GALAHAD DPS

USER DOCUMENTATION GALAHAD Optimization Library version 5.0

1 SUMMARY

Given a real n by n symmetric matrix $\mathbf{H}$, this package **construct a symmetric, positive definite matrix M so that H is diagonal in the norm $\|\mathbf{v}\|_{\mathbf{M}} = \sqrt{\mathbf{v}^T\mathbf{M}\mathbf{v}}$ induced by M**. Subsequently the package can be use to **solve the trust-region subproblem**

$$\text{minimize } q(\mathbf{x}) = \tfrac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x} + \mathbf{c}^T\mathbf{x} + f \text{ subject to } \|\mathbf{x}\|_{\mathbf{M}} \leq \Delta \tag{1.1}$$

or the **regularized quadratic problem**

$$\text{minimize } q(\mathbf{x}) + \tfrac{1}{p}\sigma\|\mathbf{x}\|_{\mathbf{M}}^p \tag{1.2}$$

for a real n vector $\mathbf{c}$ and scalars f, $\Delta > 0$, $\sigma > 0$ and $p \geq 2$.

A factorization of the matrix $\mathbf{H}$ will be required, so this package is most suited for the case where such a factorization, either dense or sparse, may be found efficiently.

ATTRIBUTES — Versions: `GALAHAD_DPS_single`, `GALAHAD_DPS_double`. **Uses:** `GALAHAD_CLOCK`, `GALAHAD_SYMBOLS`, `GALAHAD_SPACE`, `GALAHAD_NORMS`, `GALAHAD_SMT`, `GALAHAD_SPECFILE`, `GALAHAD_SLS`, `GALAHAD_TRS`, `GALAHAD_RQS` **Date:** March 2018. **Origin:** N. I. M. Gould. **Language:** Fortran 2003. **Parallelism:** Some options may use OpenMP and its runtime library.

2 HOW TO USE THE PACKAGE

The package is available using both single and double precision reals, and either 32-bit or 64-bit integers. Access to the 32-bit integer, single precision version requires the `USE` statement

```
USE GALAHAD_DPS_single
```

with the obvious substitution `GALAHAD_DPS_double`, `GALAHAD_DPS_single_64` and `GALAHAD_DPS_double_64` for the other variants.

If it is required to use more than one of the modules at the same time, the derived types `SMT_TYPE`, `DPS_control_type`, `DPS_history_type`, `DPS_inform_type`, `DPS_data_type`, (Section 2.4) and the subroutines `DPS_initialize`, `DPS_solve`, `DPS_terminate` (Section 2.5) and `DPS_read_specfile` (Section 2.7) must be renamed on one of the `USE` statements.

2.1 Matrix storage formats

The matrix $\mathbf{H}$ may be stored in a variety of input formats.

2.1.1 Dense storage format

The matrix $\mathbf{H}$ 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 $\mathbf{H}$ is symmetric, only the lower triangular part (that is the part h_{ij} for $1 \leq j \leq i \leq n$) need be held. In this case the lower triangle should be stored by rows, that is component $i*(i-1)/2+j$ of the storage array `H%val` will hold the value h_{ij} (and, by symmetry, h_{ji}) for $1 \leq j \leq i \leq n$.

2.1.2 Sparse co-ordinate storage format

Only the nonzero entries of the matrices are stored. For the l-th entry of $\mathbf{H}$, $1 \leq j \leq i \leq n$, its row index i, column index j and value h_{ij} are stored in the l-th components of the integer arrays `H%row`, `H%col` and real array `H%val`, respectively. Note that only the entries in the lower triangle should be stored.

2.1.3 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 $\mathbf{H}$, the i-th component of the integer array `A%ptr` holds the position of the first entry in this row, while `A%ptr` $(m+1)$ holds the total number of entries plus one. The column indices j, $1 \leq j \leq i$, 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.

2.2 Real and integer kinds

We use the terms integer and real to refer to the fortran keywords `REAL(rp_)` and `INTEGER(ip_)`, where `rp_` and `ip_` are the relevant kind values for the real and integer types employed by the particular module in use. The former are equivalent to default `REAL` for the single precision versions and `DOUBLE PRECISION` for the double precision cases, and correspond to `rp_ = real32` and `rp_ = real64`, respectively, as supplied by the fortran `iso_fortran_env` module. The latter are default (32-bit) and long (64-bit) integers, and correspond to `ip_ = int32` and `ip_ = int64`, respectively, again from the `iso_fortran_env` module.

2.3 Parallel usage

OpenMP may be used by the `GALAHAD_DPS` package to provide parallelism for some solvers in shared memory environments. See the documentation for the GALAHAD package `SLS` for more details. To run in parallel, OpenMP must be enabled at compilation time by using the correct compiler flag (usually some variant of `-openmp`). The number of threads may be controlled at runtime by setting the environment variable `OMP_NUM_THREADS`.

MPI may also be used by the package to provide parallelism for some solvers in a distributed memory environment. To use this form of parallelism, MPI must be enabled at runtime by using the correct compiler flag (usually some variant of `-lmpi`). Although the MPI process will be started automatically when required, it should be stopped by the calling program once no further use of this form of parallelism is needed. Typically, this will be via statements of the form

```
CALL MPI_INITIALIZED( flag, ierr )
IF ( flag ) CALL MPI_FINALIZE( ierr )
```

The code may be compiled and run in serial mode.

2.4 The derived data types

Five derived data types are accessible from the package.

2.4.1 The derived data type for holding matrices

The derived data type `SMT_TYPE` is used to hold the matrices $\mathbf{H}$ and perhaps $\mathbf{M}$ and/or $\mathbf{A}$. The components of `SMT_TYPE` used here are:

`m` is a scalar component of type `INTEGER(ip_)`, that holds the number of rows in the matrix.

n is a scalar component of type INTEGER(ip_), that holds the number of columns in the matrix.

ne is a scalar variable of type INTEGER(ip_), that holds the number of matrix entries.

type is a rank-one allocatable array of type default CHARACTER, that is used to indicate the matrix storage scheme used. Its precise length and content depends on the type of matrix to be stored.

val is a rank-one allocatable array of type REAL(rp_) and dimension at least ne, that holds the values of the entries. Each pair of off-diagonal entries $h_{ij} = h_{ji}$ of the *symmetric* matrix **H** is represented as a single entry (see §2.1.1–2.1.3). Any duplicated entries that appear in the sparse co-ordinate or row-wise schemes will be summed.

row is a rank-one allocatable array of type INTEGER(ip_), and dimension at least ne, that may hold the row indices of the entries. (see §2.1.2).

col is a rank-one allocatable array of type INTEGER(ip_), and dimension at least ne, that may hold the column indices of the entries (see §2.1.2–2.1.3).

ptr is a rank-one allocatable array of type INTEGER(ip_), and dimension at least n + 1, that may holds the pointers to the first entry in each row (see §2.1.3).

2.4.2 The derived data type for holding control parameters

The derived data type DPS_control_type is used to hold controlling data. Default values may be obtained by calling DPS_initialize (see Section 2.5.1). The components of DPS_control_type are:

error is a scalar variable of type INTEGER(ip_), that holds the stream number for error messages. Printing of error messages in DPS_solve and DPS_terminate is suppressed if error ≤ 0. The default is error = 6.

out is a scalar variable of type INTEGER(ip_), that holds the stream number for informational messages. Printing of informational messages in DPS_solve is suppressed if out < 0. The default is out = 6.

print_level is a scalar variable of type INTEGER(ip_), that is used to control the amount of informational output which is required. No informational output will occur if print_level ≤ 0. If print_level = 1 a single line of output will be produced for each iteration of the process. If print_level ≥ 2 this output will be increased to provide significant detail of each iteration. The default is print_level = 0.

new_h is a scalar variable of type INTEGER(ip_), that is used to indicate how **H** has changed (if at all) since the previous call to DPS_solve. Possible values are:

- 0 **H** is unchanged.
- 1 the values in **H** have changed, but its nonzero structure is as before.
- 2 both the values and structure of **H** have changed.

The default is new_h = 2.

taylor_max_degree is a scalar variable of type INTEGER(ip_), that specifies the maximum degree of Taylor approximant that will be used to approximate the secular function when trying to improve λ; a first-degree approximant results in Newton's method. The higher the degree, the better in general the improvement, but the larger the cost. Thus there is a balance between many cheap low-degree approximants and a few more expensive higher-degree ones. Our experience favours higher-degree approximants. The default is taylor_max_degree = 3, which is the highest degree currently supported.

eigen_min is a scalar variables of type REAL(rp_), that specifies the smallest allowable value of an eigenvalue of the block diagonal factor of **H**. Any eigenvalue smaller than eigen_min will be set to eigen_min when constructing **M**. See Section 4 for more details. The default is $\sqrt{u}$, where u is EPSILON(1.0) (EPSILON(1.0D0) in GALAHAD_DPS_double).

lower is a scalar variables of type REAL(rp_), that holds the value of any known lower bound on the required multiplier λ_*. A good lower bound may sometimes dramatically improve the performance of the package, but an incorrect value might cause the method to fail. Thus resetting lower from its default should be used with caution. The default is lower = - HUGE(1.0) (-HUGE(1.0D0) in GALAHAD_DPS_double).

upper is a scalar variables of type REAL(rp_), that holds the value of any known upper bound on the required multiplier λ_*. A good upper bound may sometimes dramatically improve the performance of the package, but an incorrect value might cause the method to fail. Thus resetting upper from its default should be used with caution. The default is upper = HUGE(1.0) (HUGE(1.0D0) in GALAHAD_DPS_double).

stop_normal and stop_absolute_normal are scalar variables of type REAL(rp_), that hold values for the standard convergence tolerances of the method (see Section 4). In particular, the method is deemed to have converged for the trust-region subproblem when the computed solution $\mathbf{x}$ and its multiplier λ satisfy either $\lambda = 0$ and $\|\mathbf{x}\|_{\mathbf{M}} < \Delta$ or $\|\mathbf{x}\|_{\mathbf{M}} - \Delta| \leq \max($ stop_normal $* \Delta$, stop_absolute_normal$)$, while convergence in the regularization case happens when $\|\mathbf{x}\|_{\mathbf{M}} - (\lambda/\sigma)^{1/(p-2)} \leq$ stop_normal $* \max(1, \|\mathbf{x}\|_{\mathbf{M}}, (\lambda/\sigma)^{1/(p-2)})$. The defaults are stop_normal = stop_absolute_normal = $u^{0.75}$, where u is EPSILON(1.0) (EPSILON(1.0D0) in GALAHAD_DPS_double).

goldfarb is a scalar variable of type default LOGICAL, that may be set .TRUE. if the user wishes the package to use Goldfarb's method to build $\mathbf{M}$ from $\mathbf{H}$, and .FALSE. if the modified-absolute-value method is to be used instead. The default is goldfarb = .FALSE..

space_critical is a scalar variable of type default LOGICAL, that may be set .TRUE. if the user wishes the package to allocate as little internal storage as possible, and .FALSE. otherwise. The package may be more efficient if space_critical is set .FALSE.. The default is space_critical = .FALSE..

deallocate_error_fatal is a scalar variable of type default LOGICAL, that may be set .TRUE. if the user wishes the package to return to the user in the unlikely event that an internal array deallocation fails, and .FALSE. if the package should be allowed to try to continue. The default is deallocate_error_fatal = .FALSE..

symmetric_linear_solver is a scalar variable of type default CHARACTER and length 30, that specifies the external package to be used to solve any symmetric linear system that might arise. Current possible choices are 'sils', 'ma27', 'ma57', 'ma77', 'ma86', 'ma97', ssids, 'pardiso' and 'wsmp', although only 'sils' and, for OMP 4.0-compliant compilers, 'ssids' are installed by default. See the documentation for the GALAHAD package SLS for further details. The default is symmetric_linear_solver = 'sils'.

prefix is a scalar variable of type default CHARACTER and length 30, that may be used to provide a user-selected character string to preface every line of printed output. Specifically, each line of output will be prefaced by the string prefix(2:LEN(TRIM(prefix))-1), thus ignoring the first and last non-null components of the supplied string. If the user does not want to preface lines by such a string, they may use the default prefix = "".

SLS_control is a scalar variable of type SLS_control_type that is used to control various aspects of the factorization package SLS. See the documentation for GALAHAD_SLS for more details.

2.4.3 The derived data type for holding timing information

The derived data type DPS_time_type is used to hold elapsed CPU and system clock times for the various parts of the calculation. The components of DPS_time_type are:

total is a scalar variable of type REAL(rp_), that gives the total CPU time spent in the package.

analyse is a scalar variable of type REAL(rp_), that gives the CPU time spent analysing required matrices prior to factorization.

factorize is a scalar variable of type REAL(rp_), that gives the CPU time spent factorizing the required matrices.

solve is a scalar variable of type REAL(rp_), that gives the CPU time spent solving diagonal trust-region or regularization subproblems.

clock_total is a scalar variable of type REAL(rp_), that gives the total elapsed system clock time spent in the package.

clock_analyse is a scalar variable of type REAL(rp_), that gives the elapsed system clock time spent analysing required matrices prior to factorization.

clock_factorize is a scalar variable of type REAL(rp_), that gives the elapsed system clock time spent factorizing the required matrices.

clock_solve is a scalar variable of type REAL(rp_), that gives the elapsed time spent solving diagonal trust-region or regularization subproblems.

2.4.4 The derived data type for holding informational parameters

The derived data type DPS_inform_type is used to hold parameters that give information about the progress and needs of the algorithm. The components of DPS_inform_type are:

status is a scalar variable of type INTEGER(ip_), that gives the current status of the algorithm. See Section 2.6 for details.

alloc_status is a scalar variable of type INTEGER(ip_), that gives the status of the last internal array allocation or deallocation. This will be 0 if status = 0.

bad_alloc is a scalar variable of type default CHARACTER and length 80, that gives the name of the last internal array for which there were allocation or deallocation errors. This will be the null string if status = 0.

mod_1by1 is a scalar variable of type INTEGER(ip_), that gives the number of eigenvalues from 1 by 1 blocks from the factorization of $\mathbf{H}$ that were modified when constructing $\mathbf{M}$.

mod_2by2 is a scalar variable of type INTEGER(ip_), that gives the number of eigenvalues from 2 by 2 blocks from the factorization of $\mathbf{H}$ that were modified when constructing $\mathbf{M}$.

obj is a scalar variable of type REAL(rp_), that holds the value of the objective function $\frac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x}+\mathbf{c}^T\mathbf{x}+f$.

obj_regularized is a scalar variable of type REAL(rp_), that holds the value of the regularized objective function $\frac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x}+\mathbf{c}^T\mathbf{x}+f+\frac{1}{p}\sigma\|\mathbf{x}\|_{\mathbf{M}}^p$.

multiplier is a scalar variable of type REAL(rp_), that holds the value of the Lagrange multiplier λ associated with the constraint.

x_norm is a scalar variable of type REAL(rp_), that holds the value of $\|\mathbf{x}\|_{\mathbf{M}}$.

pole is a scalar variable of type REAL(rp_), that holds a lower bound on $\max(0,-\lambda_1)$, where λ_1 is the left-most eigenvalue of the matrix pencil (H,M).

hard_case is a scalar variable of type default LOGICAL, that will be .TRUE. if the "hard-case" has occurred (see Section 4) and .FALSE. otherwise.

time is a scalar variable of type DPS_time_type whose components are used to hold elapsed CPU and system clock times for the various parts of the calculation (see Section 2.4.3).

SLS_inform is a scalar variable of type SLS_inform_type, that holds informational parameters concerning the analysis, factorization and solution phases performed by the GALAHAD sparse matrix factorization package SLS. See the documentation for the package SLS for details of the derived type SLS_inform_type.

2.4.5 The derived data type for holding problem data

The derived data type DPS_data_type is used to hold all the data for a particular problem between calls of DPS procedures. This data should be preserved, untouched, from the initial call to DPS_initialize to the final call to DPS_terminate.

2.5 Argument lists and calling sequences

There are three procedures for user calls (see Section 2.7 for further features):

1. The subroutine DPS_initialize is used to set default values and initialize private data.
2. The subroutine DPS_solve is called to compute the desired matrix $\mathbf{M}$ and then solve the appropriate quadratic subproblem.
3. The subroutine DPS_resolve is used to resolve the problem when the only data that has changed since the last solution are the values of the radius Δ or the regularization weight σ and, optionally, the linear and constant terms $\mathbf{c}$ and f.
4. The subroutine DPS_terminate is provided to allow the user to automatically deallocate array components of the private data, allocated by DPS_solve, at the end of the solution process.

We use square brackets [ ] to indicate OPTIONAL arguments.

2.5.1 The initialization subroutine

Default values are provided as follows:

```
CALL DPS_initialize( data, control, inform )
```

data is a scalar INTENT(INOUT) argument of type DPS_data_type (see Section 2.4.5). It is used to hold data about the problem being solved.

control is a scalar INTENT(OUT) argument of type DPS_control_type (see Section 2.4.2). On exit, control contains default values for the components as described in Section 2.4.2. These values should only be changed after calling DPS_initialize.

inform is a scalar INTENT(OUT) argument of type DPS_inform_type (see Section 2.4.4). A successful call to DPS_initialize is indicated when the component status has the value 0. For other return values of status, see Section 2.6.

2.5.2 The optimization problem solution subroutine

The optimization problem solution algorithm is called as follows:

```
CALL DPS_solve( n, H, c, f, X, data, control, inform[, delta, sigma, p] )
```

n is a scalar INTENT(IN) argument of type INTEGER(ip_), that must be set to the number of unknowns, n. **Restriction:** $n > 0$.

H is scalar INTENT(IN) argument of type SMT_TYPE that holds the Hessian matrix $\mathbf{H}$. The following components are used here:

H%type is an allocatable array of rank one and type default CHARACTER, that is used to indicate the storage scheme used. If the dense storage scheme (see Section 2.1.1) is used, the first five components of H%type must contain the string DENSE. For the sparse co-ordinate scheme (see Section 2.1.2), the first ten components of H%type must contain the string COORDINATE, and for the sparse row-wise storage scheme (see Section 2.1.3), the first fourteen components of H%type must contain the string SPARSE_BY_ROWS.

For convenience, the procedure SMT_put may be used to allocate sufficient space and insert the required keyword into H%type. For example, if we wish to store **M** using the co-ordinate scheme, we may simply

```
CALL SMT_put( H%type, 'COORDINATE' )
```

See the documentation for the GALAHAD package SMT for further details on the use of SMT_put.

H%ne is a scalar variable of type INTEGER(ip_), that holds the number of entries in the **lower triangular** part of **H** in the sparse co-ordinate storage scheme (see Section 2.1.2). It need not be set for any of the other three schemes.

H%val is a rank-one allocatable array of type REAL(rp_), that holds the values of the entries of the **lower triangular** part of the Hessian matrix **H** in any of the storage schemes discussed in Section 2.1.

H%row is a rank-one allocatable array of type INTEGER(ip_), that holds the row indices of the **lower triangular** part of **H** in the sparse co-ordinate storage scheme (see Section 2.1.2). It need not be allocated for any of the other schemes.

H%col is a rank-one allocatable array variable of type INTEGER(ip_), that holds the column indices of the **lower triangular** part of **H** in either the sparse co-ordinate (see Section 2.1.2), or the sparse row-wise (see Section 2.1.3) storage scheme. It need not be allocated when the dense storage scheme is used.

H%ptr is a rank-one allocatable array of dimension n+1 and type INTEGER(ip_), that holds the starting position of each row of the **lower triangular** part of **H**, as well as the total number of entries plus one, in the sparse row-wise storage scheme (see Section 2.1.3). It need not be allocated when the other schemes are used.

C is an array INTENT(IN) argument of dimension n and type REAL(rp_), whose i-th entry holds the component c_i of the vector **c** for the objective function.

f is a scalar INTENT(IN) variable of type REAL(rp_), that holds the scalar value f for the objective function.

X is an array INTENT(OUT) argument of dimension n and type REAL(rp_), that holds an estimate of the solution **x** of the problem (1.1) or (1.2) on exit.

data is a scalar INTENT(INOUT) argument of type DPS_data_type (see Section 2.4.5). It is used to hold data about the problem being solved. It must not have been altered **by the user** since the last call to DPS_initialize.

control is a scalar INTENT(IN) argument of type DPS_control_type. (see Section 2.4.2). Default values may be assigned by calling DPS_initialize prior to the first call to DPS_solve.

inform is a scalar INTENT(INOUT) argument of type DPS_inform_type (see Section 2.4.4) whose components need not be set on entry. A successful call to DPS_solve is indicated when the component status has the value 0. For other return values of status, see Section 2.6.

delta is an OPTIONAL scalar INTENT(IN) variable of type default REAL(rp_), that must be set on initial entry to the value of the radius of the trust-region constraint, Δ if the solution to the trust-region subproblem (1.1) is required. If delta is not PRESENT, the trust-region subproblem will not be solved, but when it is PRESENT, the regularization subproblem will not be solved regardless of the status of sigma (see below). **Restriction:** delta $>$ 0.

sigma is an OPTIONAL scalar INTENT(IN) variable of type default REAL(rp_), that must be set on initial entry to the value of the regularization weight, σ if the solution to the regularization subproblem (1.2) is required. If sigma is not PRESENT, the regularization subproblem will not be solved. **Restriction:** sigma $>$ 0.

p is an OPTIONAL scalar INTENT(IN) variable of type default REAL(rp_), that must be set on initial entry to the value of the regularization order, p if the solution to the regularization subproblem (1.2) is required. If p is not PRESENT, the regularization order is taken to be 3.0. **Restriction:** p $\geq$ 2.0.

2.5.3 The optimization problem re-solution subroutine

The optimization problem solution algorithm may be recalled with modified problem data as follows:

```
CALL DPS_resolve( n, X, data, control, inform[, C, f, delta, sigma, p] )
```

n is a scalar INTENT(IN) argument of type INTEGER(ip_), that must be set to the number of unknowns, n. This should not have been changed since the last call to DPS_solve.

X is an array INTENT(OUT) argument of dimension n and type REAL(rp_), that holds an estimate of the solution $\mathbf{x}$ of the problem (1.1) or (1.2) on exit.

data is a scalar INTENT(INOUT) argument of type DPS_data_type (see Section 2.4.5). It is used to hold data about the problem being solved. It must not have been altered **by the user** since the last call to DPS_solve.

control is a scalar INTENT(IN) argument of type DPS_control_type. (see Section 2.4.2). Default values may be assigned by calling DPS_initialize prior to the first call to DPS_solve.

inform is a scalar INTENT(INOUT) argument of type DPS_inform_type (see Section 2.4.4) whose components need not be set on entry. A successful call to DPS_solve is indicated when the component status has the value 0. For other return values of status, see Section 2.6.

C is an OPTIONAL array INTENT(IN) argument of dimension n and type REAL(rp_), that if PRESENT whose i-th entry holds the component c_i of a new vector $\mathbf{c}$ for the objective function.

f is an OPTIONAL scalar INTENT(IN) variable of type REAL(rp_), that if PRESENT holds the a new value of the scalar f for the objective function.

delta is an OPTIONAL scalar INTENT(IN) variable of type default REAL(rp_), that must be set on entry to the value of the radius of the trust-region constraint, Δ if the solution to the trust-region subproblem (1.1) is required. If delta is not PRESENT, the trust-region subproblem will not be solved, but when it is PRESENT, the regularization subproblem will not be solved regardless of the status of sigma (see below). **Restriction:** delta $>$ 0.

sigma is an OPTIONAL scalar INTENT(IN) variable of type default REAL(rp_), that must be set on entry to the value of the regularization weight, σ if the solution to the regularization subproblem (1.2) is required. If sigma is not PRESENT, the regularization subproblem will not be solved. **Restriction:** sigma $>$ 0.

p is an OPTIONAL scalar INTENT(IN) variable of type default REAL(rp_), that must be set on entry to the value of the regularization order, p if the solution to the regularization subproblem (1.2) is required. If p is not PRESENT, the regularization order is taken to be 3.0. **Restriction:** p $\geq$ 2.0.

2.5.4 The termination subroutine

All previously allocated arrays are deallocated as follows:

```
CALL DPS_terminate( data, control, inform )
```

data is a scalar INTENT(INOUT) argument of type DPS_data_type exactly as for DPS_solve that must not have been altered **by the user** since the last call to DPS_initialize. On exit, array components will have been deallocated.

control is a scalar INTENT(IN) argument of type DPS_control_type exactly as for DPS_solve.

inform is a scalar INTENT(OUT) argument of type DPS_inform_type exactly as for DPS_solve. Only the component status will be set on exit, and a successful call to DPS_terminate is indicated when this component status has the value 0. For other return values of status, see Section 2.6.

2.6 Warning and error messages

A negative value of inform%status on exit from DPS_solve or DPS_terminate indicates that an error might have occurred. No further calls should be made until the error has been corrected. Possible values are:

-1. An allocation error occurred. A message indicating the offending array is written on unit control%error, and the returned allocation status and a string containing the name of the offending array are held in inform%alloc_status and inform%bad_alloc, respectively.

-2. A deallocation error occurred. A message indicating the offending array is written on unit control%error and the returned allocation status and a string containing the name of the offending array are held in inform%alloc_status and inform%bad_alloc, respectively.

-3. (DPS_solve and DPS_resolve only) One of the restrictions n > 0, radius > 0, sigma > 0 or p $\geq$ 2 has been violated.

-9. (DPS_solve only) The analysis phase of the factorization of the matrix **H** failed.

-10. (DPS_solve only) The factorization of the matrix **H** failed.

-16. (DPS_solve and DPS_resolve only) The problem is so ill-conditioned that further progress is impossible.

-40. (DPS_solve only) An error occurred when building the preconditioner.

2.7 Further features

In this section, we describe an alternative means of setting control parameters, that is components of the variable control of type DPS_control_type (see Section 2.4.2), by reading an appropriate data specification file using the subroutine DPS_read_specfile. This facility is useful as it allows a user to change DPS control parameters without editing and recompiling programs that call DPS.

A specification file, or specfile, is a data file containing a number of "specification commands". Each command occurs on a separate line, and comprises a "keyword", which is a string (in a close-to-natural language) used to identify a control parameter, and an (optional) "value", which defines the value to be assigned to the given control parameter. All keywords and values are case insensitive, keywords may be preceded by one or more blanks but values must not contain blanks, and each value must be separated from its keyword by at least one blank. Values must not contain more than 30 characters, and each line of the specfile is limited to 80 characters, including the blanks separating keyword and value.

The portion of the specification file used by DPS_read_specfile must start with a "BEGIN DPS" command and end with an "END" command. The syntax of the specfile is thus defined as follows:

```
( .. lines ignored by DPS_read_specfile .. )
  BEGIN DPS
     keyword    value
     .......    .....
     keyword    value
  END
( .. lines ignored by DPS_read_specfile .. )
```

where keyword and value are two strings separated by (at least) one blank. The "BEGIN DPS" and "END" delimiter command lines may contain additional (trailing) strings so long as such strings are separated by one or more blanks, so that lines such as

```
  BEGIN DPS SPECIFICATION
```

and

```
END DPS SPECIFICATION
```

are acceptable. Furthermore, between the "BEGIN DPS" and "END" delimiters, specification commands may occur in any order. Blank lines and lines whose first non-blank character is ! or * are ignored. The content of a line after a ! or * character is also ignored (as is the ! or * character itself). This provides an easy manner to "comment out" some specification commands, or to comment specific values of certain control parameters.

The value of a control parameters may be of three different types, namely integer, logical or real. Integer and real values may be expressed in any relevant Fortran integer and floating-point formats (respectively). Permitted values for logical parameters are "ON", "TRUE", ".TRUE.", "T", "YES", "Y", or "OFF", "NO", "N", "FALSE", ".FALSE." and "F". Empty values are also allowed for logical control parameters, and are interpreted as "TRUE".

The specification file must be open for input when DPS_read_specfile is called, and the associated device number passed to the routine in device (see below). Note that the corresponding file is REWINDed, which makes it possible to combine the specifications for more than one program/routine. For the same reason, the file is not closed by DPS_read_specfile.

Control parameters corresponding to the components SLS_control and IR_control may be changed by including additional sections enclosed by "BEGIN SLS" and "END SLS", and "BEGIN IR" and "END IR", respectively. See the specification sheets for the packages GALAHAD_SLS and GALAHAD_IR for further details.

2.7.1 To read control parameters from a specification file

Control parameters may be read from a file as follows:

```
CALL DPS_read_specfile( control, device )
```

control is a scalar INTENT(INOUT) argument of type DPS_control_type (see Section 2.4.2). Default values should have already been set, perhaps by calling DPS_initialize. On exit, individual components of control may have been changed according to the commands found in the specfile. Specfile commands and the component (see Section 2.4.2) of control that each affects are given in Table 2.1.

command	component of control	value type
error-printout-device	%error	integer
printout-device	%out	integer
print-level	%print_level	integer
use-dense-factorization	%dense_factorization	integer
has-h-changed	%new_h	integer
max-degree-taylor-approximant	%taylor_max_degree	integer
smallest-eigenvalue-allowed	%eigen_min	real
lower-bound-on-multiplier	%lower	real
upper-bound-on-multiplier	%upper	real
stop-normal-case	%stop_normal	real
stop-absolute-normal-case	%stop_absolute_normal	real
build-goldfarb-preconditioner	%goldfarb	logical
space-critical	%space_critical	logical
deallocate-error-fatal	%deallocate_error_fatal	logical
symmetric-linear-equation-solver	%symmetric_linear_solver	character
output-line-prefix	%prefix	character

Table 2.1: Specfile commands and associated components of control.

device is a scalar INTENT(IN) argument of type INTEGER(ip_), that must be set to the unit number on which the specfile has been opened. If device is not open, control will not be altered and execution will continue, but an error message will be printed on unit control%error.

2.8 Information printed

If control%print_level is positive, information about the progress of the algorithm will be printed on unit control%out. If control%print_level = 1, a single line of output will be produced for each iteration of the process. In the first phase of the algorithm, this will include the current estimate of the multiplier and known brackets on its optimal value. In the second phase, the residual $\|\mathbf{x}\|_{\mathbf{M}} - \Delta$ (in the trust-region case), the current estimate of the multiplier and the size of the correction will be printed. If control%print_level ≥ 2, this output will be increased to provide significant detail of each iteration. This extra output includes times for various phases.

3 GENERAL INFORMATION

Use of common: None.

Workspace: Provided automatically by the module.

Other routines called directly: None.

Other modules used directly: DPS_solve and DPS_resolve call the GALAHAD packages GALAHAD_CLOCK, GALAHAD_SYMBOLS, GALAHAD_SPACE, GALAHAD__NORMS, GALAHAD_SMT, GALAHAD_SPECFILE, GALAHAD_SLS, GALAHAD_TRS and GALAHAD_RQS.

Input/output: Output is under control of the arguments control%error, control%out and control%print_level.

Restrictions: $n > 0$, $\Delta > 0$, $\sigma > 0$, $p \geq 2$.

Portability: ISO Fortran 2003. The package is thread-safe.

4 METHOD

The required solution $\mathbf{x}_*$ necessarily satisfies the optimality condition $\mathbf{H}\mathbf{x}_* + \lambda_*\mathbf{M}\mathbf{x}_* + \mathbf{c} = \mathbf{0}$, where $\lambda_* \geq 0$ is a Lagrange multiplier that corresponds to the constraint $\|\mathbf{x}\|_{\mathbf{M}} \leq \Delta$ in the trust-region case (1.1), and is given by $\lambda_* = \sigma\|\mathbf{x}_*\|^{p-2}$ for the regularization problem (1.2). In addition $\mathbf{H} + \lambda_*\mathbf{M}$ will be positive semi-definite; in most instances it will actually be positive definite, but in special "hard" cases singularity is a possibility.

The matrix $\mathbf{H}$ is decomposed as

$$\mathbf{H} = \mathbf{P}\mathbf{L}\mathbf{D}\mathbf{L}^T\mathbf{P}^T$$

by calling the GALAHAD package SLS. Here $\mathbf{P}$ is a permutation matrix, $\mathbf{L}$ is unit lower triangular and $\mathbf{D}$ is block diagonal, with blocks of dimension at most two. The spectral decomposition of each diagonal block of $\mathbf{D}$ is computed, and each eigenvalue θ is replaced by $\max(|\theta|, \theta_{\min})$, where $\theta_{\min}$ is a positive user-supplied value. The resulting block diagonal matrix is $\mathbf{B}$, from which we define the "modified-absolute-value"

$$\mathbf{M} = \mathbf{P}\mathbf{L}\mathbf{B}\mathbf{L}^T\mathbf{P}^T;$$

an alternative due to Goldfarb uses instead the simpler

$$\mathbf{M} = \mathbf{P}\mathbf{L}\mathbf{L}^T\mathbf{P}^T.$$

Given the factors of $\mathbf{H}$ (and $\mathbf{M}$), the required solution is found by making the change of variables $\mathbf{y} = \mathbf{B}^{\frac{1}{2}}\mathbf{L}^T\mathbf{P}^T\mathbf{x}$ (or $\mathbf{y} = \mathbf{L}^T\mathbf{P}^T\mathbf{x}$ in the Goldfarb case) which results in "diagonal" trust-region and regularization subproblems, whose

solution may be easily obtained suing a Newton or higher-order iteration of a resulting "secular" equation. If subsequent problems, for which **H** and **c** are unchanged, are to be attempted, the existing factorization and solution may easily be exploited.

The dominant cost is that for the factorization of the symmetric, but potentially indefinite, matrix **H** using the GALAHAD package `SLS`.

Reference:

The method is described in detail for the trust-region case in

N. I. M. Gould and J. Nocedal (1998). The modified absolute-value factorization for trust-region minimization. In "High Performance Algorithms and Software in Nonlinear Optimization" (R. De Leone, A. Murli, P. M. Pardalos and G. Toraldo, eds.), Kluwer Academic Publishers, pp. 225–241,

while the adaptation for the regularization case is obvious. The method used to solve the diagonal trust-region and regularization subproblems are as given by

H. S. Dollar, N. I. M. Gould and D. P. Robinson (2010). On solving trust-region and other regularised subproblems in optimization. *Mathematical Programming Computation* **2(1)** 21–57

with simplifications due to the diagonal Hessian.

5 EXAMPLE OF USE

Suppose we wish to solve the trust-region subproblem (1.1) in 10 unknowns, whose data is

$$\mathbf{H} = \begin{pmatrix} -2 & 1 & & & & \\ 1 & -2 & . & & & \\ & . & . & . & & \\ & & . & -2 & 1 & \\ & & & 1 & -2 & \end{pmatrix}, \ \mathbf{c} = \begin{pmatrix} 1 \\ 1 \\ . \\ 1 \\ 1 \end{pmatrix} \ \text{and} \ f = 0,$$

with a radius $\Delta = 1$ and then to change the first component of c to 2, and to resolve with radii $\Delta = 1$ and $\Delta = 10$. Then we may use the following code:

```
PROGRAM GALAHAD_DPS_EXAMPLE   !  GALAHAD 3.0 - 23/03/2018 AT 07:30 GMT.
USE GALAHAD_DPS_double                       ! double precision version
IMPLICIT NONE
INTEGER, PARAMETER :: wp = KIND( 1.0D+0 )    ! set precision
INTEGER, PARAMETER :: n = 10                 ! problem dimension
INTEGER :: i
REAL ( KIND = wp ) :: f, delta
TYPE ( SMT_type ) :: H
REAL ( KIND = wp ), DIMENSION( n ) :: C, X
TYPE ( DPS_data_type ) :: data
TYPE ( DPS_control_type ) :: control
TYPE ( DPS_inform_type ) :: inform
INTEGER :: s
H%ne = 2 * n - 1                             ! set up problem
CALL SMT_put( H%type, 'COORDINATE', s )      ! specify co-ordinate for H
ALLOCATE( H%row( H%ne ), H%col( H%ne ), H%val( H%ne ), STAT = i )
DO i = 1, n - 1
   H%row( i ) = i ; H%col( i ) = i ; H%val( i ) = - 2.0_wp
   H%row( n + i ) = i + 1; H%col( n + i ) = i ; H%val( n + i ) = 1.0_wp
END DO
H%row( n ) = n ; H%col( n ) = n ; H%val( n ) = -2.0_wp
```

```
   C = 1.0_wp ; f = 0.0_wp ; delta = 1.0_wp
   CALL DPS_initialize( data, control, inform )  ! initialize control parameters
!  control%symmetric_linear_solver = "ma27 "
   CALL DPS_solve( n, H, C, f, X, data, control, inform, delta = delta )
   WRITE( 6, "( / A, ES12.4, A, / ( 5ES12.4 ) )" )                          &
            ' optimal f =', inform%obj, ', optimal x = ', X
   C( 1 ) = 2.0_wp                       ! change the first component of C to 2
   CALL DPS_resolve( n, X, data, control, inform, C = C, delta = delta )
   WRITE( 6, "( / A, /, A, ES12.4, A, / ( 5ES12.4 ) )" )                    &
     ' change C:', ' optimal f =', inform%obj, ', optimal x = ', X
   delta = 10.0_wp                       !  increase the radius
   CALL DPS_resolve( n, X, data, control, inform, delta = delta )
   WRITE( 6, "( / A, /, A, ES12.4, A, / ( 5ES12.4 ) )" )                    &
     ' increase delta:', ' optimal f =', inform%obj, ', optimal x = ', X
   CALL DPS_terminate( data, control, inform )  ! Deallocate arrays
   END PROGRAM GALAHAD_DPS_EXAMPLE
```

This produces the following output:

```
 optimal f = -3.7571E+00, optimal x =
 -2.7911E-01 -5.0239E-01 -2.7290E-01 -4.7758E-01 -2.6316E-01
 -4.3860E-01 -2.4561E-01 -3.6842E-01 -2.0468E-01 -2.0468E-01

 change C:
 optimal f = -4.1519E+00, optimal x =
 -4.9788E-01 -4.4809E-01 -2.4341E-01 -4.2596E-01 -2.3471E-01
 -3.9119E-01 -2.1907E-01 -3.2860E-01 -1.8255E-01 -1.8255E-01

 increase delta:
 optimal f = -8.6519E+01, optimal x =
 -4.9788E+00 -4.4809E+00 -2.4341E+00 -4.2596E+00 -2.3471E+00
 -3.9119E+00 -2.1907E+00 -3.2860E+00 -1.8255E+00 -1.8255E+00
```

If, instead, we wish to solve the cubic ($p = 3$) regularization subproblem (1.2) using the same problem data and initial weight $\sigma = 1$, and then to change the first component of c to 2, and to resolve with weights $\sigma = 1$ and $\sigma = 0.1$, the following code is suitable:

```
   PROGRAM GALAHAD_DPS_EXAMPLE2  !  GALAHAD 3.0 - 23/03/2018 AT 07:30 GMT.
   USE GALAHAD_DPS_double                       ! double precision version
   IMPLICIT NONE
   INTEGER, PARAMETER :: wp = KIND( 1.0D+0 )    ! set precision
   INTEGER, PARAMETER :: n = 10                 ! problem dimension
   INTEGER :: i
   REAL ( KIND = wp ) :: f, sigma, p
   TYPE ( SMT_type ) :: H
   REAL ( KIND = wp ), DIMENSION( n ) :: C, X
   TYPE ( DPS_data_type ) :: data
   TYPE ( DPS_control_type ) :: control
   TYPE ( DPS_inform_type ) :: inform
   INTEGER :: s
   H%ne = 2 * n - 1                             ! set up problem
   CALL SMT_put( H%type, 'COORDINATE', s )      ! specify co-ordinate for H
   ALLOCATE( H%row( H%ne ), H%col( H%ne ), H%val( H%ne ), STAT = i )
   DO i = 1, n - 1
      H%row( i ) = i ; H%col( i ) = i ; H%val( i ) = - 2.0_wp
      H%row( n + i ) = i + 1; H%col( n + i ) = i ; H%val( n + i ) = 1.0_wp
```

```
  END DO
  H%row( n ) = n ; H%col( n ) = n ; H%val( n ) = -2.0_wp
  C = 1.0_wp ; f = 0.0_wp ; sigma = 1.0_wp ; p = 3.0_wp
  CALL DPS_initialize( data, control, inform )  ! initialize control parameters
  CALL DPS_solve( n, H, C, f, X, data, control, inform, sigma = sigma, p = p )
  WRITE( 6, "( / A, ES12.4, A, / ( 5ES12.4 ) )" )                              &
            ' optimal f =', inform%obj, ', optimal x = ', X
  C( 1 ) = 2.0_wp                       ! change the first component of C to 2
  CALL DPS_resolve( n, X, data, control, inform, C = C, sigma = sigma )
  WRITE( 6, "( / A, /, A, ES12.4, A, / ( 5ES12.4 ) )" )                        &
    ' change C:', ' optimal f =', inform%obj, ', optimal x = ', X
  sigma = 0.1_wp                        !  decrease the weight
  CALL DPS_resolve( n, X, data, control, inform, sigma = sigma )
  WRITE( 6, "( / A, /, A, ES12.4, A, / ( 5ES12.4 ) )" )                        &
    ' decrease sigma:', ' optimal f =', inform%obj, ', optimal x = ', X
  CALL DPS_terminate( data, control, inform )  ! Deallocate arrays
  END PROGRAM GALAHAD_DPS_EXAMPLE2
```

This produces the following output:

```
optimal f = -1.0543E+01, optimal x =
-6.6225E-01 -1.1920E+00 -6.4753E-01 -1.1332E+00 -6.2441E-01
-1.0407E+00 -5.8278E-01 -8.7417E-01 -4.8565E-01 -4.8565E-01

change C:
optimal f = -1.2103E+01, optimal x =
-1.2324E+00 -1.1092E+00 -6.0251E-01 -1.0544E+00 -5.8099E-01
-9.6831E-01 -5.4226E-01 -8.1338E-01 -4.5188E-01 -4.5188E-01

decrease sigma:
optimal f = -1.2938E+02, optimal x =
-6.3944E+00 -5.7550E+00 -3.1262E+00 -5.4708E+00 -3.0145E+00
-5.0242E+00 -2.8135E+00 -4.2203E+00 -2.3446E+00 -2.3446E+00
```