is a user-defined MATLAB structure that contains every piece of information about the problem to be solved. The output `solution` is a MATLAB structure that contains the solution and all information outputted by IPOPT regarding the optimisation procedure, such as CPU time and the number of iterations. The second output, `plotdata`, is a MATLAB structure that contains the solution interpolated a larger number of nodes to streamline the process of creating plots.

The `problem` struct has nine possible fields, five of which are optional. The four required fields in `problem` are:

- `problem.name`: A string with no blank spaces that provides a name for the problem.
- `problem.funcs`: A structure with five fields where each field is a user-defined function handle for the dynamics, path constraints, boundary constraints, path objective and boundary objective.
- `problem.bounds`: A structure that contains the upper and lower bounds on the states, controls, time and constraints. This structure is phase dependent.
- `problem.guess`: A structure that contains the guess for the state, control and time trajectories. This structure is phase dependent.

The five optional fields are:

- `problem.derivatives`: A structure with four fields (Table A.1 provides options and default settings).
    - `.method`: Specifies the derivative calculation scheme to be used.
    - `.order`: Specifies the derivative order to be used by IPOPT.
    - `.first`: If using finite-differencing or complex-step, this field can be used to set the step-size for the gradient and Jacobian approximations.
    - `.second`: This field can be used to set the step-size for the Hessian approximation. The Hessian is only approximated using finite-differencing.
- `problem.grid`: A structure with three fields (Table A.2 provides default settings).
    - `.tol`: A constant tolerance for the desired accuracy on every grid interval.
    - `.max_refine`: The maximum number of grid refinement iterations.

- `.phase(p)`: A structure with one field where (p) represents the *p*th phase of the problem.
  - `.nodes`: A structure with three fields:
    - `.initialgrid`: A row vector that contains the number of nodes in each of the intervals on the initial grid. The size of the vector designates the number of grid intervals in the initial grid. All intervals are equally spaced.
    - `.lb`: The lower bound on the number of LGR points in each new grid interval through the refinement process.
    - `.ub`: The upper bound on the number of LGR points in each new grid interval through the refinement process.

- `problem.auxdata`: A structure that contains any auxiliary data used by any of the functions in `problem.funcs`. This must be used to pass data between functions as it is also used by IPOPT. For each set of data, a new field is to be created.
- `problem.autoscale`: A string that specifies whether or not to use the in-built automatic scaling scheme. Options are either `'on'` or `'off'` and default is `'off'`.
- `problem.options`: A structure that allows the user the opportunity to modify IPOPT settings. Commonly modified settings are shown in Table A.3.

Table A.1 presents the options and default settings for the `problem.derivatives` structure.

Table A.1: Options and default settings for `problem.derivatives`.

Field	Options	Default
`.method`	`'fd'` or `'FD'` = forward difference `'cs'` or `'CS'` = complex-step `'ad'` or `'AD'` = auto-differentiation	`'fd'`
`.order`	`'1'` or `'2'`	`'1'`
`.first.stepsize.gradient`	A positive number	3×10^{-6}
`.first.stepsize.jacobian`	A positive number	3×10^{-8}
`.second.stepsize.hessian`	A positive number	1.75×10^{-5}

Note: $\mathcal{POPT}$ uses the open-source tool *MatlabAutoDiff* for automatic differentiation.

Table A.2 presents the default settings for the `problem.grid` structure.

Table A.2: Options and default settings for `problem.grid`.

Field	Options	Default
`.tol`	A positive number	1×10^{-7}
`.max_refine`	A positive integer	10
`.phase(p)` `.nodes.initialgrid`	Row vector of length equal to the number of desired grid intervals with each element equal to number of desired LGR points.	4*ones(1,10)
`.phase(p).nodes.lb`	A positive integer	3
`.phase(p).nodes.ub`	A positive integer	10

Table A.3 presents commonly modified settings for IPOPT.

Table A.3: Options and default settings for `problem.options`.

Field	Options	Default
`.ipopt.linear_solver`	`'mumps'` or `'ma57'`	`'mumps'`
`.ipopt.max_iter`	A positive integer	3000
`.ipopt.tol`	A positive number	1×10^{-7}

Creating the `problem.funcs` Struct

There are five fields in the `problem.funcs` structure, with each field being a user-defined function handle. All five fields are mandatory even if the trajectory optimisation problem does not require them. The fields are

- `.Dynamics`: Function handle for the system dynamics,
- `.PathObj`: Function handle for the Lagrange objective,
- `.PathCst`: Function handle for the path constraints,
- `.BndObj`: Function handle for the Mayer objective, and
- `.BndCst`: Function handle for the boundary constraints.

Note that there is no requirement for a linkage constraint function for multiple-phase problems. This is because the function is internally constructed by $\mathcal{POPT}$ automatically.

Syntax for Dynamics, Path Objective and Path Constraint Functions

The dynamics, path objective and path constraint functions are all set up in a similar manner. A separate function is required for each and the syntax to define the functions is:

```
function output = FUNCTION_NAME(t,x,u,auxdata)
```

where `FUNCTION_NAME` is replaced with whatever the user desires. The variable `t` is a column vector of length $N^{(p)}$, where $N^{(p)}$ is the number of LGR collocation points in phase p, `x` is a matrix of size $N^{(p)} \times N_x$, where N_x is the number of states for the problem and `u` is a matrix of size $N^{(p)} \times N_u^{(p)}$, where $N_u^{(p)}$ is the number of controls in phase p.

The output of these three functions, `output`, changes size depending on the function. For the dynamics function it is a matrix of size $N^{(p)} \times N_x$, whereas for the path objective function it is a vector of length $N^{(p)}$ and for the path constraints it is a matrix of size $N^{(p)} \times N_c^{(p)}$, where $N_c^{(p)}$ is the number of path constraints in phase p.

In order to distinguish between phases, a field `auxdata.iphase` is automatically created for multiple-phase problems and this can be used with conditional *if/else* statements if the dynamics, path objective or path constraints change from one phase to the next.

Note: For problems that do not have a path objective, a function must be supplied with the output set to `zeros(size(t))`. This syntax ensures that the output has correct dimensions and the derivatives are computed correctly. Similarly, if a problem does not have path constraints, a function must still be supplied with the output set to an empty array, `[]`.

Syntax for Boundary Objective and Boundary Constraint Functions

The boundary objective and boundary constraint functions are both set up in a similar manner. A separate function is required for each and the syntax to define the functions is:

```
function output = FUNCTION_NAME(t0,tf,x0,xf,auxdata)
```

Here t_0 and t_f are scalars representing the initial and terminal time, respectively, and x_0 and x_f are both a row vector of length N_x representing the initial and terminal state. The field `auxdata` is the same here as it was in Section 8.1.2.

The output of these two functions, `output`, again changes size depending on the function. For the boundary objective function, it is a scalar, whereas for the boundary constraint function it is a row vector of length $N_b^{(p)}$, where $N_b^{(p)}$ is the number of boundary constraints in phase p.

Note: For problems that do not have a boundary objective, a function must be supplied with the output set to `zeros(size(t0))`. This syntax ensures that the output has correct dimensions and the derivatives are computed correctly. Similarly, if a problem does not have boundary constraints, a function must still be supplied with the output set to an empty array, `[]`.

Creating the `problem.bounds` Struct

The constant upper and lower bounds on the variables and constraints can be defined within the `problem.bounds` structure. This struct contains two fields: `.phase(p)`, which is an array of P structs where P is the number of phases, and `.link(`ψ`)`, which is an array of $P-1$ structs.

The `.phase(p)` array defines the constant bounds on time, state, control and the path and boundary constraints. The pth element within this array of structs stores the following information:

- `problem.bounds.phase(p).initialtime.lb` $= t_0^- \in \mathbb{R}$
- `problem.bounds.phase(p).initialtime.ub` $= t_0^+ \in \mathbb{R}$
- `problem.bounds.phase(p).finaltime.lb` $= t_f^- \in \mathbb{R}$
- `problem.bounds.phase(p).finaltime.ub` $= t_f^+ \in \mathbb{R}$
- `problem.bounds.phase(p).initialstate.lb` $= \boldsymbol{x_0^-} \in \mathbb{R}^{1\times N_x^{(p)}}$
- `problem.bounds.phase(p).initialstate.ub` $= \boldsymbol{x_0^+} \in \mathbb{R}^{1\times N_x^{(p)}}$
- `problem.bounds.phase(p).state.lb` $= \boldsymbol{x}^- \in \mathbb{R}^{1\times N_x^{(p)}}$
- `problem.bounds.phase(p).state.ub` $= \boldsymbol{x}^+ \in \mathbb{R}^{1\times N_x^{(p)}}$
- `problem.bounds.phase(p).finalstate.lb` $= \boldsymbol{x_f^-} \in \mathbb{R}^{1\times N_x^{(p)}}$
- `problem.bounds.phase(p).finalstate.ub` $= \boldsymbol{x_f^+} \in \mathbb{R}^{1\times N_x^{(p)}}$
- `problem.bounds.phase(p).control.lb` $= \boldsymbol{u}^- \in \mathbb{R}^{1\times N_u^{(p)}}$
- `problem.bounds.phase(p).control.ub` $= \boldsymbol{u}^+ \in \mathbb{R}^{1\times N_u^{(p)}}$
- `problem.bounds.phase(p).path.lb` $= \boldsymbol{c}^- = \boldsymbol{c_L} \in \mathbb{R}^{1\times N_c^{(p)}}$
- `problem.bounds.phase(p).path.ub` $= \boldsymbol{c}^+ = \boldsymbol{c_U} \in \mathbb{R}^{1\times N_c^{(p)}}$
- `problem.bounds.phase(p).boundary.lb` $= \boldsymbol{b}^- = \boldsymbol{b}_L \in \mathbb{R}^{1\times N_b^{(p)}}$
- `problem.bounds.phase(p).boundary.ub` $= \boldsymbol{b}^+ = \boldsymbol{b_U} \in \mathbb{R}^{1\times N_b^{(p)}}$

The `.link(`ψ`)` array defines the constant bounds on the linkage constraints. The ψth element within this array of structs stores the following information:

- `problem.bounds.link(`ψ`).lb` $= \boldsymbol{\delta}^- = \boldsymbol{\delta}_L \in \mathbb{R}^{1\times N_\delta^{(\psi)}}$
- `problem.bounds.link(`ψ`).ub` $= \boldsymbol{\delta}^+ = \boldsymbol{\delta}_U \in \mathbb{R}^{1\times N_\delta^{(\psi)}}$

It is worth noting that if a problem does not have path or boundary constraints, or is not a multiple-phase problem, then the relevant field can be omitted or the lower and upper bounds

can be set to the empty array. Furthermore, infinite valued bounds are not permitted and should instead be replaced with a sufficiently large number.

Creating the `problem.guess` Struct

The initial guess to initialise the optimisation is provided within `problem.guess`. This struct contains the `.phase(p)` field which is used to define the guess for the time, state and control in every phase of the problem. A guess for the initial and terminal time, state and control must be provided at a minimum. The limit to the number of points used to define the initial guess is set by the number of LGR points in the initial grid. The same number of points must be used in the guess for the time, state and control. The format of the guess struct is:

- `problem.guess.phase(p).time` $= \boldsymbol{t}_{guess}^{(p)} \in \mathbb{R}^{g^{(p)} \times 1}$
- `problem.guess.phase(p).state` $= \boldsymbol{x}_{guess}^{(p)} \in \mathbb{R}^{g^{(p)} \times N_x}$
- `problem.guess.phase(p).control` $= \boldsymbol{u}_{guess}^{(p)} \in \mathbb{R}^{g^{(p)} \times N_u^{(p)}}$

where $g^{(p)}$ is the number of points used to define the guess and $g^{(p)} \geq 2$.

Problem Scaling

An in-built automatic scaling routine has been provided with $\mathcal{POPT}$. The scaling routine uses the bounds provided by the user on the variables and then scales the variables to the domain $[-1/2, 1/2]$.

Details for the Output of the Software

The two outputs from an execution of the software are the structure `solution` and the structure `plotdata`. The `solution` structure contains the following ten fields.

- `solution.phase(p)`: An array of structs of length P with three fields:
  - `.time`: The time solution at the discretisation points on the final grid iteration,
  - `.state`: The state solution at the discretisation points on the final grid iteration.
  - `.control`: The control solution at the discretisation points on the final grid iteration.
- `solution.cost`: The optimal objective value.
- `solution.bndcost`: The contribution from the Mayer objective to the total cost.
- `solution.pathcost`: The contribution from the Lagrange objective to the total cost.
- `solution.status`: Information regarding the reason IPOPT terminated.
- `solution.nlp_iters`: Number of IPOPT iterations.
- `solution.f_evals`: Number of IPOPT function evaluations.
- `solution.cpu_time`: A struct with two fields:
  - `.grids`: A vector containing IPOPT CPU time to solve the problem on each grid refinement iteration.
  - `.total`: The total CPU time for the entire optimisation procedure.
- `solution.grid_iters`: The number of grid refinement iterations.
- `solution.grid_analysis`: An array of structs that has length equal to the number of grid refinement iterations. This contains a history of the solution procedure with the optimal cost, trajectory solution, discretisation error and IPOPT information on each of the grid iterations.

The `plotdata` structure contains the same information as `solution.phase(p)`, but the results are provided at five times the number of points in order to streamline the process of creating plots.

Limitations of the Software

Whilst $\mathcal{POPT}$ can solve a broad range of trajectory optimisation problems, the following restrictions are currently present:

- Dynamics and constraint functions must have continuous first and second derivatives within a phase.
- Solutions obtained by $\mathcal{POPT}$ are only locally optimal as IPOPT is only a local solver.
- The performance of the in-built automatic scaling routine is dependent upon accurate bounds being provided by the user. If the bounds are inaccurate, the scaling routine can adversely affect the solution.
- Phases in multiple-phase problems must be connected sequentially.
- The number of states within a multiple-phase problem must remain constant across phases.
- Static parameter optimisation is not supported.
- There is no option for the user to provide analytical derivatives.
- The automatic differentiation toolbox used within $\mathcal{POPT}$ does not support MATLAB functions that use the `repmat()` function.
- The following MATLAB functions can lead to issues when using complex-step differentiation: `abs()`, `min()`, `max()` and `dot()`.
- Care must be taken when using complex-step differentiation to ensure that transposes are taken using the "dot-transpose" method in MATLAB and not the "complex-conjugate transpose".