GDOC Version 1.0 Manual

Adam B. Singer*, Benoît Chachuat, and Paul I. Barton

Department of Chemical Engineering
Massachusetts Institute of Technology
July, 2005

*Current affiliation: ExxonMobil Upstream Research Company, Houston, TX

Contents

1 Introduction

The purpose of this manual is to explain the usage of GDOC to solve dynamic optimization problems globally. GDOC is **not** in the public domain, but it is made available for free for both education and non-profit research purposes (see Appendix 5 for more details on the license terms and conditions).

GDOC-1.0 is the current version of a research code originally developed as part of Adam Singer's graduate studies at MIT. Originally, the program consisted of several loosely coupled libraries and programs with no well-defined, non-expert user interface. To use the program, one needed to possess a working knowledge of global optimization theory, a working knowledge of numerical integration, a working knowledge of unpublished results from an unwritten thesis, and an intermediate knowledge of C++. As more and more students began to require access to the numerics (probably coerced by their research advisors), the need arose to package the program in a state usable by anyone wishing to solve global dynamic optimization problems. Thus, GDOC-1.0 was born.

GDOC , short for Global Dynamic Optimization Collection, is a group of modules coupled together to solve globally the following optimization problem:

$$\min_{\mathbf{p}} J(\mathbf{p}) = \phi(\mathbf{x}(t_f, \mathbf{p}), \mathbf{p}) + \int_{t_0}^{t_f} \ell(t, \mathbf{x}(t, \mathbf{p}), \mathbf{p})\ dt$$

subject to

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{f}(t, \mathbf{x}, \mathbf{p}) \\ \mathbf{x}(t_0, \mathbf{p}) &= \mathbf{x}_0(\mathbf{p}). \end{aligned}$$

The theory is explained thoroughly in the following references [1, 2, 3, 4] and the exact implementation needs little explanation since GDOC is distributed with source code.

The two major modules contained in GDOC are the gdoc-compiler and the GDOC command line interface (CLI). In addition to this, the distribution contains a branch-and-bound code (libBandB), a numerical integrator (CVODES – with discontinuity locking extensions), and an interface to various LP and NLP solvers (libNLPSLV /libMILPSLV). It should be noted that each of these later modules is meant to be interchangeable, and some interchange may be necessary due to various licensing restrictions.

- The branch-and-bound code utilized is libBandB version 3.2 [5]. libBandB is a C++ branch-and-bound library that Adam Singer also wrote as part of his thesis work. It should be available as a stand alone component or bundled together with GDOC , provided there are no MIT license restrictions. Because libBandB has separate documentation, the branch-and-bound library will not be described here. One should refer to the relevant documentation to understand the "options.bnb" user configurable file to control the runtime branch-and-bound behavior of GDOC . The only option unique to GDOC is that one can also specify the integration tolerance with the integration tolerance keyword; its default value is 10^{-8}.

- The numerical integrator CVODES [6] is an ordinary differential equation (ODE) solver with sensitivity analysis capabilities developed at Lawrence Livermore National Labs (LLNL). The integrator is replaceable, but not nearly as easily as the other components, for the CVODES code has been extended to include a discontinuity locking wrapper and special interface to the gdoc-compiler generated code. Fortunately, CVODES is distributed under the BSD license, which enables the legal distribution of CVODES with GDOC even with these modifications as either source or binary format.

- Obtaining upper and lower bounds to the global solution at each node of the branch-and-bound tree requires the solution of linear (LP) or nonlinear (NLP) programming problems.

  For NLP problems, the code utilized is libNLPSLV version 0.2 [7]. libNLPSLV is a collection of C++ classes written by Benoît Chachuat and Cha Kun Lee that allows one to solve NLP problems based on a variety of free and commercial NLP solvers. Note that, similarly to libBandB , libNLPSLV may be used as a stand alone component. The solvers supported in the version 0.2 of libNLPSLV are: (*i*) SLSQP, a

free SQP solver written by D. Kraft [8]; (*ii*) IPOPT, a free, interior-point based, large-scale NLP solver by A. Waechter and L.T. Biegler [9]; (*iii*) NLPQL, a commercial dense SQP code by K. Schittkowski [10]; and (*iv*) SNOPT (version 5.3), another commercial SQP code for large-scale constrained problems by P.E. Gill, W. Murray, and M.A. Saunders [11]. But due to licensing restriction, only the source code for SLSQP is currently included in the distribution (see section 2 for instructions concerning the installation of the other NLP solvers).

Concerning LP problems, the code utilized is libMILPSLV version 0.9. libMILPSLV is a collection of C++ classes written by Cha Kun Lee. Note that the original libMILPSLV allows the user to solve MILP problems based either on the free MILP code LPSOLVE-5.1 [12] or on CPLEX-9.1 [13]. Due to licensing restrictions, however, libMILPSLV has been modified, and the version included in the GDOC distribution only supports LPSOLVE-5.1; besides, the libraries for LPSOLVE are included in the GDOC distribution (see section 2 below).

This implementation allows the user to switch between the supported LP and NLP solver through invocation of the GDOC CLI (see subsection 3.3 for further details); it is also meant to be flexible enough, thus making it relatively easy for the user to include his or her own NLP code (this requires C++ programming knowledge).

The remainder of this document describes usage of the GDOC CLI and the gdoc-compiler . As with the other components, these modules are replaceable. While the CLI is trivially replaceable, the compiler is responsible for generating discontinuity locked Fortran code designed specifically to interface with the extended version of CVODES and is not easily removed; it is really the heart of GDOC . Specifically, the compiler automatically applies the relaxation theory described in [1] to generate a residual evaluation code for the problem as defined in the customized GDOC input language. However, even this component is removable, if, for example, one wished to implement his or her own relaxation strategy.

2 Installation

2.1 Building the GDOC CLI and gdoc-compiler

The code was originally developed in Linux with gcc and is known to build properly with gcc 3.3.4 or later. With some modification of the makefiles, however, the code should build properly in any POSIX environment with standards compliant C++ and Fortran 77 compilers. The entire program is provided in source format and should compile with no errors by typing 'make' at the command line in the home GDOC directory. The program builds several component libraries and two executables, GDOC and the gdoc-compiler ; symbolic links to the libraries and executables are generated in the lib and bin directories, respectively. In order to execute the GDOC CLI, both the GDOC lib path and the current path must be in the library load path. In linux, this is most easily accomplished by exporting these libraries in the LD_LIBRARY_PATH environment variable.

As discussed earlier in the introduction section, GDOC CLI allows the user to switch between different LP and NLP solvers:

- Only the source code for SLSQP (slsqp.f) is included in the GDOC-1.0 distribution.

- The required libraries for LPSOLVE-5.1 are also included in the src/libMILPSLV directory of the distribution (LPSOLVE-5.1 is released under the GNU Lesser Public License (LGPL) — see http://www.opensource.org/licenses/lgpl-license.php). This includes the shared libraries liblpsolve51.so and libxli_CPLEX.so, as well as of the header files lp_Hash.h, lp_lib.h, lp_matrix.h, lp_mipbb.h, lp_SOS.h, lp_types.h and lp_utils.h. Note that all these files can be obtained, for free, directly from the website http://groups.yahoo.com/group/lp_solve.

- The static library for IPOPT is not included in the distribution due to licensing restrictions. Instead, a dummy fortran 77 file (ipopt_dum.f) is provided, therefore enabling proper compilation of the GDOC CLI. However, the user is strongly encouraged to download and install IPOPT on his or her own computer, and then recompile GDOC with the actual IPOPT static library; this is easily done, *e.g.*, by replacing:

  ```
  ipopt = -L./lib -lipopt_dum
  ```

  by:

  ```
  ipopt = -L/[IPOPT_dir]/lib -lipopt
  ```

  in the makefile of the GDOC-1.0 directory, where [IPOPT_dir] is the directory where IPOPT was installed. Besides, the latest IPOPT distribution can be retreived from http://www.coin-or.org/Ipopt/.

- Concerning NLPQL and SNOPT, dummy fortran 77 files (nlpql_dum.f and snoptf_dum.f) are also provided in the src/libNLPSLV directory of the GDOC-1.0 distribution in order to enable proper compilation of the GDOC CLI. Should the user have valid licenses for NLPQL and/or SNOPT-5.3, these dummy files can be easily substituted with the actual distributions, after minor modifications of the makefiles.

2.2 Building an individual problem

In addition to the core GDOC libraries, GDOC requires a dynamic library called '**libres.so**' that contains the residual evaluation for both the upper and lower bounding problems. For typical usage, the user will write their problem in the GDOC language and use the gdoc-compiler to generate automatically Fortran 77 source code for the residual evaluations. This Fortran file can then be compiled as the shared library libres.so to be linked at runtime to the GDOC CLI. Because of the late binding of the library, libres.so need not be present when GDOC is built.

3 Using GDOC

Using GDOC is easily divided into three distinct tasks: (*i*) writing a GDOC model, (*ii*) using the gdoc-compiler to generate the associated residual routine library, and (*iii*) using the GDOC CLI to obtain the desired results (optimization, simulation, or multi-start analysis). Each topic is now individually discussed in detail.

3.1 Writing a GDOC model

The GDOC language is a flexible language designed to allow a user without expert knowledge of optimization to perform global optimization provided he or she is capable of expressing the problem in a mathematically abstract manner. In particular, the GDOC language allows the user to specify either an integral or algebraic objective function (or mixed), constraining ordinary differential equations, and various other components necessary to completely specify the problem. The symbols +, -, * and / denote summation, substraction, multiplication and division, respectively, while ^ stands for the power operator; the keywords EXP and LN denote the exponential and natural logarithmic functions, respectively.

The language is divided into several required and optional sections (the full language grammar is defined by the yacc file in the source code); the usage of each section is independently described below. A section begins with a keyword (or words), and each section is terminated by the keyword END. The required sections must be written in the order in which they are described below. The optional sections are order independent, but any optional section must follow the required sections. A comment consists of any text following a # until a newline is encountered. Because the target language (Fortran 77) is case insensitive, GDOC models are also case insensitive.

3.1.1 The Declaration Section [required]

The Declaration Section is initialized by the `DECLARATION` keyword. In the Declaration Section, the user specifies information about the size of the problem. The `gdoc-compiler` uses this information to perform basic consistency checks on the model and to size the Fortran variables in the output. Each line in the Declaration Section consists of a keyword, a colon, and an argument. The required keywords are `state`, `parameter`, and `time`. The `state` keyword is used to declare the state variable and its size. The syntax is given as

```
state: x(1:3)
```

In the above example, the variable `x` (the user may select the name of the state variable) represents the state, and we have declared the state to be a vector composed from `x(1)`, `x(2)`, and `x(3)`. The special size `(0:0)` indicates that the state is a scalar; scalars are used throughout the model without a subscript. The optimization parameter for the problem is specified identically to the state variable with the use of the keyword `parameter` in lieu of the keyword `state`.

In `GDOC` , the independent variable is referred to as time, and the independent variable must appear as `t` in a model. However, conceptually, the independent variable need not represent the physical quantity of time. Essentially, the `time` keyword is utilized to specify the domain of integration. The syntax for specifying time is

```
time:[0,10]
```

Here, the independent variable has the range of `0` to `10`. Time is dimensionless and therefore "inherits" the units implied by the differential equations.

The optional keywords for the Declaration Section are `intermediate`, `constant` and `data`.

- The `intermediate` keyword is used to denote an intermediate variable for the specification of the differential equations; the expressions of the intermediate variables are then given in the `EQUATION` section (see example in subsection 3.1.3 below). The declaration of an intermediate variable takes the same form as the declaration of a state variable or a parameter.

- The `constant` keyword is used to denote a constant parameter for the problem; the values for the constants are then declared in the `CONSTANT VALUES` section. The declaration of a constant takes the same form as the declaration of a state variable or a parameter.

- The `data` keyword is used to declare that numerical data is utilized in the current problem (*e.g.*, time series data for parameter estimation problems). The data is passed to `GDOC` via a data file at the command line. A data file is an ASCII file consisting of multiple rows of data. The first column is always the independent variable and subsequent columns are the data; data is comma delimited, and rows are separated by newlines. The data declaration follows the following syntax:

  ```
  data:xdata(1:2)
  ```

  Here, we have chosen to name the data `xdata` (data may not be named `data`, for this is a reserved keyword), and the data may be used in the problem as `xdata(1)` and `xdata(2)`. In this case, the data file would possess three columns. As previously stated, the first column represents the independent variable, the second column represents `xdata(1)`, and the third column represents `xdata(2)`.

The following syntax represents a complete Declaration Section (indentation is not required and is present simply for readability):

```
DECLARATION
  state: x(1:5)
  parameter: p(0:0)
```

```
  time: [0, 4.46]
  intermediate: z(1:2)
  constant: k(1:5)
  data: xdata(1:1)
END
```

3.1.2 Parameter Values Section [required]

Parameter initial values and parameter bounds are declared in a section denoted by the keyword `PARAMETER VALUES`. Suppose we have a problem with three parameters. The following syntax represents a typical Parameter Values Section:

```
PARAMETER VALUES
  p(1) = 528.92  : [10.0, 1200.0]
  p(2) = 401.69  : [10.0, 1000.0]
  p(3) =  28.63  : [0.001, 40.0]
END
```

In the above sample section, each parameter is defined separately. The first number following the equals sign is the initial guess for the parameter at the root node. Following the colon is the set defining the lower and upper bounds for the parameter. Each parameter must have an entry in this section.

3.1.3 Equation Section [required]

The Equation Section (initialized by the keyword `EQUATION`) of the `GDOC` model is where the user defines the differential equations defining the embedded dynamic model. In the notation of the introduction, this section defines the equation $\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x}, \mathbf{p})$. Each equation in this section is an explicit ODE. The left hand side of each equation must be a state variable preceded by the `$` symbol, where this symbol represents the derivative of the state variable with respect to the independent variable. The right hand side of each equation is an algebraic expression representing the function f_i (for the ith ODE). Equations are terminated by a `;`, and the number of equations in this section must equal the number of state variables. As an example, suppose we have the following system of differential equations:

$$\begin{aligned} \dot{x}_1 &= -p_1 x_1 \\ \dot{x}_2 &= p_1 x_1 - p_2 x_2 \\ \dot{x}_3 &= p_2 x_2 \end{aligned}$$

Then, the following represents the Equation Section for the above system:

```
EQUATION
  $x(1) = -p(1) * x(1);
  $x(2) = p(1) * x(1) - p(2) * x(2);
  $x(3) = p(2) * x(2);
END
```

or, equivalently, with the intermediate variables `z`:

```
EQUATION
  $x(1) = -z(1);
  $x(2) = z(1) - z(2);
  $x(3) = z(2);
  z(1)  = p(1) * x(1);
  z(2)  = p(2) * x(2);
END
```

3.1.4 Reference Section [required]

The Reference Section (delimited by the keyword `REFERENCE`) is the section in which the reference trajectory for the problem is specified. The reference trajectory is the trajectory along which the right hand sides of the differential equations are linearized in the convex ODE relaxation theory. To fully understand the meaning of the reference trajectory, the interested reader is referred to [1, 2, 3]. The format for the Reference Section is similar to the format for the Equation Section. An example of a Reference Section is

```
REFERENCE
  x(1) = xL(1);
  x(2) = xL(2);
  p(1) = pU(1);
  p(2) = 0.5*(pL(2)+pU(2));
END
```

The Reference Section must contain an equation for each state and an equation for each parameter. The user should note that appending a `U` (or `_U`) to a state or parameter represents the upper bound while appending an `L` (or `_L`) to a state or parameter represents the lower bound. To ensure that a reference trajectory is valid in a given node, a good reference trajectory is usually some function of the bounds such that the equation evaluates to some number interior to the bound set. In the absence of knowledge indicating a good reference trajectory, the user should simply select one of the bounds.

3.1.5 Initial Values Section [required]

The Initial Values Section (delimited by the keyword `INITIAL`) is the section in which the initial condition for the ODE is specified. In the context of the Introduction, the Initial Values Section is where the equation $\mathbf{x}(t_0, \mathbf{p}) = \mathbf{x}_0(\mathbf{p})$ is defined. In order to be a well determined ODE, the number of equations in the Initial Values Section must equal the number of state variables. The user should note that parameters may appear in the right hand side of an equation; the left hand side of any given equation must be a single state variable. The following is an example of a valid Initial Values Section for an ODE with two state variables:

```
INITIAL
  x(1) = 3.0;
  x(2) = p(1);
END
```

3.1.6 Objective Section [optional]

The Objective Section (delimited by the keyword `OBJECTIVE`) is the section in which the non-integral part of the objective function is defined. In the notation of the Introduction, this section defines $\phi(\mathbf{x}(t_f, \mathbf{p}), \mathbf{p})$. In the absence of a data file, the objective function is always defined at the final time point on the domain of integration (except optimal control problems). In the presence of a data file, the default behavior of the Objective Section is to represent a summation of the function evaluated at each time point specified in the data file (optimal control problems default to a summation of the Objective Section at the time mesh points). To evaluate any objective function only at the final time, the default behavior can be overridden by specifying `-f` at the command line.

As an example, suppose for a parameter estimation problem that we have data spaced over ten equal intervals in the domain [0, 1] as specified by having ten rows in our data file. For an objective function defined as

$$\phi(\mathbf{p}) = \sum_{i=1}^{10} (x - \hat{x}_i)^2$$

the objective function would be written with the following `GDOC` code:

```
OBJECTIVE
  (x-xdata)^2;
END
```

Again, if one wished to only evaluate the objective at the final time point, the `-f` command could be given to `GDOC` at the command line to override the default summation.

3.1.7 Integrand Section [optional]

The Integrand Section (delimited by the keyword `INTEGRAND`) is the section in which the integral portion of the objective function is defined. In the notation of the Introduction, the Integrand Section defines $\ell(t, \mathbf{x}(t, \mathbf{p}), \mathbf{p})$, where the integration is on the independent variable t; the limits of the integral are defined by the domain of integration specified in the Declaration Section. Suppose our problem possessed the following integral for its objective function:

$$L(\mathbf{p}) = \int_{t_0}^{t_f} (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 \; dt.$$

The Integrand Section would be specified as

```
INTEGRAND
  (x(1)^2+x(2)-11)^2+(x(1)+x(2)^2-7)^2;
END
```

The user should note that the Integrand Section and the Objective Section are not mutually exclusive. For mixed (Bolza) type objective functions, both sections may be specified concurrently. In this case, the objective function is given as the sum of the integral and non-integral portions. That is, $J = \phi + L$.

3.1.8 Natural Bounds Section [optional]

The Natural Bounds Section (delimited by the keyword `NATURAL BOUNDS`) is utilized to indicate the natural bounds on the states as known *a priori* from physical considerations. For example, consider a state variable representing a species concentration. If the differential equation defining the state were a decay relationship, from physical insight, we would know that the concentration in the system would be bounded between 0 and the initial concentration for all time. Further details concerning the natural bounds for a system can be found in [1, 2, 3].

The following is a sample of a natural bounds section:

```
NATURAL BOUNDS
  x(1):[0,1]
  x(3):[0,--]
  x(4):[--,1]
END
```

In addition to syntax, the above example demonstrates several important principles concerning the natural bounds. First, the index gap in the state variable specification is intentional. Natural bounds may not be known for all state variables; therefore, only states for which bounds are known should be specified. Second, possibly only one bound is known for a particular state; in such a case, the symbol `--` is used to indicate a free bound. Also note that `GDOC-1.0` does not allow specification of parameter dependent mathematical expressions for the natural bounds, nor does it allow specification of complex invariants such as $x_1^2 + x_2^2 = K$. This capability will be added in a future release of `GDOC` .

3.1.9 Constant Values Section [optional]

The Constant Values Section (delimited by the keyword `CONSTANT VALUES`) enables the user to specify values for the constants instantiated in the Declaration Section. The section is listed as optional because constants are not required for a well-posed `GDOC` model. However, if constants are declared, then this section must be included, and the number of constants must equal the size of the constant vector declared in the Declaration Section. The following is an example of a properly specified Constant Values Section:

```
CONSTANT VALUES
  k(1) = 50.0;
  k(2) = 1200.0;
  k(3) = 1.0E-5;
END
```

3.1.10 Plot Section [optional]

The Plot Section (delimited by the keyword `PLOT`) enables the user to generate data for visualizing a defined function (and the state variables) versus time. The operating mode (simulation mode) for generating this data is specified by the `-s` command line argument to `GDOC` . When `GDOC` is run in simulation mode, rather than executing a global optimization, a simulation is performed with the parameter values fixed at the values given in the input file. The time mesh for a simulation is given by the mesh defined in the input data file. If no input data is required for the current problem, then a dummy input file can be constructed giving only time values (each line for a dummy file must then be terminated by a ',' and a newline). The results of a simulation run are displayed on standard out (the `-o [file]` option may be given to instead specify an output file) and consist of space separated columns of data. The first column is always time, the next n_d columns are the n_d data columns read from the data file, the next column is the function defined in the Plot Section, and the last n_x columns are the n_x state variables. The output is suitable for plotting programs such as gnuplot or Matlab. The following is an example of a properly specified Plot Section:

```
PLOT
  2100.0*x(1)+200.0*(x(4)+x(5));
END
```

3.2 Using the `gdoc-compiler`

The `gdoc-compiler` is the part of `GDOC` responsible for translating the `GDOC` input file, applying the dynamic relaxation theory, and generating a Fortran residual file for the numerical integrator. The compiler runs in batch mode (suitable for use in a makefile), and the different compiler options are controlled by command line arguments. The `gdoc-compiler` is called with the following syntax:

```
gdoc-compiler [-hubd] [-o output] input
```

Options

`--help (-h)` prints the usage to standard out.

`--upper (-u)` bypasses generation of the convex relaxation for the problem; only the residual routines necessary for simulating the upper bound are produced.

`--bounds (-b)` implements nonsmooth bounds cutting; if a Natural Bounds Section is included in the input file, the default behavior is to implement bounds tightening only via interval arithmetic. By specifying the nonsmooth bounds cutting option, the compiler generates code that ensures that the state bounds do not exceed the natural bounds in excess of some predetermined ε.

`--disclock (-d)` indicates that discontinuity locking should be employed for the residual evaluation of the convex relaxation. Additionally, information is generated to automatically inform the included numerical integrator that the lower bound is discontinuity locked. Unless the user understands the precise mathematical structure of the lower bounding problem, it is recommended to always use discontinuity locking. Without discontinuity locking, the residual of the convex relaxation may not be Lipschitz continuous, and the numerical integrator may fail during the computation of the lower bound.

`-o output` stands for the output file option and must be followed by the name of an output file. If this option is not present, the output of the compilation is sent to standard out.

The compiler shipped with this distribution is version 2.0 as opposed to version 1.0 as discussed in [1] (note that `gdoc-compiler` 's version differs from `GDOC` 's version because the `gdoc-compiler` actually existed before `GDOC`). Version 2.0 was written for two reasons. First, despite writing it himself, Adam did not like the design and rewrote almost the entire code simply to improve design and maintainability. Second, the original compiler was limited by the use of symbolic manipulation of equations (for derivatives and convex relaxations). Version 2.0 employs algorithmic differentiation concepts to compute both derivatives and convex relaxations. The utilization of AD means that the version 2.0 compiler generates more compact and more efficient residual files than the version 1.0 compiler. In fact, if one desired, he or she could even compute a theoretical temporal upper bound for the residual calculation of the relaxations; no one has yet done so.

3.3 Using the `GDOC` CLI

The `GDOC` command line interface (CLI) is a batch interface for simulating and globally optimizing dynamic optimization problems. At run time, `GDOC` must dynamically bind to a shared library called `libres.so` that contains the residual routines generated by the `gdoc-compiler` . Consult your compiler documentation to learn how to create shared libraries (the makefile in the examples directory demonstrates how to build a shared library with gcc).

Assuming the existence of `libres.so`, `GDOC` is executed using the following syntax:

```
GDOC [-hlsedfN] [-L solver] [-U solver] [-n partitions] [-r samples] [-o output] [input]
```

Options

`--help (-h)` prints the usage to standard out.

`--lnscaling (-l)` automatically implements scaling of the parameters by the natural logarithm.

`--simulation (-s)` runs `GDOC` in simulation mode instead of performing optimization; simulation mode was previously discussed in Section 3.1.10.

`--empty (-e)` indicates that the current problem does not require a data input file. For problems that do not specify `--empty (-e)`, an input file **must** be given. Input files are described in Section 3.1.1.

`--disable (-d)` disables exploitation of the affine structure for computing the lower bound. By construction, the relaxation generated by `gdoc-compiler` is affine at any fixed time. For problems only requiring computation of the objective at fixed time points (e.g., sum squared error parameter estimation problems), `GDOC` utilizes this information to optimize the computation of the convex relaxation by only calling the numerical integrator once per branch-and-bound node and computing the relaxation by linear algebra on subsequent calls from the optimizer. For problems with integral objective functions, however, the affine structure optimization cannot be exploited and therefore must be

disabled (the affine structure optimization can be disabled for any problem, but the lower bounds will be computed significantly slower).

`--final (-f)` tells `GDOC` to only compute the objective function at the final time point rather than as the sum of the objective at each distinct time interval (as defined either by the data file or by the piecewise optimal control mesh).

`--nonsmooth (-N)` tells `GDOC` to calculate the lower bounds by solving a (potentially) nonsmooth optimization problem. Solving nonsmooth lower problems generally results in faster computations, but may also produce unexpected results as most NLP solvers require that the participating functions be (at least) once continuously differentiable. If the `--nonsmooth (-N)` is not specified, then `GDOC` performs a smooth reformulation of the nonsmooth lower problem by introducing additional constraints and decision variables

`-L solver` specifies to `GDOC` which solver shall be used for the lower problems. Currently supported solvers are the LP solver `LPSOLVE-5.1` (default solver), as well as the NLP solvers `SLSQP`, `IPOPT`, `NLPQL` and `SNOPT-5.3` — More details about the supported solvers can be found in the Introduction Section. Note that the specification of the `-L LPSOLVE` option together with the `--nonsmooth (-N)` option results in a run time error.

`-U solver` specifies to `GDOC` which solver shall be used for the upper problems. Currently supported solvers are the NLP solvers `SLSQP`, `IPOPT`, `NLPQL` and `SNOPT-5.3`. More details about the supported solvers can be found in the Introduction Section.

`-n partitions` specifies to `GDOC` that the current problem is an optimal control problem with a piecewise constant control profile; the argument of the `-n` option is the number of equally spaced time intervals requested. When solving optimal control problems, `GDOC` assumes the control variable is always at the end of the parameter array defined in the `GDOC` input file (only one control variable can be handled in `GDOC-1.0`).

`-r samples` runs `GDOC` in multi-start analysis mode instead of performing global optimization; the argument of the `-r` option is the number of samples (random points) requested. A local optimization problem is solved for every randomly generated starting point.

`-o output` is used for simulation mode to specify an output file for the simulation run; output of a simulation defaults to standard out.

4 Executing the Examples

`GDOC` comes with an extensive set of test problems to illustrate usage for several different types of problems. Typing `make` at the command line will print `GDOC` 's usage to standard out and then provide a list of example problems that can be built. Note that all the test problems reported in the papers [1, 3] are included in the `examples` directory. However, the computational results given in these papers had been obtained with the NLP solver `NPSOL` by [14], which is no longer supported in `GDOC-1.0`. Also note that for several test problems (including all least-square parameter estimation problems), the relaxed objective function is known to be at least once continuously differentiable. Accordingly, it is recommended to solve these problems using the `--nonsmooth (-N)` option (see subsection 3.3), which generally allows faster computations.

Besides, the `examples` directory is an excellent resource for understanding how to build your own example problems. The `input` subdirectory contains all the `GDOC` input files for the example problems, and the `data` subdirectory contains data files necessary for executing the examples. The `makefile` in the `examples` directory contains commands for compiling `GDOC` input files into Fortran files and then subsequently compiling the Fortran files into `libres.so` shared libraries.

For example, to build the $A \rightarrow B \rightarrow C$ estimation problem, type '`make AtoBtoC`' at the command line. This causes the `AtoBtoC.oai` file in the `input` directory to be compiled into a `libres.so` shared library. After compilation, `make` returns instructions for executing the example. For this example problem, type '`GDOC data/AtoBtoC.inp`' at the command line and press enter.

The user should note that the provided `makefile` is appropriate for a linux system using `gcc` as the compiler; consult your local documentation for other systems or compilers. Also note that running the examples requires that the `GDOC` library path and the current path are in the run time loader's library search path. On a linux system, one can ensure the loader can find the appropriate libraries by setting the `LD_LIBRARY_PATH` environment variable to the appropriate directories. For example, on a linux system using `bash`, type

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:.:[GDOC_base]/lib
```

at the command line, where `[GDOC_base]` is the directory where `GDOC` was installed.

5 Epilogue

`GDOC` , the `gdoc-compiler` , and `libBandB` were all programs originally designed and written by Adam B. Singer as part of his thesis work at MIT. The authors acknowledge that both the design and implementation are far from perfect; however, the source code for these programs is provided so that other researchers may understand the exact algorithms employed and use these tools however they see fit. In fact, we openly encourage other researchers to extend and improve upon this code base in their own endeavors. Questions and comments may be sent to Benoît Chachuat at `bchachua@mit.edu` or Adam B. Singer at `absinger@alum.mit.edu`. Due to contractual obligations and intellectual property issues, however, Adam is not currently permitted to either distribute `GDOC` or contribute new code to `GDOC` . He is, however, happy to answer questions and discuss the design, implementation, or theory behind any of the components in `GDOC` .

References

[1] A. B. Singer. *Global Dynamic Optimization.* PhD thesis, Massachusetts Institute of Technology, 2004.

[2] A. B. Singer and P. I. Barton. Bounding the solutions of parameter dependent nonlinear ordinary differential equations. In press: *SIAM Journal on Scientific Computing*, 2004.

[3] A. B. Singer and P. I. Barton. Global optimization with nonlinear ordinary differential equations. In press: *Journal of Global Optimization*, 2004.

[4] A. B. Singer and P. I. Barton. Global solution of linear dynamic embedded optimization problems. *Journal of Optimization Theory and Applications*, 121(3):613–646, 2004.

[5] A. B. Singer. `LibBandB` version 3.2 manual. Technical report, Massachusetts Institute of Technology, January 2004.

[6] A. C. Hindmarsh and R. Serban. User documentation for `CVODES`, an ODE solver with sensitivity analysis capabilities. Technical report, Lawrence Livermore National Laboratory, July 2002.

[7] B. Chachuat and C. K. Lee. `LibNLPSLV` version 0.2 manual. Technical report, Massachusetts Institute of Technology, July 2005.
(`doxygen` documentation).

[8] D. Kraft. A software package for sequential quadratic programming. Technical report, DFVLR Oberpfaffenhofen, 1988.
(available from `http://www.netlib.org`).

[9] A. Waechter and L.T. Biegler. On the implementation of a interior-point filter-line search algorithm for large-scale nonlinear programming. Technical Report RC23149, IBM T.J. Watson Research Center, Yorktown, NY, USA.
(available from `http://www.coin-or.org/Ipopt/`).

[10] K. Schittkowski. `NLPQL`: A fortran subroutine solving constrained nonlinear programming problems. *Annals of Operations Research*, 5:485–500, 1985/1986.
(see `http://www.uni-bayreuth.de/departments/math/~kschittkowski/nlpql.htm`).

[11] P.E. Gill, W. Murray, and M.A. Saunders. `SNOPT`: An SQP algorithm for large-scale constrained optimization. *SIAM Journal on Optimization*, 12:979–1006, 2002.
(see `http://www.sbsi-sol-optimize.com/asp/sol_product_snopt.htm`).

[12] M. Berkelaar, K. Eikland, and P. Notebaert. Introduction to `lp_solve 5.1.1.3`, May 2004.
(available at `http://www.geocities.com/lpsolve/`).

[13] ILOG Cplex 9.1. User's manual and reference manual.
(see `http://www.ilog.com/`).

[14] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. User's guide for `NPSOL 5.0`: A fortran package for nonlinearm programming. Technical report, Stanford University, July 1998.

GDOC-1.0 License Terms and Conditions

GDOC-1.0 is **not** in the public domain. However, it is made available, for free, for both education and non-profit research purposes. Any entity desiring permission to incorporate this software or a work based on the software into commercial products or otherwise use it for commercial purposes should contact:

Paul I. Barton
Professor of Chemical Engineering
Phone: 1.617.253.6526
Fax: 1.617.258.5042
Email: pib@mit.edu
Web: http://web.mit.edu/cheme/people/faculty/barton.html

1. The "Software", below, refers to GDOC-1.0 (in either source-code, object-code or executable-code form) which includes the GDOC CLI and the gdoc-compiler , as well as the third party components libBandB, libNLPSLV and libMILPSLV. Each licensee is addressed as "you" or "Licensee."

2. The Massachusetts Institute of Technology is the copyright holder of the Software. The copyright holders reserve all rights except those expressly granted to the Licensee herein.

3. Licensee shall use the Software solely for education and non-profit research purposes within Licensee's organization. Permission to copy the Software for use within Licensee's organization is hereby granted to Licensee, provided that the copyright notice and this license accompany all such copies. Licensee shall not permit the Software, or source code generated using the Software, to be used by persons outside Licensee's organization or for the benefit of third parties. Licensee shall not have the right to relicense or sell the Software or to transfer or assign the Software or source code generated using the Software.

4. If you modify a copy or copies of the Software or any portion of it, thus forming a work based on the Software, and make and/or distribute copies of such work within your organization, you must meet the following conditions:

   a) You must cause the modified Software to carry prominent notices stating that you changed specified portions of the Software.

   b) If you make a copy of the Software (modified or verbatim) for use within your organization it must include the copyright notice and this license.

5. Neither the Massachusetts Institute of Technology nor any of its employees makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed and covered by a license granted under this license agreement, or represents that its use would not infringe privately owned rights.

6. In no event will the Massachusetts Institute of Technology be liable for any damages, including direct, incidental, special, or consequential damages resulting from exercise of this license agreement or the use of the licensed software.

Any use of this code constitutes acceptance of the terms of the copyright notice and license agreement.