Appendix

Software

All of the algorithms described in the book have been implemented and tested on realistic test problems. In addition, all of the examples presented have been solved using this software, and FORTRAN implementations of the examples can be obtained by contacting the author. The $\mathbb{SOCS}$ [38] library, which contains all of the software, has a great deal of functionality and it is not intended that this appendix should be viewed as a user's manual. On the other hand, it is worthwhile to outline the basic capability of the tool to assist the reader in the formulation and solution of practical problems. To that end, the following sections present a brief overview of the library. Additional information can be obtained by contacting the author at `john.betts@comcast.net`.

A.1 Simplified Usage Dense NLP

The subroutine HDNLPD provides the most basic functionality for solving small, dense NLP problems. The primary information that must be supplied by the user (outlined with a double box in Figure A.1) is a subroutine called FUNBOX that evaluates the objective and constraint functions. This is the *function generator* described in Section 3.8. All algorithm parameters are given default values, which are appropriate for a simple problem. If desired, the default values can be redefined using a utility routine HHSNLP that is common to all software in the $\mathbb{SOCS}$ library. The simplified usage software uses a *forward-communication* format and, as such, the optimization algorithm HDNLPD calls the user-supplied subroutine, which has a fixed calling argument list. For more sophisticated applications, the *reverse-communication* subroutine HDNLPR can be used. HDNLPR is appropriate for small, dense applications when the user can supply gradient and Hessian information. It is also appropriate for use with complicated simulation programs and when parallel processing is used.

A.2 Sparse NLP with Sparse Finite Differences

Large, sparse NLP problems necessarily demand more information from the user because of the need to specify matrix sparsity and provide corresponding derivative information.

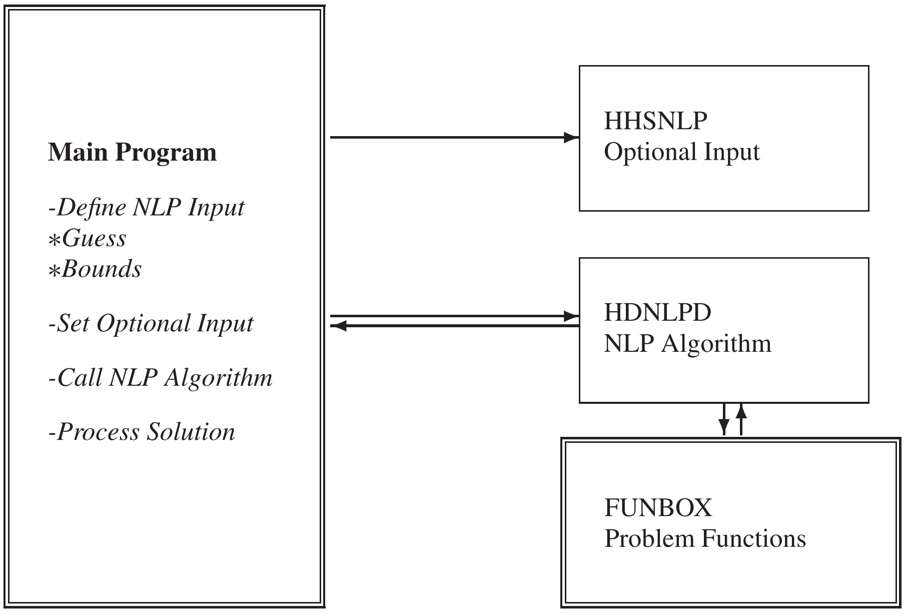

Figure A.1. *Dense NLP software.*

Figure A.2 illustrates the usage of the sparse NLP algorithm HDSNLP, which employs a reverse-communication format. Again, the user-supplied software is shown inside a double box. In addition, software for constructing sparse finite difference first and second derivatives HDSFDJ and HDSFDH is also available. A utility procedure (HJSFDI) can be used for constructing sparse difference index sets based on the user-supplied matrix sparsity patterns. Optional input can be set using the standard $\mathbb{SOCS}$ utility HHSNLP. Similar functionality is available when solving large, sparse nonlinear least squares problems using subroutine HDSLSQ rather than HDSNLP.

A.3 Optimal Control Using Sparse NLP

Implementing the solution of an optimal control problem using the direct transcription method in $\mathbb{SOCS}$ requires some software supplied by the user. Figure A.3 illustrates the software organization of an application. As before, user-supplied procedures are shown with double boxes. However, many of the user-supplied routines are optional, as indicated by an asterisk in the illustration. The user must call the $\mathbb{SOCS}$ algorithm HDSOCS. The user must define the problem via the subroutine ODEINP. All other information is optional and can be supplied either by the user or by using dummy routines from the $\mathbb{SOCS}$ library instead. Typically, the user will specify the right-hand sides of the DAEs and quadrature functions using subroutine ODERHS. For applications with nonlinear boundary conditions, the point function routine ODEPTF must be supplied. If special output (e.g., graphics files) is desired, the user can supply a special-purpose ODEPRT routine. The default initial guess

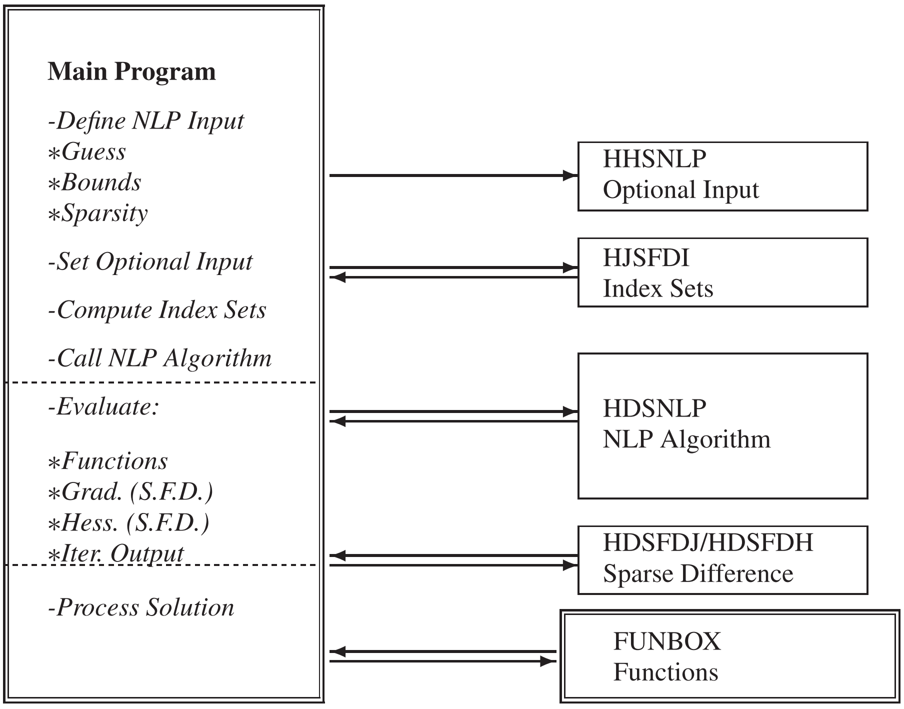

Figure A.2. *Sparse NLP software.*

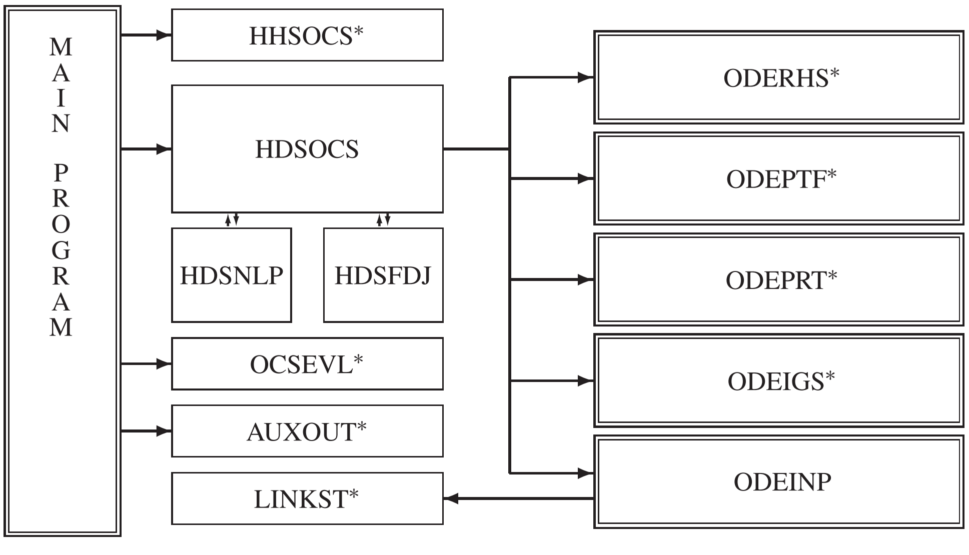

Figure A.3. *Sparse optimal control software.*

for the state and control variables is a linear function between the phase boundaries. When other initialization procedures are used, subroutine ODEIGS provides this functionality. When solving an optimal control problem, *it is not necessary* to define the matrix sparsity, to compute derivatives, or to call the sparse NLP algorithm. However, nonstandard NLP options can be set using the HHSNLP utility. Nonstandard optimal control algorithm options can be set using the HHSOCS utility. For example, the discretization accuracy and number of mesh-refinement iterations can be set by HHSOCS. A feasible (but suboptimal) trajectory can be computed by using the feasibility (F) option as set by HHSNLP. The solution computed by $\mathbb{SOCS}$ is represented using B-spline function(s). The solution can be evaluated at arbitrary points with the utility OCSEVL and/or the auxiliary output procedure AUXOUT. Multiphase formulations may also incorporate the LINKST utility when linking phases together. The overall $\mathbb{SOCS}$ library has been designed with flexibility and functionality in mind and is especially suited for use with complex simulation systems.