C interfaces to GALAHAD

Jari Fowkes and Nick Gould
STFC Rutherford Appleton Laboratory
Mon Feb 21 2022

Chapter 1

GALAHAD C packages

1.1 Introduction

GALAHAD is foremost a modern fortran library of packages designed to solve continuous optimization problems, with a particular emphasis on those that involve a large number of unknowns. Since many application programs or applications are written in other languages, of late there has been a considerable effort to provide interfaces to GALAHAD. Thus there are Matlab interfaces, and here we provide details of those to C using the standardized ISO C support now provided within fortran.

1.1.1 Main authors

N. I. M. Gould, STFC-Rutherford Appleton Laboratory, England,
D. Orban, Ecole Polytechnique de Montreal, Canada,
D. P. Robinson, Leheigh University, USA, and
Ph. L. Toint, The University of Namur, Belgium.

C interfaces, additionally J. Fowkes, STFC-Rutherford Appleton Laboratory.

1.2 Scope

GALAHAD provides packages as named for the following problems:

- fdc - determine consistency and redundancy of linear systems (link)
- lpa - linear programming using an active-set method (link)
- lpb - linear programming using an interior-point method (link)
- wcp - linear feasibility using an interor-point method (link)
- bqp - bound-constrained convex quadratic programming using a gradient-projection method
- bqpb - bound-constrained convex quadratic programming using an interor-point method (link)
- lsqp - linear and seprable quadratic programming using an interor-point method (link)
- cqp - convex quadratic programming using an interor-point method (link)

- dqp - convex quadratic programming using a dual active-set method (link)
- eqp - equality-constrained quadratic programming using an iterative method (link)
- trs - the trust-region subproblem using matrix factorization (link)
- gltr - the trust-region subproblem using matrix-vector products (link)
- rqs - the regularized quadratic subproblem using matrix factorization (link)
- glrt - the regularized quadratic subproblem using matrix-vector products (link)
- dps - the trust-region and regularized quadratic subproblems in a diagonalising norm (link)
- lstr - the least-squares trust-region subproblem using matrix-vector products (link)
- lsrt - the regularized least-squares subproblem using matrix-vector products (link)
- l2rt - the regularized linear l_2 norm subproblem using matrix-vector products (link)
- qpa - general quadratic programming using an active-set method (link)
- qpb - general quadratic programming using an interor-point method (link)
- blls - bound-constrained linear-least-squares using a gradient-projection method
- bllsb - bound-constrained linear-least-squares using an interior-point method (in preparation)
- tru - unconstrained optimization using a trust-region method (link)
- arc - unconstrained optimization using a regularization method (link)
- nls - least-squares optimization using a regularization method (link)
- trb - bound-constrained optimization using a gradient-projection trust-region method (link)
- nlsb - bound-constrained least-squares optimization using a gradient-projection regularization method (in preparation)
- lancelot - general constrained optimization using an augmented Lagrangian method (interface in preparation)
- fisqp - general constrained optimization using an SQP method (in preparation)

In addition, there are packages for solving a variety of required sub tasks, and most specifically interface routines to external solvers for solving linear equations:

- uls - unsymetric linear systems (link)
- sls - symetric linear systems (link)
- sbls - symetric block linear systems (link)
- psls - preconditioners for symetric linear systems (link)

C interfaces to all of these are underway, and each will be released once it is ready. If **you** have a particular need, please let us know, and we will raise its priority!

1.3 Further topics

1.3.1 Unsymmetric matrix storage formats

An unsymmetric m by n matrix A may be presented and stored in a variety of convenient input formats.

Both C-style (0 based) and fortran-style (1-based) indexing is allowed. Choose `control.f_indexing` as `false` for C style and `true` for fortran style; the discussion below presumes C style, but add 1 to indices for the corresponding fortran version.

Wrappers will automatically convert between 0-based (C) and 1-based (fortran) array indexing, so may be used transparently from C. This conversion involves both time and memory overheads that may be avoided by supplying data that is already stored using 1-based indexing.

1.3.1.1 Dense storage format

The matrix A is stored as a compact dense matrix by rows, that is, the values of the entries of each row in turn are stored in order within an appropriate real one-dimensional array. In this case, component $n * i + j$ of the storage array A_val will hold the value A_{ij} for $0 \leq i \leq m-1$, $0 \leq j \leq n-1$.

1.3.1.2 Dense storage format

The matrix A is stored as a compact dense matrix by columns, that is, the values of the entries of each column in turn are stored in order within an appropriate real one-dimensional array. In this case, component $m * j + i$ of the storage array A_val will hold the value A_{ij} for $0 \leq i \leq m-1$, $0 \leq j \leq n-1$.

1.3.1.3 Sparse co-ordinate storage format

Only the nonzero entries of the matrices are stored. For the l-th entry, $0 \leq l \leq ne-1$, of A, its row index i, column index j and value A_{ij}, $0 \leq i \leq m-1$, $0 \leq j \leq n-1$, are stored as the l-th components of the integer arrays A_row and A_col and real array A_val, respectively, while the number of nonzeros is recorded as A_ne = ne.

1.3.1.4 Sparse row-wise storage format

Again only the nonzero entries are stored, but this time they are ordered so that those in row i appear directly before those in row i+1. For the i-th row of A the i-th component of the integer array A_ptr holds the position of the first entry in this row, while A_ptr(m) holds the total number of entries plus one. The column indices j, $0 \leq j \leq n-1$, and values A_{ij} of the nonzero entries in the i-th row are stored in components l = A_ptr(i), $\ldots$, A_ptr(i+1)-1, $0 \leq i \leq m-1$, of the integer array A_col, and real array A_val, respectively. For sparse matrices, this scheme almost always requires less storage than its predecessor.

1.3.1.5 Sparse column-wise storage format

Once again only the nonzero entries are stored, but this time they are ordered so that those in column j appear directly before those in column j+1. For the j-th column of A the j-th component of the integer array A_ptr holds the position of the first entry in this column, while A_ptr(n) holds the total number of entries plus one. The row indices i, $0 \leq i \leq m-1$, and values A_{ij} of the nonzero entries in the j-th columnsare stored in components l = A_ptr(j), $\ldots$, A_ptr(j+1)-1, $0 \leq j \leq n-1$, of the integer array A_row, and real array A_val, respectively. As before, for sparse matrices, this scheme almost always requires less storage than the co-ordinate format.

1.3.2 Symmetric matrix storage formats

Likewise, a symmetric n by n matrix H may be presented and stored in a variety of formats. But crucially symmetry is exploited by only storing values from the lower triangular part (i.e, those entries that lie on or below the leading diagonal).

1.3.2.1 Dense storage format

The matrix H is stored as a compact dense matrix by rows, that is, the values of the entries of each row in turn are stored in order within an appropriate real one-dimensional array. Since H is symmetric, only the lower triangular part (that is the part H_{ij} for $0 \leq j \leq i \leq n-1$) need be held. In this case the lower triangle should be stored by rows, that is component $i * i/2 + j$ of the storage array H_val will hold the value H_{ij} (and, by symmetry, h_{ji}) for $0 \leq j \leq i \leq n-1$.

1.3.2.2 Sparse co-ordinate storage format

Only the nonzero entries of the matrices are stored. For the l-th entry, $0 \leq l \leq ne-1$, of H, its row index i, column index j and value h_{ij}, $0 \leq j \leq i \leq n-1$, are stored as the l-th components of the integer arrays H_row and H_col and real array H_val, respectively, while the number of nonzeros is recorded as H_ne = ne. Note that only the entries in the lower triangle should be stored.

1.3.2.3 Sparse row-wise storage format

Again only the nonzero entries are stored, but this time they are ordered so that those in row i appear directly before those in row i+1. For the i-th row of H the i-th component of the integer array H_ptr holds the position of the first entry in this row, while H_ptr(n) holds the total number of entries plus one. The column indices j, $0 \leq j \leq i$, and values H_{ij} of the entries in the i-th row are stored in components l = H_ptr(i), . . ., H_ptr(i+1)-1 of the integer array H_col, and real array H_val, respectively. Note that as before only the entries in the lower triangle should be stored. For sparse matrices, this scheme almost always requires less storage than its predecessor.

1.3.2.4 Diagonal storage format

If H is diagonal (i.e., $h_{ij} = 0$ for all $0 \leq i \neq j \leq n-1$) only the diagonals entries h_{ii}, $0 \leq i \leq n-1$ need be stored, and the first n components of the array H_val may be used for the purpose.

1.3.2.5 Multiples of the identity storage format

If H is a multiple of the identity matrix, (i.e., $H = \alpha I$ where I is the n by n identity matrix and α is a scalar), it suffices to store α as the first component of H_val.

1.3.2.6 The identity matrix format

If H is the identity matrix, no values need be stored.

1.3.2.7 The zero matrix format

The same is true if H is the zero matrix.

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

File Documentation

3.1 galahad.h File Reference

Index