


GALAHAD NREK

USER DOCUMENTATION GALAHAD Optimization Library version 5.4

1 SUMMARY

Given real n by n symmetric matrices $\mathbf{H}$ and $\mathbf{S}$ (with $\mathbf{S}$ diagonally dominant), a real n vector $\mathbf{c}$ and scalars f, $p > 2$ and $\sigma > 0$, this package uses an **extended Krylov-subspace method** to find a **global minimizer of the norm-regularised quadratic objective function** $\frac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x} + \mathbf{c}^T\mathbf{x} + f + \frac{1}{p}\sigma\|\mathbf{x}\|_{\mathbf{M}}^p$, where the $\mathbf{S}$-norm of $\mathbf{x}$ is $\|\mathbf{x}\|_{\mathbf{S}} = \sqrt{\mathbf{x}^T\mathbf{S}\mathbf{x}}$. This problem commonly occurs as a regularization subproblem in nonlinear optimization calculations. The matrix $\mathbf{S}$ need not be provided in the commonly-occurring ℓ_2-trust-region case for which $\mathbf{S} = \mathbf{I}$, the n by n identity matrix.

A factorization of the matrix $\mathbf{H}$ (and $\mathbf{S}$, if it is required) will be used, so this package is most suited for the case where such a factorization may be found efficiently. If this is not the case, the package `GALAHAD_GLRT` may be preferred.

ATTRIBUTES — Versions: `GALAHAD_NREK_single`, `GALAHAD_NREK_double`. **Uses:** `GALAHAD_CLOCK`, `GALAHAD_SYMBOLS`, `GALAHAD_SPACE`, `GALAHAD_NORMS`, `GALAHAD_ROOTS`, `GALAHAD_SPECFILE`, `GALAHAD_SMT`, `GALAHAD_RAND`, `GALAHAD_RQS`, `GALAHAD_SLS`, `GALAHAD_MOP` **Date:** November 2025. **Origin:** H. Al Daas and N. I. M. Gould, Rutherford Appleton Laboratory. **Language:** Fortran 95 + TR 15581 or Fortran 2003. **Parallelism:** Some options may use OpenMP and its runtime library.

2 HOW TO USE THE PACKAGE

The package is available with single, double and (if available) quadruple 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_NREK_single
```

with the obvious substitution `GALAHAD_NREK_double`, `GALAHAD_NREK_quadruple`, `GALAHAD_NREK_single_64`, `GALAHAD_NREK_double_64` and `GALAHAD_NREK_quadruple_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`, `NREK_control_type`, `NREK_inform_type`, `NREK_data_type`, (Section 2.4) and the subroutines `NREK_initialize`, `NREK_solve`, `NREK_terminate` (Section 2.5) and `NREK_read_specfile` (Section 2.7) must be renamed on one of the `USE` statements.

2.1 Matrix storage formats

The matrices $\mathbf{H}$ and (if required) $\mathbf{S}$ 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 $1 \leq j \leq i \leq n$. The same is true for $\mathbf{S}$ if it is used.

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. The same scheme may be used for $\mathbf{S}$ if it is required.

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 `H%ptr` holds the position of the first entry in this row, while `H%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. This scheme may also be used for $\mathbf{S}$ if it is required.

2.1.4 Diagonal storage format

If $\mathbf{H}$ is diagonal (i.e., $h_{ij} = 0$ for all $1 \leq i \neq j \leq n$) only the diagonals entries h_{ii}, $1 \leq i \leq n$, need be stored, and the first n components of the array `H%val` may be used for the purpose. The same applies to $\mathbf{S}$ if it is required.

2.1.5 Scaled-identity-matrix storage format

If $\mathbf{H}$ is a scalar multiple of the identity matrix (i.e., $h_{ii} = h_{11}$ and $h_{ij} = 0$ for all $1 \leq i \neq j \leq n$) only the first diagonal entry h_{11} needs be stored, and the first component of the array `H%val` may be used for the purpose. The same applies to $\mathbf{S}$ if it is required.

2.1.6 Identity-matrix storage format

If $\mathbf{H}$ is the identity matrix (i.e., $h_{ii} = 1$ and $h_{ij} = 0$ for all $1 \leq i \neq j \leq n$), no explicit entries needs be stored. The same would be true for $\mathbf{S}$, but $\mathbf{S}$ need not be provided if $\mathbf{S} = \mathbf{I}$.

2.1.7 Zero-matrix storage format

If $\mathbf{H} = \mathbf{0}$ (i.e., $h_{ij} = 0$ for all $1 \leq i, j \leq n$), no explicit entries needs be stored. This is not relevant for $\mathbf{S}$, as $\mathbf{S}$ is required to be non-singular.

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, `DOUBLE PRECISION` for the double precision cases and quadruple-precision if 128-bit reals are available, and correspond to `rp_ = real32`, `rp_ = real64` and `rp_ = real128` respectively as defined 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_NREK` 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

Six 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 **H** and perhaps **S**. 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 NREK_control_type is used to hold controlling data. Default values may be obtained by calling NREK_initialize (see Section 2.5.1). The components of NREK_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 NREK_solve and NREK_terminate is suppressed if error $\leq$ 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 NREK_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 $\leq$ 0. If print_level = 1 a single line of output will be produced for each iteration of the process. If print_level $\geq$ 2 this output will be increased to provide significant detail of each iteration. The default is print_level = 0.

eks_max is a scalar variable of type INTEGER(ip_), that specifies the maximum dimension of the extended Krylov space employed. If a negative value is given, this will be replaced by 100. The default is eks_max = - 1.

it_max is a scalar variable of type INTEGER(ip_), that is used to specify the maximum number of iterations allowed. If a negative value is given, this will be replaced by 100. The default is it_max = - 1.

f is a scalar variable of type default REAL(rp_), that gives the value of the constant term f in the quadratic objective function. This value has no effect on the computed minimizer $\mathbf{x}$. The default is f = 0.0.

increase is a scalar variable of type REAL(rp_), that holds the value of the increase factor for suggested subsequent regularization weights (see control%next_weight). The suggested weight will be reduction times the current weight. The default is increase = 2.0.

stop_residual is a scalar variable of type REAL(rp_), that holds the value of the stopping tolerance used by the algorithm. The iteration stops as soon as $\mathbf{x}$ and λ are found to satisfy $\|(\mathbf{H}+\lambda\mathbf{S})\mathbf{x}+\mathbf{c}\| <$ stop_residual $\times \max(1,\|\mathbf{c}\|)$. The default is stop_residual = $10\sqrt{u}$, where u is EPSILON(1.0_rp_)

reorthogonalize is a scalar variable of type default LOGICAL, that may be set .TRUE. if the user wishes the package to reorthogonalise the generated basis of the extended Krylov space at every iteration. This can be very expensive, and is generally not warranted. The default is reorthogonalize = .FALSE. which is our recommendation.

s_version_52 is a scalar variable of type default LOGICAL, that may be set .TRUE. if the user wishes to use Algorithm 5.2 in the paper to generate the extended Krylov space recurrences when a non-unit S is given, and .FALSE. if those from Algorithm B.3 should be used instead. In practice, there is very little difference in performance and accuracy. The default is s_version_52 = .TRUE..

perturb_c is a scalar variable of type default LOGICAL, that may be set .TRUE. if the user wishes the package make a tiny pseudo-random perturbations to the components of the term $\mathbf{c}$ to try to protect from the so-called (probability zero) "hard case". Perturbations are generally not needed, and should only be used in very exceptional cases. The default is perturb_c = .FALSE..

stop_check_all_orders is a scalar variable of type default LOGICAL, that may be set .TRUE. if the user wishes the package to check for termination for each new member of the extended Krylov space. Such checks incur some extra cost, and experience shows that testing every second member is sufficient, using stop_check_all_orders = .FALSE.. The default is stop_check_all_orders = .FALSE..

new_weight is a scalar variable of type default LOGICAL, that if .TRUE. will resolve the problem with the previous $\mathbf{H}$, $\mathbf{c}$ and (if provided) $\mathbf{S}$, but with the larger radius, radius, provided to NREK_solve. The default is new_radius = .FALSE..

new_values is a scalar variable of type default LOGICAL, that if PRESENT and .TRUE. will resolve the problem with the data structures set for the previous $\mathbf{H}$ and (if provided) $\mathbf{S}$, but with new values H%val, C and/or S%val provided to NREK_solve. The default is new_values = .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..

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 positive-definite linear system involving **H** that might arise. Current possible choices are 'sils', 'ma27', 'ma57', 'ma77', 'ma86', 'ma87', 'ma97', ssids, 'pardiso', 'wsmp', 'sytr', 'potr' and 'pbtr' although only 'sytr', 'potr', 'pbtr' 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 linear_solver = 'ssids', but we recommend 'pbtr' instead if **H** is banded with a small bandwidth.

linear_solver_for_S is a scalar variable of type default CHARACTER and length 30, that specifies the external package to be used to solve any symmetric positive-definite linear system involving the optional **S** that might arise. Current possible choices are 'sils', 'ma27', 'ma57', 'ma77', 'ma86', 'ma87', 'ma97', ssids, 'pardiso', 'wsmp', 'sytr', 'potr' and 'pbtr' although only 'sytr', 'potr', 'pbtr' 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 linear_solver = 'ssids', but we recommend 'pbtr' instead if **S** is banded with a small bandwidth.

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.

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

TRS_control is a scalar variable of type TRS_control_type that is used to control various aspects of the diagonal trust-region solver from the package TRS. See the documentation for GALAHAD_TRS for more details.

2.4.3 The derived data type for holding timing information

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

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

assemble is a scalar variable of type REAL(rp_), that gives the CPU time spent assembling the matrices **H** and (if provided) **S** from their constituent parts.

analyse is a scalar variable of type REAL(rp_), that gives the CPU time spent analysing the matrices **H** and (if provided) **S** prior to factorization.

factorize is a scalar variable of type REAL(rp_), that gives the CPU time spent factorizing the matrices **H** and (if provided) **S**.

solve is a scalar variable of type REAL(rp_), that gives the CPU time spent using the factors to solve relevant linear equations.

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

clock_assemble is a scalar variable of type REAL(rp_), that gives the elapsed system clock time spent assembling the matrices **H** and (if provided) **S** from their constituent parts.

clock_analyse is a scalar variable of type REAL(rp_), that gives the elapsed system clock time spent analysing the matrices **H** and (if provided) **S** 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 system clock time spent using the factors to solve relevant linear equations.

2.4.4 The derived data type for holding informational parameters

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

iter is a scalar variable of type INTEGER(ip_), that gives the total number of iterations required.

n_vec is a scalar variable of type INTEGER(ip_), that gives the number of orthogonal vectors required (the dimension of the extended-Krylov subspace).

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

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

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

weight is a scalar variable of type REAL(rp_), that holds the value of the current regularization weight.

next_weight is a scalar variable of type REAL(rp_), that holds the value of the proposed next weight to be used if the current weight proves to be too large (see inform%increase).

error is a scalar variable of type REAL(rp_), that holds the value of the norm of the maximum relative residual error, $\|(\mathbf{H}+\lambda\mathbf{S})\mathbf{x}+\mathbf{c}\|/\max(1,\|\mathbf{c}\|)$.

time is a scalar variable of type NREK_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 for $\mathbf{H}$ 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.

SLS_S_inform is a scalar variable of type SLS_inform_type, that holds informational parameters concerning the analysis, factorization and solution phases for $\mathbf{S}$, if present, 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.

TRS_inform is a scalar variable of type TRS_inform_type, that holds informational parameters concerning the diagonal trust-region subroutine contained in the GALAHAD refinement package TRS. See the documentation for the package TRS for details of the derived type TRS_inform_type.

2.4.5 The derived data type for holding problem data

The derived data type NREK_data_type is used to hold all the data for a particular problem between calls of NREK procedures. This data should be preserved, untouched, from the initial call to NREK_initialize to the final call to NREK_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 NREK_initialize is used to set default values and initialize private data.
2. The subroutine NREK_solve is called to solve the problem.
3. The subroutine NREK_terminate is provided to allow the user to automatically deallocate array components of the private data, allocated by NREK_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 NREK_initialize( data, control, inform )
```

data is a scalar INTENT(INOUT) argument of type NREK_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 NREK_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 NREK_initialize.

inform is a scalar INTENT(OUT) argument of type NREK_inform_type (see Section 2.4.4). A successful call to NREK_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 trust-region solution algorithm is called as follows:

```
CALL NREK_solve( n, H, C, radius, X, data, control, inform[, S])
```

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, 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, and for the diagonal storage scheme (see Section 2.1.4), the first eight components of H%type must contain the

string DIAGONAL, for the scaled-identity matrix storage scheme (see Section 2.1.5), the first fifteen components of H%type must contain the string SCALED_IDENTITY, for the identity matrix storage scheme (see Section 2.1.6), the first eight components of H%type must contain the string IDENTITY, and for the zero matrix storage scheme (see Section 2.1.7), the first four components of H%type must contain the string ZERO.

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 **S** 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 three 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 or diagonal storage schemes are 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.

p is a scalar INTENT(IN) variable of type REAL(rp_), that must be set on initial entry to the order of the regularisation, p. **Restriction:** $p > 2$.

sigma is a scalar INTENT(IN) variable of type REAL(rp_), that must be set on initial entry to the value of the regularisation weight, σ. **Restriction:** $\sigma > 0$.

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 on exit.

data is a scalar INTENT(INOUT) argument of type NREK_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 NREK_initialize.

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

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

S is an OPTIONAL scalar INTENT(IN) argument of type SMT_TYPE that holds the diagonally dominant scaling matrix **S**. It need only be set if $\mathbf{S} \neq \mathbf{I}$ and in this case the following components are used:

S%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 S%type must contain the string DENSE. For the sparse co-ordinate scheme (see Section 2.1.2), the first ten components of S%type must contain the string COORDINATE, for the sparse row-wise storage scheme (see Section 2.1.3), the first fourteen components of S%type must contain the string SPARSE_BY_ROWS, and for the diagonal storage scheme (see Section 2.1.4), the first eight components of S%type must contain the string DIAGONAL, for the scaled-identity matrix storage scheme (see Section 2.1.5), the first fifteen components of S%type must contain the string SCALED_IDENTITY, for the identity matrix storage scheme (see Section 2.1.6), the first eight components of S%type must contain the string IDENTITY, and for the zero matrix storage scheme (see Section 2.1.7), the first four components of S%type must contain the string ZERO.

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

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

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

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

S%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 scaling matrix **S** in any of the storage schemes discussed in Section 2.1.

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

S%col is a rank-one allocatable array variable of type INTEGER(ip_), that holds the column indices of the **lower triangular** part of **S** 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 or diagonal storage schemes are used.

S%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 **S**, 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.

If S is absent, the ℓ_2-norm, $\|\mathbf{x}\|_2 = \sqrt{\mathbf{x}^T\mathbf{x}}$, will be employed.

2.5.3 The termination subroutine

All previously allocated arrays are deallocated as follows:

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

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

control is a scalar INTENT(IN) argument of type NREK_control_type exactly as for NREK_solve.

inform is a scalar INTENT(OUT) argument of type NREK_inform_type exactly as for NREK_solve. Only the component status will be set on exit, and a successful call to NREK_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 NREK_solve or NREK_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. (NREK_solve only) One of the restrictions n > 0 radius > 0, H%n $\neq$ n, S%n $\neq$ n or the radius has not decreased when resolve = /true/ has been violated.

-9. (NREK_solve only) The analysis phase of the factorization of one of the matrices **H** and (if provided) **S** failed.

-10. (NREK_solve only) The factorization of one of the matrices **H** and (if provided) **S** failed.

-11. (NREK_solve only) The solve phase involving one of the matrices **H** and (if provided) **S** failed.

-15. (NREK_solve only) The matrix **S** appears not to be diagonally dominant.

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

-18. (NREK_solve only) Too many iterations have been required. This may happen if control%eks_max is too small, but may also be symptomatic of a badly scaled problem.

-31. (NREK_solve only) A resolve call has been made before an initial call (see control%new_radius and control%new_values).

-38. (NREK_solve only) An error occurred in a call to an LAPACK subroutine.

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 NREK_control_type (see Section 2.4.2), by reading an appropriate data specification file using the subroutine NREK_read_specfile. This facility is useful as it allows a user to change NREK control parameters without editing and recompiling programs that call NREK.

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 NREK_read_specfile must start with a "BEGIN NREK" command and end with an "END" command. The syntax of the specfile is thus defined as follows:

```
( .. lines ignored by NREK_read_specfile .. )
  BEGIN NREK
     keyword    value
     .......    .....
     keyword    value
  END
( .. lines ignored by NREK_read_specfile .. )
```

where keyword and value are two strings separated by (at least) one blank. The "BEGIN NREK" 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 NREK SPECIFICATION
```

and

```
END NREK SPECIFICATION
```

are acceptable. Furthermore, between the "BEGIN NREK" 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 NREK_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 NREK_read_specfile.

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

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 NREK_read_specfile( control, device )
```

control is a scalar INTENT(INOUT) argument of type NREK_control_type (see Section 2.4.2). Default values should have already been set, perhaps by calling NREK_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.

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. This will include the size of the subspace, the norm of the current estimate of $\mathbf{x}$ and the radius, the current estimate of the shift λ, the error $\|(\mathbf{H}+\lambda\mathbf{S})\mathbf{x}+\mathbf{c}\|/\max(1,\|\mathbf{c}\|)$, and the current value of the objective function. If control%print_level $\geq$ 2, this output will be increased to provide significant detail of each iteration. This extra output includes times for various phases.

command	component of control	value type
error-printout-device	%error	integer
printout-device	%out	integer
print-level	%print_level	integer
maximum-subspace-dimension	%eks_max	integer
maximum-number-of-iterations	%it_max	integer
weight-increase-factor	%increase	real
residual-accuracy	%stop_residual	real
reorthogonalize-vectors	%reorthogonalize	logical
s-version-52	%s_version_52	logical
stop-check-all-orders	%stop-check-all-orders	logical
perturb-c	%perturb-c	logical
space-critical	%space_critical	logical
deallocate-error-fatal	%deallocate_error_fatal	logical
linear-equation-solver	%linear_solver	character
linear-equation-solver-for-S	%linear_solver_for_S	character
output-line-prefix	%prefix	character

Table 2.1: Specfile commands and associated components of control.

3 GENERAL INFORMATION

Use of common: None.

Workspace: Provided automatically by the module.

Other routines called directly: None.

Other modules used directly: NREK_solve calls the GALAHAD packages GALAHAD_CLOCK, GALAHAD_SYMBOLS, GALAHAD_SPACE, GALAHAD_NORMS, GALAHAD_ROOTS, GALAHAD_SPECFILE, GALAHAD_SMT, GALAHAD_RAND, GALAHAD_RQS, GALAHAD_SLS, and GALAHAD_MOP.

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

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

Portability: ISO Fortran 95 + TR 15581 or Fortran 2003. The package is thread-safe.

4 METHOD

The required solution $\mathbf{x}_*$ necessarily satisfies the optimality condition $\mathbf{H}\mathbf{x}_* + \lambda_*\mathbf{S}\mathbf{x}_* + \mathbf{c} = \mathbf{0}$, where $\lambda_* = \sigma\|\mathbf{x}_*\|^{p-2}$ is a Lagrange multiplier corresponding to the regularization. In addition in all cases, the matrix $\mathbf{H} + \lambda_*\mathbf{S}$ will be positive semi-definite; in most instances it will actually be positive definite, but in special "hard" cases singularity is a possibility.

The method is iterative, and is based upon building a solution approximation from an orthogonal basis of the evolving extended Krylov subspaces $\mathcal{K}_{2m+1}(\mathbf{H},\mathbf{c}) = \text{span}\{\mathbf{c},\mathbf{H}^{-1}\mathbf{c},\mathbf{H}\mathbf{c},\mathbf{H}^{-2}\mathbf{c},\mathbf{H}^2\mathbf{c},\ldots,\mathbf{H}^{-m}\mathbf{c},\mathbf{H}^m\mathbf{c}\}$ as m increases. The key observations are (i) the manifold of solutions to the optimality system

$$(\mathbf{H} + \lambda\mathbf{I})\mathbf{x}(\lambda) = -\mathbf{c}$$

as a function of σ is of approximately very low rank, (ii) the subspace $\mathcal{K}_{2m+1}(\mathbf{H},\mathbf{c})$ rapidly gives a very good approximation to this manifold, (iii) it is straightforward to build an orthogonal basis of $\mathcal{K}_{2m+1}(\mathbf{H},\mathbf{c})$ using short-term

recurrences and a single factorization of **H**, and (iv) solutions to the regularization subproblem restricted to elements of the orthogonal subspace may be found very efficiently using effective high-order root-finding methods. Coping with general scalings **S** is a straightforward extension so long as factorization of **S** is also possible.

Reference: The method is described in detail in

H. Al Daas and N. I. M. Gould. Extended-Krylov-subspace methods for trust-region and norm-regularization subproblems. Preprint STFC-P-2025-002, Rutherford Appleton Laboratory, Oxfordshire, England.

5 EXAMPLE OF USE

Suppose we wish to solve a problem in 10,000 unknowns, whose data is

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

starting with a regularisation weight $\sigma = 10$ and order $p = 3$ but then gradually doubling the weight at every subsequent stage. Then we may use the following code:

This produces the following output: