Data Fitting Functions

Data Fitting functions in Intel® MKL provide spline-based interpolation capabilities that you can use to approximate functions, function derivatives or integrals, and perform cell search operations.

The Data Fitting component is task based. The task is a data structure or descriptor that holds the parameters related to a specific Data Fitting operation. You can modify the task parameters using the task editing functionality of the library.

For definition of the implemented operations, see Mathematical Conventions.

Data Fitting routines use the following workflow to process a task:

1. Create a task or multiple tasks.
2. Modify the task parameters.
3. Perform a Data Fitting computation.
4. Destroy the task or tasks.

All Data Fitting functions fall into the following categories:

Task Creation and Initialization Routines - routines that create a new Data Fitting task descriptor and initialize the most common parameters, such as partition of the interpolation interval, values of the vector-valued function, and the parameters describing their structure.

Task Editors - routines that set or modify parameters in an existing Data Fitting task.

Computational Routines - routines that perform Data Fitting computations, such as construction of a spline, interpolation, computation of derivatives and integrals, and search.

Task Destructors - routines that delete Data Fitting task descriptors and deallocate resources.

You can access the Data Fitting routines through the Fortran and C89/C99 language interfaces. You can also use the C89 interface with more recent versions of C/C++, or the Fortran 90 interface with programs written in Fortran 95

The `${MKL}/include` directory of the Intel® MKL contains the following Data Fitting header files:

- C/C++: `mkl_df.h`
- Fortran: `mkl_df.f90` and `mkl_df.f77`

You can find examples that demonstrate C/C++ and Fortran usage of Data Fitting routines in the `${MKL}/examples/datafittingc` and `${MKL}/examples/datafittingf` directories, respectively.

Naming Conventions

The Fortran interfaces of the Data Fitting functions are in lowercase, while the names of the types and constants are in uppercase.

The C/C++ interface of the Data Fitting functions, types, and constants are case-sensitive and can be in lowercase, uppercase, and mixed case.

The names of all routines have the following structure:

`df[`*`datatype`*`]<`*`base_name`*`>`

where

- `df` is a prefix indicating that the routine belongs to the Data Fitting component of Intel MKL.
- `[`*`datatype`*`]` field specifies the type of the input and/or output data and can be `s` (for the single precision real type), `d` (for the double precision real type), or `i` (for the integer type). This field is omitted in the names of the routines that are not data type dependent.
- *`<base_name>`* field specifies the functionality the routine performs. For example, this field can be `NewTask1D`, `Interpolate1D`, or `DeleteTask`.

Data Types

The Data Fitting component provides routines for processing single and double precision real data types. The results of cell search operations are returned as a generic integer data type.

All Data Fitting routines use the following data type:

Type	Data Object
Fortran: `TYPE(DF_TASK)`	Pointer to a task
C: `DFTaskPtr`	

NOTE
The actual size of the generic integer type is platform-dependent. Before compiling your application, you need to set an appropriate byte size for integers. For details, see section *Using the ILP64 Interface vs. LP64 Interface* of the Intel® MKL User's Guide.

Mathematical Conventions

This section explains the notation used for Data Fitting function descriptions. Spline notations are based on the terminology and definitions of [deBoor2001]. The Subbotin quadratic spline definition follows the conventions of [StechSub76]. The quasi-uniform partition definition is based on [Schumacher2007]

Mathematical Notation in the Data Fitting Component

Concept	Mathematical Notation
Partition of interpolation interval [a, b] , where • x_i denotes breakpoints. • [x_i, x_{i+1}) denotes a sub-interval (cell) of size $\Delta_i = x_{i+1} - x_i$.	$\{x_i\}_{i=1,...,n}$, where $a = x_1 < x_2 < ... < x_n = b$
Quasi-uniform partition of interpolation interval [a, b]	Partition $\{x_i\}_{i=1,...,n}$ which meets the constraint with a constant C defined as $1 \le M/ m \le C$, where • $M = \max_{i=1,...,n-1} (\Delta_i)$ • $m = \min_{i=1,...,n-1} (\Delta_i)$ • $\Delta_i = x_{i+1} - x_i$
Vector-valued function of dimension p being fit	$f(x) = (f_1(x),..., f_p(x))$
Piecewise polynomial (PP) function f of order $k+1$	$f(x) := P_i(x)$, if $x \in [x_i, x_{i+1})$, $i = 1,..., n-1$ where • $\{x_i\}_{i=1,..., n}$ is a strictly increasing sequence of breakpoints. • $P_i(x) = c_{i,0} + c_{i,1}(x - x_i) + ... + c_{i,k}(x - x_i)^k$ is a polynomial of degree k (order k+1) over the interval $x \in [x_i, x_{i+1})$.
Function p agrees with function g at the points $\{x_i\}_{i=1,...,n}$.	For every point ζ in sequence $\{x_i\}_{i=1,...,n}$ that occurs m times, the equality $p^{(i-1)}(\zeta) = g^{(i-1)}(\zeta)$ holds for all $i = 1,...,m$, where $p^{(i)}(t)$ is the derivative of the i-th order.

Concept	Mathematical Notation
The k-th divided difference of function g at points $x_i,\ldots, x_{i+k}$. This difference is the leading coefficient of the polynomial of order k+1 that agrees with g at $x_i,\ldots, x_{i+k}$.	$[x_i,\ldots, x_{i+k}]\, g$ In particular, • $[x_1]g = g(x_1)$ • $[x_1, x_2]\, g = (g(x_1) - g(x_2)) / (x_1 - x_2)$
A k-order derivative of interpolant $f(x)$ at interpolation site τ.	$f^{(k)}(\tau)$

Interpolants to the Function g at $x_1,\ldots, x_n$ and Boundary Conditions

Concept	Mathematical Notation
Linear interpolant	$P_i(x) = c_{1,i} + c_{2,i}(x - x_i)$, where • $x \in [x_i, x_{i+1})$ • $c_{1,i} = g(x_i)$ • $c_{2,i} = [x_i, x_{i+1}]g$ • $i = 1,\ldots, n-1$
Piecewise parabolic interpolant	$P_i(x) = c_{1,i} + c_{2,i}(x - x_i) + c_{3,i}(x - x_i)^2$, $x \in [x_i, x_{i+1})$ Coefficients $c_{1,i}$, $c_{2,i}$, and $c_{3,i}$ depend on the conditions: • $P_i(x_i) = g(x_i)$ • $P_i(x_{i+1}) = g(x_{i+1})$ • $P_i((x_{i+1} + x_i) / 2) = v_{i+1}$ where parameter v_{i+1} depends on the interpolant being continuously differentiable: $P_{i-1}^{(1)}(x_i) = P_i^{(1)}(x_i)$
Piecewise parabolic Subbotin interpolant	$P(x) = P_i(x) = c_{1,i}+c_{2,i}(x-x_i)+c_{3,i}(x-x_i)^2+d_{3,i}((x-t_i)_+)^2$, where • $x \in [t_i, t_{i+1})$ • $\{t_i\}_{i=1,\ldots,n+1}$ is a sequence of knots such that • $t_1 = x_1$, $t_{n+1} = x_n$ • $t_i \in (x_{i-1}, x_i)$, $i = 2,\ldots, n$ • $x_+ = f(x) = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{if } x \geq 0 \end{cases}$ Coefficients $c_{1,i}$, $c_{2,i}$, $c_{3,i}$, and $d_{3,i}$ depend on the following conditions: • $P_i(x_i) = g(x_i)$, $P_i(x_{i+1}) = g(x_{i+1})$ • $P(x)$ is a continuously differentiable polynomial of the second degree on $[t_i, t_{i+1})$, $i = 1,\ldots, n$.
Piecewise cubic Hermite interpolant	$P_i(x) = c_{1,i} + c_{2,i}(x - x_i) + c_{3,i}(x - x_i)^2 + c_{4,i}(x - x_i)^3$, where • $x \in [x_i, x_{i+1})$ • $c_{1,i} = g(x_i)$ • $c_{2,i} = s_i$ • $c_{3,i} = ([x_i, x_{i+1}]g - s_i) / (\Delta x_i) - c_{4,i}(\Delta x_i)$ • $c_{4,i} = (s_i + s_{i+1} - 2[x_i, x_{i+1}]g) / (\Delta x_i)^2$

Concept	Mathematical Notation
	• $i = 1,\ldots, n-1$ • $s_i = g^{(1)}(x_i)$
Piecewise cubic Bessel interpolant	$P_i(x) = c_{1,i} + c_{2,i}(x - x_i) + c_{3,i}(x - x_i)^2 + c_{4,i}(x - x_i)^3$, where • $x\in [\ x_i, x_{i+1})$ • $c_{1,i} = g(x_i)$ • $c_{2,i} = s_i$ • $c_{3,i} = ([x_i, x_{i+1}]g - s_i\ ) / (\Delta x_i) - c_{4,i}(\Delta x_i)$ • $c_{4,i} = (s_i + s_{i+1} - 2[x_i, x_{i+1}]g) / (\Delta x_i)^2$ • $i = 1,\ldots, n-1$ • $s_i = (\Delta x_i[x_{i-1}, x_i]g + \Delta x_{i-1}[x_i, x_{i+1}]g) / (\Delta x_i + \Delta x_{i+1})$
Piecewise cubic Akima interpolant	$P_i(x) = c_{1,i} + c_{2,i}(x - x_i) + c_{3,i}(x - x_i)^2 + c_{4,i}(x - x_i)^3$, where • $x\in [\ x_i, x_{i+1})$ • $c_{1,i} = g(x_i)$ • $c_{2,i} = s_i$ • $c_{3,i} = ([x_i, x_{i+1}]g - s_i\ ) / (\Delta x_i) - c_{4,i}(\Delta x_i)$ • $c_{4,i} = (s_i + s_{i+1} - 2[x_i, x_{i+1}]g) / (\Delta x_i)^2$ • $i = 1,\ldots, n-1$ • $s_i = (w_{i+1}[x_{i-1}, x_i]g + w_{i-1}[x_i, x_{i+1}]g) / (w_{i+1} + w_{i-1})$, where $w_i = \|[x_i, x_{i+1}]g - [x_{i-1}, x_i]g\|$
Piecewise natural cubic interpolant	$P_i(x) = c_{1,i} + c_{2,i}(x - x_i) + c_{3,i}(x - x_i)^2 + c_{4,i}(x - x_i)^3$, where • $x\in [\ x_i, x_{i+1})$ • $c_{1,i} = g(x_i)$ • $c_{2,i} = s_i$ • $c_{3,i} = ([x_i, x_{i+1}]g - s_i\ ) / (\Delta x_i) - c_{4,i}(\Delta x_i)$ • $c_{4,i} = (s_i + s_{i+1} - 2[x_i, x_{i+1}]g) / (\Delta x_i)^2$ • $i = 1,\ldots, n-1$ • Parameter s_i depends on the condition that the interpolant is twice continuously differentiable: $P_{i-1}^{(2)}(x_i) = P_i^{(2)}(x_i)$.
Not-a-knot boundary condition.	Parameters s_1 and s_n provide $P_1 = P_2$ and $P_{n-1} = P_n$, so that the first and the last interior breakpoints are inactive.
Free-end boundary condition.	$f''(x_1) = f''(x_n) = 0$
Look-up interpolator for discrete set of points $(x_1, y_1),\ldots, (x_n, y_n)$.	$y(x) = \begin{cases} y_1, & \text{if } x = x_1 \\ y_2, & \text{if } x = x_2 \\ & \ldots \\ y_n, & \text{if } x = x_n \\ error, & otherwise \end{cases}$

Concept	Mathematical Notation
Step-wise constant continuous right interpolator.	$y(x)=\begin{cases} y_1, & \text{if } x_1 \le x < x_2 \\ y_2, & \text{if } x_2 \le x < x_3 \\ & \dots \\ y_{n-1}, & \text{if } x_{n-1} \le x < x_n \\ y_n, & \text{if } x = x_n \end{cases}$
Step-wise constant continuous left interpolator.	$y(x)=\begin{cases} y_1, & \text{if } x = x_1 \\ y_2, & \text{if } x_1 < x \le x_2 \\ y_3, & \text{if } x_2 < x \le x_3 \\ & \dots \\ y_n, & \text{if } x_{n-1} < x \le x_n \end{cases}$

Data Fitting Usage Model

Consider an algorithm that uses the Data Fitting functions. Typically, such algorithms consist of four steps or stages:

1. Create a task. You can call the Data Fitting function several times to create multiple tasks.

   ```
   status = dfdNewTask1D( &task, nx, x, xhint, ny, y, yhint );
   ```
2. Modify the task parameters.

   ```
   status = dfdEditPPSpline1D( task, s_order, c_type, bc_type, bc, ic_type, ic,
   scoeff, scoeffhint );
   ```
3. Perform Data Fitting spline-based computations. You may reiterate steps 2-3 as needed.

   ```
   status = dfdInterpolate1D(task, estimate, method, nsite, site, sitehint, ndorder,
   dorder, datahint, r, rhint, cell );
   ```
4. Destroy the task or tasks.

   ```
   status = dfDeleteTask( &task );
   ```

See Also

Data Fitting Usage Examples

Data Fitting Usage Examples

The examples below illustrate several operations that you can perform with Data Fitting routines. If you want to run or reuse similar examples, you can get both C and Fortran source code in the `.\examples\datafittingc` and `.\examples\datafittingf` subdirectories of the Intel MKL installation directory.

The following C example demonstrates the construction of a linear spline using Data Fitting routines. The spline approximates a scalar function defined on non-uniform partition. The coefficients of the spline are returned as a one-dimensional array:

C Example of Linear Spline Construction

```
#include "mkl.h"
#define N 500                    /* Size of partition, number of breakpoints */
#define SPLINE_ORDER DF_PP_LINEAR /* Linear spline to construct */

int main()
```

```
{
    int status;          /* Status of a Data Fitting operation */
    DFTaskPtr task;      /* Data Fitting operations are task based */

    /* Parameters describing the partition */
    MKL_INT nx;          /* The size of partition x */
    double x[N];         /* Partition x */
    MKL_INT xhint;       /* Additional information about the structure of breakpoints */

    /* Parameters describing the function */
    MKL_INT ny;          /* Function dimension */
    double y[N];         /* Function values at the breakpoints */
    MKL_INT yhint;       /* Additional information about the function */

    /* Parameters describing the spline */
    MKL_INT  s_order;    /* Spline order */
    MKL_INT  s_type;     /* Spline type */
    MKL_INT  ic_type;    /* Type of internal conditions */
    MKL_INT* ic;         /* Array of internal conditions */
    MKL_INT  bc_type;    /* Type of boundary conditions */
    MKL_INT* bc;         /* Array of boundary conditions */

    double scoeff[(N-1)* ORDER];   /* Array of spline coefficients */
    MKL_INT scoeffhint;            /* Additional information about the coefficients */

    /* Initialize the partition */
    nx = N;
    /* Set values of partition x */
    ...
    xhint = DF_NO_HINT;    /* No additional information about the function is provided.
                              By default, the partition is non-uniform. */
    /* Initialize the function */
     ny = 1;               /* The function is scalar. */

    /* Set function values */
    ...
    yhint = DF_NO_HINT;    /* No additional information about the function is provided. */

    /* Create a Data Fitting task */
    status = dfdNewTask1D( &task, nx, x, xhint, ny, y, yhint );

    /* Check the Data Fitting operation status */
    ...

    /* Initialize spline parameters */
    s_order = DF_PP_LINEAR;    /* Spline is of the second order. */
    s_type = DF_PP_DEFAULT;    /* Spline is of the default type. */

    /* Define internal conditions for linear spline construction (none in this example) */
    ic_type = DF_NO_IC;
    ic = NULL;

    /* Define boundary conditions for linear spline construction (none in this example) */
    bc_type = DF_NO_BC;
    bc = NULL;
    scoeffhint = DF_NO_HINT;    /* No additional information about the spline. */

    /* Set spline parameters  in the Data Fitting task */
    status = dfdEditPPSpline1D( task, s_order, s_type, bc_type, bc, ic_type,
                                ic, scoeff, scoeffhint );
    /* Check the Data Fitting operation status */
    ...

    /* Use a standard computation method to construct a linear spline: */
    /* $P_i(x) = c_{1,i}+c_{2,i}(x-x_i)$, i=0,..., N-2 */
    /* The library packs spline coefficients to array scoeff. */
    /* scoeff[2*i+0]=$c_{1,i}$ and scoeff[2*i+1]=$c_{2,i}$, i=0,..., N-2 */
    status = dfdConstruct1D( task, DF_PP_SPLINE, DF_METHOD_STD );

    /* Check the Data Fitting operation status */
    ...

    /* Process spline coefficients */
    ...

    /* Deallocate Data Fitting task resources */
    status = dfDeleteTask( &task ) ;

    /* Check the Data Fitting operation status */
    ...
    return 0 ;
}
```

The following C example demonstrates cubic spline-based interpolation using Data Fitting routines. In this example, a scalar function defined on non-uniform partition is approximated by Bessel cubic spline using not-a-knot boundary conditions. Once the spline is constructed, you can use the spline to compute spline values at the given sites. Computation results are packed by the Data Fitting routine in row-major format.

C Example of Cubic Spline-Based Interpolation

```
#include "mkl.h"

#define NX 100                     /* Size of partition, number of breakpoints */
#define NSITE 1000                 /* Number of interpolation sites */
#define SPLINE_ORDER DF_PP_CUBIC   /* A cubic spline to construct */

int main()
{
    int status;          /* Status of a Data Fitting operation */
    DFTaskPtr task;      /* Data Fitting operations are task based */

    /* Parameters describing the partition */
    MKL_INT nx;          /* The size of partition x */
    double x[N];         /* Partition x */
    MKL_INT xhint;       /* Additional information about the structure of breakpoints */

    /* Parameters describing the function */
    MKL_INT ny;          /* Function dimension */
    double y[N];         /* Function values at the breakpoints */
    MKL_INT yhint;       /* Additional information about the function */

    /* Parameters describing the spline */
    MKL_INT  s_order;    /* Spline order */
    MKL_INT  s_type;     /* Spline type */
    MKL_INT  ic_type;    /* Type of internal conditions */
    MKL_INT* ic;         /* Array of internal conditions */
    MKL_INT  bc_type;    /* Type of boundary conditions */
    MKL_INT* bc;         /* Array of boundary conditions */

    double scoeff[(N-1)* ORDER];   /* Array of spline coefficients */
    MKL_INT scoeffhint;            /* Additional information about the coefficients */

    /* Parameters describing interpolation computations */
    MKL_INT nsite;        /* Number of interpolation sites */
    double site[NSITE];   /* Array of interpolation sites */
    MKL_INT sitehint;     /* Additional information about the structure of
                             interpolation sites */

    MKL_INT ndorder, dorder;    /* Parameters defining the type of interpolation */

    double* datahint;   /* Additional information on partition and interpolation sites */

    double r[NSITE];    /* Array of interpolation results */
    MKL_INT* rhint;     /* Additional information on the structure of the results */
    MKL_INT* cell;      /* Array of cell indices */

    /* Initialize the partition */
    nx = N;

    /* Set values of partition x */
    ...
    xhint = DF_NON_UNIFORM_PARTITION;  /* The partition is non-uniform. */

    /* Initialize the function */
     ny = 1;               /* The function is scalar. */

    /* Set function values */
    ...
    yhint = DF_NO_HINT;    /* No additional information about the function is provided. */

    /* Create a Data Fitting task */
    status = dfdNewTask1D( &task, nx, x, xhint, ny, y, yhint );

    /* Check the Data Fitting operation status */
    ...

    /* Initialize spline parameters */
    s_order = DF_PP_CUBIC;     /* Spline is of the fourth order (cubic spline). */
    s_type = DF_PP_BESSEL;     /* Spline is of the Bessel cubic type. */

    /* Define internal conditions for cubic spline construction (none in this example) */
    ic_type = DF_NO_IC;
    ic = NULL;

    /* Use not-a-knot boundary conditions. In this case, the is first and the last
     interior breakpoints are inactive, no additional values are provided. */
    bc_type = DF_BC_NOT_A_KNOT;
    bc = NULL;
    scoeffhint = DF_NO_HINT;    /* No additional information about the spline. */

    /* Set spline parameters  in the Data Fitting task */
    status = dfdEditPPSpline1D( task, s_order, s_type, bc_type, bc, ic_type,
                                ic, scoeff, scoeffhint );

    /* Check the Data Fitting operation status */
    ...

    /* Use a standard method to construct a cubic Bessel spline: */
    /* $P_i(x) = c_{1,i} + c_{2,i}(x - x_i) + c_{3,i}(x - x_i)^2 + c_{4,i}(x - x_i)^3$, */
    /* The library packs spline coefficients to array scoeff: */
    /* scoeff[4*i+0] = $c_{1,i}$, scoef[4*i+1] = $c_{2,i}$,          */
    /* scoeff[4*i+2] = $c_{3,i}$, scoef[4*i+1] = $c_{4,i}$,          */
    /* i=0,...,N-2  */
    status = dfdConstruct1D( task, DF_PP_SPLINE, DF_METHOD_STD );

    /* Check the Data Fitting operation status */
    ...

    /* Initialize interpolation parameters */
    nsite = NSITE;

    /* Set site values */
    ...

    sitehint = DF_NON_UNIFORM_PARTITION; /* Partition of sites is non-uniform */

    /* Request to compute spline values */
    ndorder = 1;
    dorder = 1;
    datahint = DF_NO_APRIORI_INFO;  /* No additional information about breakpoints or
                                       sites is provided. */
    rhint = DF_MATRIX_STORAGE_ROWS; /* The library packs interpolation results
                                       in row-major format. */
    cell = NULL;                    /* Cell indices are not required. */

    /* Solve interpolation problem using the default method: compute the sline values
       at the points site(i), i=0,..., nsite-1 and place the results to array r */
    status = dfdInterpolate1D( task, DF_INTERP, DF_METHOD_STD, nsite, site,
    sitehint, ndorder, &dorder, datahint, r, rhint, cell );

    /* Check Data Fitting operation status */
     ...

    /* De-allocate Data Fitting task resources */
     status = dfDeleteTask( &task );
    /* Check Data Fitting operation status */
     ...
    return 0;
}
```

The following C example demonstrates how to compute indices of cells containing given sites. This example uses uniform partition presented with two boundary points. The sites are in the ascending order.

C Example of Cell Search

```
#include "mkl.h"

#define NX 100                      /* Size of partition, number of breakpoints */
#define NSITE 1000                  /* Number of interpolation sites */

int main()
{
    int status;          /* Status of a Data Fitting operation */
    DFTaskPtr task;      /* Data Fitting operations are task based */

    /* Parameters describing the partition */
    MKL_INT nx;          /* The size of partition x */
    double x[2];         /* Partition x is uniform and holds endpoints
                            of interpolation interval [a, b] */
    MKL_INT xhint;       /* Additional information about the structure of breakpoints */
    /* Parameters describing the function */
    MKL_INT ny;          /* Function dimension */
    float   *y;          /* Function values at the breakpoints */
    MKL_INT yhint;       /* Additional information about the function */

    /* Parameters describing cell search */
    MKL_INT nsite;       /* Number of interpolation sites */
    double  site[NSITE]; /* Array of interpolation sites  */
    MKL_INT sitehint;    /* Additional information about the structure of sites */

    float* datahint;     /* Additional information on partition and interpolation sites */

    MKL_INT cell[NSITE]; /* Array for cell indices */

    /* Initialize a uniform partition */
    nx = N;
    /* Set values of partition x: for uniform partition,           */
    /* provide end-points of the interpolation interval [-1.0,1.0] */
    x[0] = -1.0f; x[1] = 1.0f;
    xhint = DF_UNIFORM_PARTITION; /* Partition is uniform */

    /* Initialize function parameters */
    /* In cell search, function values are not necessary and are set to zero/NULL values */
    ny    = 0;
    y     = NULL;
    yhint = DF_NO_HINT;

    /* Create a Data Fitting task */
    status = dfdNewTask1D( &task, nx, x, xhint, ny, y, yhint );

    /* Check Data Fitting operation status */
    ...

    /* Initialize interpolation (cell search) parameters */
    nsite = NSITE;

   /* Set sites in the ascending order */
    ...
    sitehint = DF_SORTED_DATA;      /* Sites are provided in the ascending order. */
    datahint = DF_NO_APRIORI_INFO;  /* No additional information
                                       about breakpoints/sites is provided.*/

    /* Use a standard method to compute indices of the cells that contain
       interpolation sites. The library places the index of the cell containing
       site(i) to the cell(i), i=0,...,nsite-1 */
    status = dfsSearchCells1D( task, DF_METHOD_STD, nsite, site, sitehint,
                            datahint, cell );
    /* Check Data Fitting operation status */
     ...

    /* Process cell indices */
     ...

    /* Deallocate Data Fitting task resources */
    status = dfDeleteTask( &task );

    /* Check Data Fitting operation status */
     ...
    return 0;
}
```

Task Status and Error Reporting

The Data Fitting routines report a task status through integer values. Negative status values indicate errors, while positive values indicate warnings. An error can be caused by invalid parameter values or a memory allocation failure.

The status codes have symbolic names predefined in the respective header files. For the C/C++ interface, these names are defined as macros via the `#define` statements. For the Fortran interface, the names are defined as integer constants via the `PARAMETER` operators.

If no error occurred, the function returns the `DF_STATUS_OK` code defined as zero:

`C/C++:`	`#define DF_STATUS_OK 0`
`F90/F95:`	`INTEGER, PARAMETER::DF_STATUS_OK = 0`

In case of an error, the function returns a non-zero error code that specifies the origin of the failure. Header files for both C/C++ and Fortran languages define the following status codes:

Status Codes in the Data Fitting Component

Status Code	Description
Common Status Codes	
`DF_STATUS_OK`	Operation completed successfully.
`DF_ERROR_NULL_TASK`	Data Fitting task is a `NULL` pointer.
`DF_ERROR_MEM_FAILURE`	Memory allocation failure.
`DF_ERROR_METHOD_NOT_SUPPORTED`	Requested method is not supported.
`DF_ERROR_COMP_TYPE_NOT_SUPPORTED`	Requested computation type is not supported.
Data Fitting Task Creation and Initialization, and Generic Editing Operations	
`DF_ERROR_BAD_NX`	Invalid number of breakpoints.
`DF_ERROR_BAD_X`	Array of breakpoints is invalid.
`DF_ERROR_BAD_X_HINT`	Invalid hint describing the structure of the partition.
`DF_ERROR_BAD_NY`	Invalid dimension of vector-valued function y.
`DF_ERROR_BAD_Y`	Array of function values is invalid.
`DF_ERROR_BAD_Y_HINT`	Invalid flag describing the structure of function y
Data Fitting Task-Specific Editing Operations	
`DF_ERROR_BAD_SPLINE_ORDER`	Invalid spline order.
`DF_ERROR_BAD_SPLINE_TYPE`	Invalid spline type.
`DF_ERROR_BAD_IC_TYPE`	Type of internal conditions used for spline construction is invalid.
`DF_ERROR_BAD_IC`	Array of internal conditions for spline construction is not defined.
`DF_ERROR_BAD_BC_TYPE`	Type of boundary conditions used in spline construction is invalid.
`DF_ERROR_BAD_BC`	Array of boundary conditions for spline construction is not defined.

Status Code	Description
DF_ERROR_BAD_PP_COEFF	Array of piecewise polynomial spline coefficients is not defined.
DF_ERROR_BAD_PP_COEFF_HINT	Invalid flag describing the structure of the piecewise polynomial spline coefficients.
DF_ERROR_BAD_PERIODIC_VAL	Function values at the endpoints of the interpolation interval are not equal as required in periodic boundary conditions.
DF_ERROR_BAD_DATA_ATTR	Invalid attribute of the pointer to be set or modified in Data Fitting task descriptor with the `df?editidxptr` task editor.
DF_ERROR_BAD_DATA_IDX	Index of the pointer to be set or modified in the Data Fitting task descriptor with the `df?editidxptr` task editor is out of the pre-defined range.
Data Fitting Computation Operations	
DF_ERROR_BAD_NSITE	Invalid number of interpolation sites.
DF_ERROR_BAD_SITE	Array of interpolation sites is not defined.
DF_ERROR_BAD_SITE_HINT	Invalid flag describing the structure of interpolation sites.
DF_ERROR_BAD_NDORDER	Invalid size of the array defining derivative orders to be computed at interpolation sites.
DF_ERROR_BAD_DORDER	Array defining derivative orders to be computed at interpolation sites is not defined.
DF_ERROR_BAD_DATA_HINT	Invalid flag providing additional information about partition or interpolation sites.
DF_ERROR_BAD_INTERP	Array of spline-based interpolation results is not defined.
DF_ERROR_BAD_INTERP_HINT	Invalid flag defining the structure of spline-based interpolation results.
DF_ERROR_BAD_CELL_IDX	Array of indices of partition cells containing interpolation sites is not defined.
DF_ERROR_BAD_NLIM	Invalid size of arrays containing integration limits.
DF_ERROR_BAD_LLIM	Array of the left-side integration limits is not defined.
DF_ERROR_BAD_RLIM	Array of the right-side integration limits is not defined.
DF_ERROR_BAD_INTEGR	Array of spline-based integration results is not defined.
DF_ERROR_BAD_INTEGR_HINT	Invalid flag providing the structure of the array of spline-based integration results.
DF_ERROR_BAD_LOOKUP_INTERP_SITE	Bad site provided for interpolation with look-up interpolator.

NOTE
The routine that estimates piecewise polynomial cubic spline coefficients can return internal error codes related to the specifics of the implementation. Such error codes indicate invalid input data or other issues unrelated to Data Fitting routines.

Task Creation and Initialization Routines

Task creation and initialization routines are functions used to create a new task descriptor and initialize its parameters. The Data Fitting component provides the `df?newtask1d` routine that creates and initializes a new task descriptor for a one-dimensional Data Fitting task.

df?newtask1d

Creates and initializes a new task descriptor for a one-dimensional Data Fitting task.

Syntax

```
status = dfsnewtask1d(task, nx, x, xhint, ny, y, yhint)

status = dfdnewtask1d(task, nx, x, xhint, ny, y, yhint)

status = dfsNewTask1D(&task, nx, x, xhint, ny, y, yhint)

status = dfdNewTask1D(&task, nx, x, xhint, ny, y, yhint)
```

Include Files

- Fortran: `mkl_df.f90`
- C: `mkl.h`

Input Parameters

Name	Type	Description
nx	**Fortran:** `INTEGER` **C:** `const MKL_INT`	Number of breakpoints representing partition of interpolation interval [`a`, `b`].

x	**Fortran:** `REAL(KIND=4) DIMENSION(*)` for `dfsnewtask1d`
	`REAL(KIND=8) DIMENSION(*)` for `dfdnewtask1d`
	C: `const float*` for `dfsNewTask1D`
	`const double*` for `dfdNewTask1D`
		One-dimensional array containing the sorted breakpoints from interpolation interval [`a`, `b`]. The structure of the array is defined by parameter *xhint*:
		• If partition is non-uniform or quasi-uniform, the array should contain *nx* ordered values.
		• If partition is uniform, the array should contain two entries that represent endpoints of interpolation interval [`a`, `b`].

xhint	**Fortran:** `INTEGER` **C:** `const MKL_INT`	A flag describing the structure of partition *x*. For the list of possible values of *xhint*, see table "Hint Values for Partition *x*". If you set the flag to the `DF_NO_HINT` value, the library interprets the partition as non-uniform.
ny	**Fortran:** `INTEGER`	Dimension of vector-valued function *y*.

Name	Type	Description
	C: `const MKL_INT`	

y	**Fortran:** `REAL(KIND=4) DIMENSION(*)` for `dfsnewtask1d`
	`REAL(KIND=8) DIMENSION(*)` for `dfdnewtask1d`
	C: `const float*` for `dfsNewTask`
	`const double*` for `dfdNewTask`
		Vector-valued function *y*, array of size *nx***ny*.
		The storage format of function values in the array is defined by the value of flag *yhint*.

yhint	**Fortran:** `INTEGER` **C:** `const MKL_INT`	A flag describing the structure of array *y*. Valid hint values are listed in table "Hint Values for Vector-Valued Function *y*". If you set the flag to the `DF_NO_HINT` value, the library assumes that all *ny* coordinates of the vector-valued function *y* are provided and stored in row-major format.

Output Parameters

Name	Type	Description
task	**Fortran:** `TYPE(DF_TASK)` **C:** `DFTaskPtr`	Descriptor of the task.
status	**Fortran:** `INTEGER` **C:** `int`	Status of the routine: • `DF_STATUS_OK` if the task is created successfully. • Non-zero error code if the task creation failed. See "Task Status and Error Reporting" for error code definitions.

Description

The `df?newtask1d` routine creates and initializes a new Data Fitting task descriptor with user-specified parameters for a one-dimensional Data Fitting task. The *x* and *nx* parameters representing the partition of interpolation interval [*a*, *b*] are mandatory. If you provide invalid values for these parameters, such as a `NULL` pointer *x* or the number of breakpoints smaller than two, the routine does not create the Data Fitting task and returns an error code.

If you provide a vector-valued function *y*, make sure that the function dimension *ny* and the array of function values *y* are both valid. If any of these parameters are invalid, the routine does not create the Data Fitting task and returns an error code.

If you store coordinates of the vector-valued function *y* in non-contiguous memory locations, you can set the *yhint* flag to `DF_1ST_COORDINATE`, and pass only the first coordinate of the function into the task creation routine. After successful creation of the Data Fitting task, you can pass the remaining coordinates using the `df?editidxptr` task editor.

If the routine fails to create the task descriptor, it returns a `NULL` task pointer.

The routine supports the following hint values for partition *x*:

Hint Values for Partition x

Value	Description
`DF_NON_UNIFORM_PARTITION`	Partition is non-uniform.
`DF_QUASI_UNIFORM_PARTITION`	Partition is quasi-uniform.
`DF_UNIFORM_PARTITION`	Partition is uniform.
`DF_NO_HINT`	No hint is provided. By default, partition is interpreted as non-uniform.

The routine supports the following hint values for the vector-valued function:

Hint Values for Vector-Valued Function y

Value	Description
`DF_MATRIX_STORAGE_ROWS`	Data is stored in row-major format according to C conventions.
`DF_MATRIX_STORAGE_COLS`	Data is stored in column-major format according to Fortran conventions.
`DF_1ST_COORDINATE`	The first coordinate of vector-valued data is provided.
`DF_NO_HINT`	No hint is provided. By default, the coordinates of vector-valued function y are provided and stored in row-major format.

NOTE
You must preserve the arrays x (breakpoints) and y (vector-valued functions) through the entire workflow of the Data Fitting computations for a task, as the task stores the addresses of the arrays for spline-based computations.

Task Editors

Task editors initialize or change the predefined Data Fitting task parameters. You can use task editors to initialize or modify pointers to arrays or parameter values.

Task editors can be task-specific and generic. Task-specific editors can modify more than one parameter related to a specific task. Generic editors modify a single parameter at a time.

The Data Fitting component of the Intel MKL provides the following task editors:

Data Fitting Task Editors

Editor	Description	Type
`df?editppspline1d`	Changes parameters of the piecewise polynomial spline.	Task-specific
`df?editptr`	Changes a pointer in the task descriptor.	Generic
`dfieditval`	Changes a value in the task descriptor.	Generic
`df?editidxptr`	Changes a coordinate of data represented in matrix format. For example, a vector-valued function or spline coefficients.	Generic

df?editppspline1d

Modifies parameters representing a spline in a Data Fitting task descriptor.

Syntax

status = dfseditppspline1d(*task*, *s_order*, *s_type*, *bc_type*, *bc*, *ic_type*, *ic*, *scoeff*, *scoeffhint*)

status = dfdeditppspline1d(*task*, *s_order*, *s_type*, *bc_type*, *bc*, *ic_type*, *ic*, *scoeff*, *scoeffhint*)

status = dfsEditPPSpline1D(*task*, *s_order*, *s_type*, *bc_type*, *bc*, *ic_type*, *ic*, *scoeff*, *scoeffhint*)

status = dfdEditPPSpline1D(*task*, *s_order*, *s_type*, *bc_type*, *bc*, *ic_type*, *ic*, *scoeff*, *scoeffhint*)

Include Files

- Fortran: `mkl_df.f90`
- C: `mkl.h`

Input Parameters

Name	Type	Description
task	**Fortran:** `TYPE(DF_TASK)` **C:** `DFTaskPtr`	Descriptor of the task.
s_order	**Fortran:** `INTEGER` **C:** `const MKL_INT`	Spline order. The parameter takes one of the values described in table "Spline Orders Supported by Data Fitting Functions".
s_type	**Fortran:** `INTEGER` **C:** `const MKL_INT`	Spline type. The parameter takes one of the values described in table "Spline Types Supported by Data Fitting Functions".
bc_type	**Fortran:** `INTEGER` **C:** `const MKL_INT`	Type of boundary conditions. The parameter takes one of the values described in table "Boundary Conditions Supported by Data Fitting Functions".

bc	**Fortran:** `REAL(KIND=4) DIMENSION(*)` for `dfseditppspline1d`
	`REAL(KIND=8) DIMENSION(*)` for `dfdeditppspline1d`
	C: `const float*` for `dfsEditPPSpline1D`
	`const double*` for `dfdEditPPSpline1D`
		Pointer to boundary conditions. The size of the array is defined by the value of parameter *bc_type*:
		• If you set free-end or not-a-knot boundary conditions, pass the `NULL` pointer to this parameter.
		• If you combine boundary conditions at the endpoints of the interpolation interval, pass an array of two elements.
		• If you set a boundary condition for the default quadratic spline or a periodic condition for Hermite or the default cubic spline, pass an array of one element.

ic_type	**Fortran:** `INTEGER` **C:** `const MKL_INT`	Type of internal conditions. The parameter takes one of the values described in table "Internal Conditions Supported by Data Fitting Functions".

Name	Type	Description
ic	**Fortran:** REAL(KIND=4) DIMENSION(*) for dfseditppspline1d REAL(KIND=8) DIMENSION(*) for dfdeditppspline1d **C:** const float* for dfsEditPPSpline1D const double* for dfdEditPPSpline1D	A non-NULL pointer to the array of internal conditions. The size of the array is defined by the value of parameter *ic_type*: • If you set first derivatives or second derivatives internal conditions (*ic_type*=DF_IC_1ST_DER or *ic_type*=DF_IC_2ND_DER), pass an array of n-1 derivative values at the internal points of the interpolation interval. • If you set the knot values internal condition for Subbotin spline (*ic_type*=DF_IC_Q_KNOT) and the knot partition is non-uniform, pass an array of n+1 elements. • If you set the knot values internal condition for Subbotin spline (*ic_type*=DF_IC_Q_KNOT) and the knot partition is uniform, pass an array of four elements.
scoeff	**Fortran:** REAL(KIND=4) DIMENSION(*) for dfseditppspline1d REAL(KIND=8) DIMENSION(*) for dfdeditppspline1d **C:** const float* for dfsEditPPSpline1D const double* for dfdEditPPSpline1D	Spline coefficients. An array of size *s_order**(*nx*-1). The storage format of the coefficients in the array is defined by the value of flag *scoeffhint*.
scoeffhint	**Fortran:** INTEGER **C:** const MKL_INT	A flag describing the structure of the array of spline coefficients. For valid hint values, see table "Hint Values for Spline Coefficients". The library stores the coefficients in row-major format. The default value is DF_NO_HINT.

Output Parameters

Name	Type	Description
status	**Fortran:** INTEGER **C:** int	Status of the routine: • DF_STATUS_OK if the routine execution completed successfully. • Non-zero error code if the routine execution failed. See "Task Status and Error Reporting" for error code definitions.

Description

The editor modifies parameters that describe the order, type, boundary conditions, internal conditions, and coefficients of a spline. The spline order definition is provided in the "Mathematical Conventions" section. You can set the spline order to any value supported by Data Fitting functions. The table below lists the available values:

Spline Orders Supported by the Data Fitting Functions

Order	Description
`DF_PP_STD`	Artificial value. Use this value for look-up and step-wise constant interpolants only.
`DF_PP_LINEAR`	Piecewise polynomial spline of the second order (linear spline).
`DF_PP_QUADRATIC`	Piecewise polynomial spline of the third order (quadratic spline).
`DF_PP_CUBIC`	Piecewise polynomial spline of the fourth order (cubic spline).

To perform computations with a spline not supported by Data Fitting routines, set the parameter defining the spline order and pass the spline coefficients to the library in the supported format. For format description, see figure "Row-major Coefficient Storage Format".

The table below lists the supported spline types:

Spline Types Supported by Data Fitting Functions

Type	Description
`DF_PP_DEFAULT`	The default spline type. You can use this type with linear, quadratic, or user-defined splines.
`DF_PP_SUBBOTIN`	Quadratic splines based on Subbotin algorithm, [StechSub76].
`DF_PP_NATURAL`	Natural cubic spline.
`DF_PP_HERMITE`	Hermite cubic spline.
`DF_PP_BESSEL`	Bessel cubic spline.
`DF_PP_AKIMA`	Akima cubic spline.
`DF_LOOKUP_INTERPOLANT`	Look-up interpolant.
`DF_CR_STEPWISE_CONST_INTERPOLANT`	Continuous right step-wise constant interpolant.
`DF_CL_STEPWISE_CONST_INTERPOLANT`	Continuous left step-wise constant interpolant.

If you perform computations with look-up or step-wise constant interpolants, set the spline order to the `DF_PP_STD` value.

Construction of specific splines may require boundary or internal conditions. To compute coefficients of such splines, you should pass boundary or internal conditions to the library by specifying the type of the conditions and providing the necessary values. For splines that do not require additional conditions, such as linear splines, set condition types to `DF_NO_BC` and `DF_NO_IC`, and pass `NULL` pointers to the conditions. The table below defines the supported boundary conditions:

Boundary Conditions Supported by Data Fitting Functions

Boundary Condition	Description	Spline
`DF_NO_BC`	No boundary conditions provided.	All
`DF_BC_NOT_A_KNOT`	Not-a-knot boundary conditions.	Akima, Bessel, Hermite, natural cubic

Boundary Condition	**Description**	**Spline**
`DF_BC_FREE_END`	Free-end boundary conditions.	Akima, Bessel, Hermite, natural cubic, quadratic Subbotin
`DF_BC_1ST_LEFT_DER`	The first derivative at the left endpoint.	Akima, Bessel, Hermite, natural cubic, quadratic Subbotin
`DF_BC_1ST_RIGHT_DER`	The first derivative at the right endpoint.	Akima, Bessel, Hermite, natural cubic, quadratic Subbotin
`DF_BC_2ST_LEFT_DER`	The second derivative at the left endpoint.	Akima, Bessel, Hermite, natural cubic, quadratic Subbotin
`DF_BC_2ND_RIGHT_DER`	The second derivative at the right endpoint.	Akima, Bessel, Hermite, natural cubic, quadratic Subbotin
`DF_BC_PERIODIC`	Periodic boundary conditions.	Linear, all cubic splines
`DF_BC_Q_VAL`	Function value at point $(x_0 + x_1)/2$	Default quadratic

NOTE
To construct a natural cubic spline, pass these settings to the editor:

- `DF_PP_CUBIC` as the spline order,
- `DF_PP_NATURAL` as the spline type, and
- `DF_BC_FREE_END` as the boundary condition.

To construct a cubic spline with other boundary conditions, pass these settings to the editor:

- `DF_PP_CUBIC` as the spline order,
- `DF_PP_NATURAL` as the spline type, and
- the required type of boundary condition.

For Akima, Hermite, Bessel and default cubic splines use the corresponding type defined in Table Spline Types Supported by Data Fitting Functions.

You can combine the values of boundary conditions with a bitwise `OR` operation. This permits you to pass combinations of first and second derivatives at the endpoints of the interpolation interval into the library. To pass a first derivative at the left endpoint and a second derivative at the right endpoint, set the boundary conditions to `DF_BC_1ST_LEFT_DER OR DF_BC_2ND_RIGHT_DER`.

You should pass the combined boundary conditions as an array of two elements. The first entry of the array contains the value of the boundary condition for the left endpoint of the interpolation interval, and the second entry - for the right endpoint. Pass other boundary conditions as arrays of one element.

For the conditions defined as a combination of valid values, the library applies the following rules to identify the boundary condition type:

- If not required for spline construction, the value of boundary conditions is ignored.
- Not-a-knot condition has the highest priority. If set, other boundary conditions are ignored.
- Free-end condition has the second priority after the not-a-knot condition. If set, other boundary conditions are ignored.
- Periodic boundary condition has the next priority after the free-end condition.
- The first derivative has higher priority than the second derivative at the right and left endpoints.

If you set the periodic boundary condition, make sure that function values at the endpoints of the interpolation interval are identical. Otherwise, the library returns an error code. The table below specifies the values to be provided for each type of spline if the periodic boundary condition is set.

Boundary Requirements for Periodic Conditions

Spline Type	Periodic Boundary Condition Support	Boundary Value
Linear	Yes	Not required
Default quadratic	No	
Subbotin quadratic	No	
Natural cubic	Yes	Not required
Bessel	Yes	Not required
Akima	Yes	Not required
Hermite cubic	Yes	First derivative
Default cubic	Yes	Second derivative

Internal conditions supported in the Data Fitting domain that you can use for the *`ic_type`* parameter are the following:

Internal Conditions Supported by Data Fitting Functions

Internal Condition	Description	Spline
`DF_NO_IC`	No internal conditions provided.	
`DF_IC_1ST_DER`	Array of first derivatives of size `n`-2, where `n` is the number of breakpoints. Derivatives are applicable to each coordinate of the vector-valued function.	Hermite cubic
`DF_IC_2ND_DER`	Array of second derivatives of size `n`-2, where `n` is the number of breakpoints. Derivatives are applicable to each coordinate of the vector-valued function.	Default cubic
`DF_IC_Q_KNOT`	Knot array of size `n`+1, where `n` is the number of breakpoints.	Subbotin quadratic

To construct a Subbotin quadratic spline, you have three options to get the array of knots in the library:

- If you do not provide the knots, the library uses the default values of knots $t = \{t_i\}$, $i = 0, \ldots, n$ according to the rule:

  $t_0 = x_0$, $t_n = x_{n-1}$, $t_i = (x_i + x_{i-1})/2$, $i = 1, \ldots, n - 1$.
- If you provide the knots in an array of size $n + 1$, the knots form a non-uniform partition. Make sure that the knot values you provide meet the following conditions:

  $t_0 = x_0$, $t_n = x_{n-1}$, $t_i \in (x_{i-1}, x_i)$, $i = 1, \ldots, n - 1$.
- If you provide the knots in an array of size 4, the knots form a uniform partition

  $t_0 = x_0$, $t_1 = l$, $t_2 = r$, $t_3 = x_{n-1}$, where $l \in (x_0, x_1)$ and $r \in (x_{n-2}, x_{n-1})$.

  In this case, you need to set the value of the *`ic_type`* parameter holding the type of internal conditions to `DF_IC_Q_KNOT OR DF_UNIFORM_PARTITION`.

NOTE
Since the partition is uniform, perform an `OR` operation with the `DF_UNIFORM_PARTITION` partition hint value described in Table Hint Values for Partition x.

For computations based on look-up and step-wise constant interpolants, you can avoid calling the `df?editppspline1d` editor and directly call one of the routines for spline-based computation of spline values, derivatives, or integrals. For example, you can call the `df?construct1d` routine to construct the required spline with the given attributes, such as order or type.

The memory location of the spline coefficients is defined by the *scoeff* parameter. Make sure that the size of the array is sufficient to hold *s_order* * (*nx*-1) values.

The `df?editppspline1d` routine supports the following hint values for spline coefficients:

Hint Values for Spline Coefficients

Order	Description
`DF_1ST_COORDINATE`	The first coordinate of vector-valued data is provided.
`DF_NO_HINT`	No hint is provided. By default, all sets of spline coefficients are stored in row-major format.

The coefficients for all coordinates of the vector-valued function are packed in memory one by one in successive order, from function y_1 to function y_{ny}.

Within each coordinate, the library stores the coefficients as an array, in row-major format:

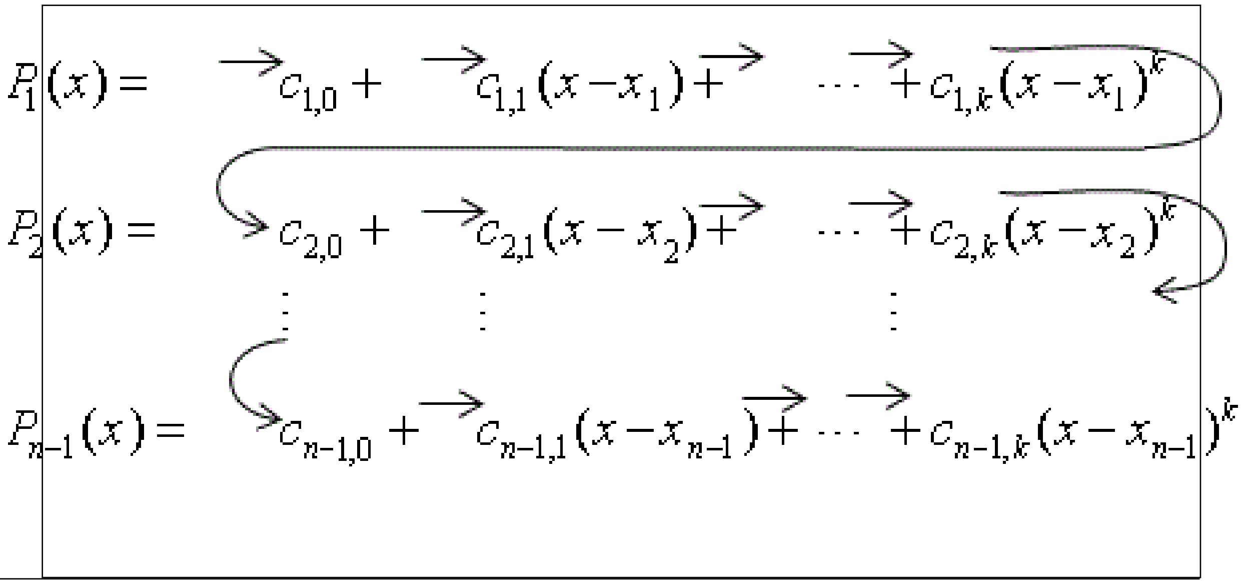
$c_{1,0}, c_{1,1}, \ldots, c_{1,k}, c_{2,0}, c_{2,1}, \ldots, c_{2,k}, \ldots, c_{n-1,0}, c_{n-1,1}, \ldots, c_{n-1,k}$

Mapping of the coefficients to storage in the *scoeff* array is described below, where $c_{i,j}$ is the *j*th coefficient of the function

$$P_i(x) = c_{i,0} + c_{i,1}(x - x_i) + \ldots + c_{i,k}(x - x_i)^k.$$

See Mathematical Conventions for more details on nomenclature and interpolants.

Row-major Coefficient Storage Format



If you store splines corresponding to different coordinates of the vector-valued function at non-contiguous memory locations, do the following:

1. Set the `scoeffhint` flag to `DF_1ST_COORDINATE` and provide the spline for the first coordinate.
2. Pass the spline coefficients for the remaining coordinates into the Data Fitting task using the `df?editidxptr` task editor.

Using the `df?editppspline1d` task editor, you can provide to the Data Fitting task an already constructed spline that you want to use in computations. To ensure correct interpretation of the memory content, you should set the following parameters:

- Spline order and type, if appropriate. If the spline is not supported by the library, set the *`s_type`* parameter to `DF_PP_DEFAULT`.
- Pointer to the array of spline coefficients in row-major format.
- The *`scoeffhint`* parameter describing the structure of the array:
    - Set the *`scoeffhint`* flag to the `DF_1ST_COORDINATE` value to pass spline coefficients stored at different memory locations. In this case, you can set the parameters that describe boundary and internal conditions to zero.
    - Use the default value `DF_NO_HINT` for all other cases.

Before passing an already constructed spline into the library, you should call the `dfieditval` task editor to provide the dimension of the spline `DF_NY`. See table "Parameters Supported by the `dfieditval` Task Editor" for details.

After you provide the spline to the Data Fitting task, you can run computations that use this spline.

NOTE
You must preserve the arrays *`bc`* (boundary conditions), *`ic`* (internal conditions), and *`scoeff`* (spline coefficients) through the entire workflow of the Data Fitting computations for a task, as the task stores the addresses of the arrays for spline-based computations.

df?editptr

Modifies a pointer to an array held in a Data Fitting task descriptor.

Syntax

```
status = dfseditptr(task, ptr_type, ptr)

status = dfdeditptr(task, ptr_type, ptr)

status = dfsEditPtr(task, ptr_type, ptr)

status = dfdEditPtr(task, ptr_type, ptr)
```

Include Files

- Fortran: `mkl_df.f90`
- C: `mkl.h`

Input Parameters

Name	Type	Description
`task`	**Fortran:** `TYPE(DF_TASK)`	Descriptor of the task.

Name	Type	Description
	C: `DFTaskPtr`	
ptr_type	**Fortran:** `INTEGER` **C:** `const MKL_INT`	The parameter to change. For details, see the *Pointer Type* column in table "Pointers Supported by the `df?editptr` Task Editor".

ptr	**Fortran:** `REAL(KIND=4) DIMENSION(*)` for `dfseditptr`
	`REAL(KIND=8) DIMENSION(*)` for `dfdeditptr`
	C: `const float*` for `dfsEditPtr`
	`const double*` for `dfdEditPtr`
		New pointer. For details, see the *Purpose* column in table "Pointers Supported by the `df?editptr` Task Editor".

Output Parameters

Name	Type	Description
status	**Fortran:** `INTEGER` **C:** `int`	Status of the routine: • `DF_STATUS_OK` if the routine execution completed successfully. • Non-zero error code otherwise. See "Task Status and Error Reporting" for error code definitions.

Description

The `df?editptr` editor replaces the pointer of type *ptr_type* stored in a Data Fitting task descriptor with a new pointer *ptr*. The table below describes types of pointers supported by the editor:

Pointers Supported by the `df?editptr` Task Editor

Pointer Type	Purpose
`DF_X`	Partition x of the interpolation interval
`DF_Y`	Vector-valued function y
`DF_IC`	Internal conditions for spline construction. For details, see table "Internal Conditions Supported by Data Fitting Functions".
`DF_BC`	Boundary conditions for spline construction. For details, see table "Boundary Conditions Supported by Data Fitting Functions".
`DF_PP_SCOEFF`	Spline coefficients

You can use `df?editptr` to modify different types of pointers including pointers to the vector-valued function and spline coefficients stored in contiguous memory. Use the `df?editidxptr` editor if you need to modify pointers to coordinates of the vector-valued function or spline coefficients stored at non-contiguous memory locations.

If you modify a partition of the interpolation interval, then you should call the `dfieditval` task editor with the corresponding value of `DF_XHINT`, even if the structure of the partition remains the same.

If you pass a `NULL` pointer to the `df?editptr` task editor, the task remains unchanged and the routine returns an error code. For the predefined error codes, please see "Task Status and Error Reporting".

NOTE
You must preserve the arrays *x* (breakpoints), *y* (vector-valued functions), *bc* (boundary conditions), *ic* (internal conditions), and *scoeff* (spline coefficients) through the entire workflow of the Data Fitting computations which use those arrays, as the task stores the addresses of the arrays for spline-based computations.

dfieditval

Modifies a parameter value in a Data Fitting task descriptor.

Syntax

status = `dfieditval(`*task*, *val_type*, *val*`)`

status = `dfiEditVal(`*task*, *val_type*, *val*`)`

Include Files

- Fortran: `mkl_df.f90`
- C: `mkl.h`

Input Parameters

Name	Type	Description
task	**Fortran:** `TYPE(DF_TASK)` **C:** `DFTaskPtr`	Descriptor of the task.
val_type	**Fortran:** `INTEGER` **C:** `const MKL_INT`	The parameter to change. See table "Parameters Supported by the `dfieditval` Task Editor".
val	**Fortran:** `INTEGER` **C:** `const MKL_INT`	A new parameter value. See table "Parameters Supported by the `dfieditval` Task Editor".

Output Parameters

Name	Type	Description
status	**Fortran:** `INTEGER` **C:** `int`	Status of the routine: • `DF_STATUS_OK` if the routine execution completed successfully. • Non-zero error code otherwise. See "Task Status and Error Reporting" for error code definitions.

Description

The `dfieditval` task editor replaces the parameter of type *val_type* stored in a Data Fitting task descriptor with a new value *val*. The table below describes valid types of parameter *val_type* supported by the editor:

Parameters Supported by the dfieditval Task Editor

Parameter	Purpose
`DF_NX`	Number of breakpoints
`DF_XHINT`	A flag describing the structure of partition. See table "Hint Values for Partition x" for the list of available values.
`DF_NY`	Dimension of the vector-valued function
`DF_YHINT`	A flag describing the structure of the vector-valued function. See table "Hint Values for Vector Function y" for the list of available values.
`DF_SPLINE_ORDER`	Spline order. See table "Spline Orders Supported by Data Fitting Functions" for the list of available values.
`DF_SPLINE_TYPE`	Spline type. See table "Spline Types Supported by Data Fitting Functions" for the list of available values.
`DF_BC_TYPE`	Type of boundary conditions used in spline construction. See table "Boundary Conditions Supported by Data Fitting Functions" for the list of available values.
`DF_IC_TYPE`	Type of internal conditions used in spline construction. See table "Internal Conditions Supported by Data Fitting Functions" for the list of available values.
`DF_PP_COEFF_HINT`	A flag describing the structure of spline coefficients. See table "Hint Values for Spline Coefficients" for the list of available values.
`DF_CHECK_FLAG`	A flag which controls checking of Data Fitting parameters. See table "Possible Values for the `DF_CHECK_FLAG` Parameter" for the list of available values.

If you pass a zero value for the parameter describing the size of the arrays that hold coefficients for a partition, a vector-valued function, or a spline, the parameter held in the Data fitting task remains unchanged and the routine returns an error code. For the predefined error codes, see "Task Status and Error Reporting".

Possible Values for the `DF_CHECK_FLAG` Parameter

Value	Description
`DF_ENABLE_CHECK_FLAG`	Checks the correctness of parameters of Data Fitting computational routines (default mode).
`DF_DISABLE_CHECK_FLAG`	Disables checking of the correctness of parameters of Data Fitting computational routines.

Use `DF_CHECK_FLAG` for *`val_type`* in order to control validation of parameters of Data Fitting computational routines such as `Construct1d`, `Interpolate1D/InterpolateEx1d`, and `SearchCells1D/SearchCellsEx1D`, which can perform better with a small number of interpolation sites or integration limits (fewer than one dozen). The default mode, with checking of parameters enabled, should be used as you develop a Data Fitting-based application. After you complete development you can disable parameter checking in order to improve the performance of your application.

If you modify the parameter describing dimensions of the arrays that hold the vector-valued function or spline coefficients in contiguous memory, you should call the `df?editptr` task editor with the corresponding pointers to the vector-valued function or spline coefficients even when this pointer remains unchanged. Call the `df?editidxptr` editor if those arrays are stored in non-contiguous memory locations.

You must call the `dfieditval` task editor to edit the structure of the partition `DF_XHINT` every time you modify a partition using `df?editptr`, even if the structure of the partition remains the same.

df?editidxptr

Modifies a pointer to the memory representing a coordinate of the data stored in matrix format.

Syntax

```
status = dfseditidxptr(task, type, idx, ptr)
status = dfdeditidxptr(task, type, idx, ptr)
status = dfsEditIdxPtr(task, type, idx, ptr)
status = dfdEditIdxPtr(task, type, idx, ptr)
```

Include Files

- Fortran: `mkl_df.f90`
- C: `mkl.h`

Input Parameters

Name	Type	Description
task	**Fortran:** `TYPE(DF_TASK)` **C:** `DFTaskPtr`	Descriptor of the task.
type	**Fortran:** `INTEGER` **C:** `const MKL_INT`	Type of the data to be modified. The parameter takes one of the values described in "Data Attributes Supported by the `df?editidxptr` Task Editor".
idx	**Fortran:** `INTEGER` **C:** `const MKL_INT`	Index of the coordinate whose pointer is to be modified.

ptr	**Fortran:** `REAL(KIND=4) DIMENSION(*)` for `dfseditidxptr`
	`REAL(KIND=8) DIMENSION(*)` for `dfdeditidxptr`
	C: `const float*` for `dfsEditIdxPtr`
	`const double*` for `dfdEditIdxPtr`
		Pointer to the data that holds values of coordinate *idx*. For details, see table "Data Attributes Supported by the `df?editidxptr` Task Editor".

Output Parameters

Name	Type	Description
status	**Fortran:** `INTEGER` **C:** `int`	Status of the routine: • `DF_STATUS_OK` if the routine execution completed successfully.

Name	Type	Description
		• Non-zero error code otherwise. See "Task Status and Error Reporting" for error code definitions.

Description

The routine modifies a pointer to the array that holds the *idx* coordinate of vector-valued function *y* or the pointer to the array of spline coefficients corresponding to the given coordinate.

You can use the editor if you need to pass into a Data Fitting task or modify the pointer to coordinates of the vector-valued function or spline coefficients held at non-contiguous memory locations.

Before calling this editor, make sure that you have created and initialized the task using a task creation function or a relevant editor such as the generic or specific `df?editppspline1d` editor.

Data Attributes Supported by the df?editidxptr Task Editor

Data Attribute	Description
`DF_Y`	Vector-valued function *y*
`DF_PP_SCOEFF`	Piecewise polynomial spline coefficients

When using `df?editidxptr`, you might receive an error code in the following cases:

- You passed an unsupported parameter value into the editor.
- The value of the index exceeds the predefined value that equals the dimension *ny* of the vector-valued function.
- You pass a `NULL` pointer to the editor. In this case, the task remains unchanged.

The code example below demonstrates how to use the editor for providing values of a vector-valued function stored in two non-contiguous arrays:

```
#define NX 1000  /* number of break points       */
#define NY 2     /* dimension of vector-valued function */
int main()
{
    DFTaskPtr task;
    double x[NX];
    double y1[NX], y2[NX]; /* vector-valued function is stored as two arrays */
    /* Provide first coordinate of two-dimensional function y into creation routine */
    status = dfdNewTask1D( &task, NX, x, DF_NON_UNIFORM_PARTITION, NY, y1,
                           DF_1ST_COORDINATE );
    /* Provide second coordiante of two-dimensional function */
    status = dfdEditIdxPtr(task, DF_Y, 1, y2 );
    ...
}
```

Computational Routines

Data Fitting computational routines are functions used to perform spline-based computations, such as:

- spline construction
- computation of values, derivatives, and integrals of the predefined order
- cell search

Once you create a Data Fitting task and initialize the required parameters, you can call computational routines as many times as necessary.

The table below lists the available computational routines:

Data Fitting Computational Routines

Routine	Description
`df?construct1d`	Constructs a spline for a one-dimensional Data Fitting task.
`df?interpolate1d`	Computes spline values and derivatives.
`df?interpolateex1d`	Computes spline values and derivatives by calling user-provided interpolants.
`df?integrate1d`	Computes spline-based integrals.
`df?integrateex1d`	Computes spline-based integrals by calling user-provided integrators.
`df?searchcells1d`	Finds indices of cells containing interpolation sites.
`df?searchcellsex1d`	Finds indices of cells containing interpolation sites by calling user-provided cell searchers.

If a Data Fitting computation completes successfully, the computational routines return the `DF_STATUS_OK` code. If an error occurs, the routines return an error code specifying the origin of the failure. Some possible errors are the following:

- The task pointer is `NULL`.
- Memory allocation failed.
- The computation failed for another reason.

For the list of available status codes, see "Task Status and Error Reporting".

NOTE
Data Fitting computational routines do not control errors for floating-point conditions, such as overflow, gradual underflow, or operations with Not a Number (NaN) values.

df?construct1d

Constructs a spline of the given type.

Syntax

status = `dfsconstruct1d(`*task*, *s_format*, *method*`)`

status = `dfdconstruct1d(`*task*, *s_format*, *method*`)`

status = `dfsConstruct1D(`*task*, *s_format*, *method*`)`

status = `dfdConstruct1D(`*task*, *s_format*, *method*`)`

Include Files

- Fortran: `mkl_df.f90`
- C: `mkl.h`

Input Parameters

Name	Type	Description
task	**Fortran:** `TYPE(DF_TASK)` **C:** `DFTaskPtr`	Descriptor of the task.

Name	Type	Description
s_format	**Fortran:** INTEGER **C:** const MKL_INT	Spline format. The supported value is DF_PP_SPLINE.
method	**Fortran:** INTEGER **C:** const MKL_INT	Construction method. The supported value is DF_METHOD_STD.

Output Parameters

Name	Type	Description
status	**Fortran:** INTEGER **C:** int	Status of the routine: • DF_STATUS_OK if the routine execution completed successfully. • Non-zero error code if the routine execution failed. See "Task Status and Error Reporting" for error code definitions.

Description

Before calling df?construct1d, you need to create and initialize the task, and set the parameters representing the spline. Then you can call the df?construct1d routine to construct the spline. The format of the spline is defined by parameter *s_format*. The method for spline construction is defined by parameter *method*. Upon successful construction, the spline coefficients are available in the user-provided memory location in the format you set through the Data Fitting editor. For the available storage formats, see table "Hint Values for Spline Coefficients".

df?interpolate1d/df?interpolateex1d

Runs data fitting computations.

Syntax

status = dfsinterpolate1d(*task*, *type*, *method*, *nsite*, *site*, *sitehint*, *ndorder*, *dorder*, *datahint*, *r*, *rhint*, *cell*)

status = dfdinterpolate1d(*task*, *type*, *method*, *nsite*, *site*, *sitehint*, *ndorder*, *dorder*, *datahint*, *r*, *rhint*, *cell*)

status = dfsinterpolateex1d(*task*, *type*, *method*, *nsite*, *site*, *sitehint*, *ndorder*, *dorder*, *datahint*, *r*, *rhint*, *cell*, *le_cb*, *le_params*, *re_cb*, *re_params*, *i_cb*, *i_params*, *search_cb*, *search_params*)

status = dfdinterpolateex1d(*task*, *type*, *method*, *nsite*, *site*, *sitehint*, *ndorder*, *dorder*, *datahint*, *r*, *rhint*, *cell*, *le_cb*, *le_params*, *re_cb*, *re_params*, *i_cb*, *i_params*, *search_cb*, *search_params*)

status = dfsInterpolate1D(*task*, *type*, *method*, *nsite*, *site*, *sitehint*, *ndorder*, *dorder*, *datahint*, *r*, *rhint*, *cell*)

status = dfdInterpolate1D(*task*, *type*, *method*, *nsite*, *site*, *sitehint*, *ndorder*, *dorder*, *datahint*, *r*, *rhint*, *cell*)

status = dfsInterpolateEx1D(*task*, *type*, *method*, *nsite*, *site*, *sitehint*, *ndorder*, *dorder*, *datahint*, *r*, *rhint*, *cell*, *le_cb*, *le_params*, *re_cb*, *re_params*, *i_cb*, *i_params*, *search_cb*, *search_params*)

```
status = dfdInterpolateEx1D(task, type, method, nsite, site, sitehint, ndorder,
dorder, datahint, r, rhint, cell, le_cb, le_params, re_cb, re_params, i_cb, i_params,
search_cb, search_params)
```

Include Files

- Fortran: `mkl_df.f90`
- C: `mkl.h`

Input Parameters

Name	Type	Description
task	**Fortran:** `TYPE(DF_TASK)` **C:** `DFTaskPtr`	Descriptor of the task.
type	**Fortran:** `INTEGER` **C:** `const MKL_INT`	Type of spline-based computations. The parameter takes one or more values combined with an `OR` operation. For the list of possible values, see table "Computation Types Supported by the `df?interpolate1d`/ `df?interpolate1d` Routines".
method	**Fortran:** `INTEGER` **C:** `const MKL_INT`	Computation method. The supported value is `DF_METHOD_PP`.
nsite	**Fortran:** `INTEGER` **C:** `const MKL_INT`	Number of interpolation sites.

site	**Fortran:** `REAL(KIND=4) DIMENSION(*)` for `dfsinterpolate1d`/ `dfsinterpolateex1d`
	`REAL(KIND=8) DIMENSION(*)` for `dfdinterpolate1d`/ `dfdinterpolateex1d`
	C: `const float*` for `dfsInterpolate1D`/ `dfsInterpolateEx1D`
	`const double*` for `dfdInterpolate1D`/ `dfdInterpolateEx1D`
		Array of interpolation sites of size *nsite*. The structure of the array is defined by the *sitehint* parameter:
		• If sites form a non-uniform partition, the array should contain *nsite* values.
		• If sites form a uniform partition, the array should contain two entries that represent the left and the right interpolation sites. The first entry of the array contains the left-most interpolation point. The second entry of the array contains the right-most interpolation point.

sitehint	**Fortran:** `INTEGER` **C:** `const MKL_INT`	A flag describing the structure of the interpolation sites. For the list of possible values of *sitehint*, see table "Hint Values for Interpolation Sites". If you set the flag to `DF_NO_HINT`, the library interprets the site-defined partition as non-uniform.
ndorder	**Fortran:** `INTEGER` **C:** `const MKL_INT`	Maximal derivative order increased by one to be computed at interpolation sites.

Name	**Type**	**Description**
dorder	**Fortran:** INTEGER DIMENSION(*) **C:** const MKL_INT*	Array of size *ndorder* that defines the order of the derivatives to be computed at the interpolation sites. If all the elements in *dorder* are zero, the library computes the spline values only. If you do not need interpolation computations, set *ndorder* to zero and pass a NULL pointer to *dorder*.
datahint	**Fortran:** REAL(KIND=4) DIMENSION(*) for dfsinterpolate1d/ dfsinterpolateex1d REAL(KIND=8) DIMENSION(*) for dfdinterpolate1d/ dfdinterpolateex1d **C:** const float* for dfsInterpolate1D/ dfsInterpolateEx1D const double* for dfdInterpolate1D/ dfdInterpolateEx1D	Array that contains additional information about the structure of partition *x* and interpolation sites. This data helps to speed up the computation. If you provide a NULL pointer, the routine uses the default settings for computations. For details on the *datahint* array, see table "Structure of the *datahint* Array".
r	**Fortran:** REAL(KIND=4) DIMENSION(*) for dfsinterpolate1d/ dfsinterpolateex1d REAL(KIND=8) DIMENSION(*) for dfdinterpolate1d/ dfdinterpolateex1d **C:** float* for dfsInterpolate1D/ dfsInterpolateEx1D double* for dfdInterpolate1D/ dfdInterpolateEx1D	Array that contains results of computations at the interpolation sites. If you do not need spline-based interpolation or integration, set this pointer to NULL.
rhint	**Fortran:** INTEGER **C:** const MKL_INT	A flag describing the structure of the results. For the list of possible values of *rhint*, see table "Hint Values for the *rhint* Parameter". If you set the flag to DF_NO_HINT, the library stores the result in row-major format.
cell	**Fortran:** INTEGER DIMENSION(*) **C:** MKL_INT*	Array of cell indices in partition *x* that contain the interpolation sites. If you do not need cell indices, set this parameter to NULL.
le_cb	**Fortran:** INTEGER	User-defined callback function for extrapolation at the sites to the left of the interpolation interval. Set to NULL if you are not supplying a callback function.

Name	Type	Description
	C: constdfsInterpCallBack for dfsInterpolateEx1D constdfdInterpCallBack for dfdInterpolateEx1D	
le_params	**Fortran:** INTEGER DIMENSION(*) **C:** const void*	Pointer to additional user-defined parameters passed by the library to the *le_cb* function. Set to NULL if there are no additional parameters or if you are not supplying a callback function.
re_cb	**Fortran:** INTEGER **C:** constdfsInterpCallBack for dfsInterpolateEx1D constdfdInterpCallBack for dfdInterpolateEx1D	User-defined callback function for extrapolation at the sites to the right of the interpolation interval. Set to NULL if you are not supplying a callback function.
re_params	**Fortran:** INTEGER DIMENSION(*) **C:** const void*	Pointer to additional user-defined parameters passed by the library to the *re_cb* function. Set to NULL if there are no additional parameters or if you are not supplying a callback function.
i_cb	**Fortran:** INTEGER **C:** constdfsInterpCallBack for dfsInterpolateEx1D constdfdInterpCallBack for dfdInterpolateEx1D	User-defined callback function for interpolation within the interpolation interval. Set to NULL if you are not supplying a callback function.
i_params	**Fortran:** INTEGER DIMENSION(*) **C:** const void*	Pointer to additional user-defined parameters passed by the library to the *i_cb* function. Set to NULL if there are no additional parameters or if you are not supplying a callback function.
search_cb	**Fortran:** INTEGER **C:** constdfsSearchCellsCallBack for dfsInterpolateEx1D constdfdSearchCellsCallBack for dfdInterpolateEx1D	User-defined callback function for computing indices of cells that can contain interpolation sites. Set to NULL if you are not supplying a callback function.
search_params	**Fortran:** INTEGER DIMENSION(*) **C:** const void*	Pointer to additional user-defined parameters passed by the library to the *search_cb* function. Set to NULL if there are no additional parameters or if you are not supplying a callback function.

Output Parameters

Name	Type	Description
status	**Fortran:** INTEGER **C:** int	Status of the routine: • DF_STATUS_OK if the routine execution completed successfully. • Non-zero error code if the routine execution failed. See "Task Status and Error Reporting" for error code definitions.

Description

The df?interpolate1d/df?interpolateex1d routine performs spline-based computations with user-defined settings. The routine supports two types of computations for interpolation sites provided in array *site*:

Computation Types Supported by the df?interpolate1d/df?interpolateex1d Routines

Type	Description
DF_INTERP	Compute derivatives of predefined order. The derivative of the zero order is the spline value.
DF_CELL	Compute indices of cells in partition *x* that contain the sites.

If the sites do not belong to interpolation interval [*a*, *b*] , the library uses:

- polynomial P_0 of the spline constructed on interval [x_0, x_1] for computations at the sites to the left of *a*.
- polynomial P_{n-2} of the spline constructed on interval [x_{n-2}, x_{n-1}] for computations at the sites to the right of *b*.

Interpolation sites support the following hints:

Hint Values for Interpolation Sites

Value	Description
DF_NON_UNIFORM_PARTITION	Partition is non-uniform.
DF_UNIFORM_PARTITION	Partition is uniform.
DF_SORTED_DATA	Interpolation sites are sorted in the ascending order and define a non-uniform partition.
DF_NO_HINT	No hint is provided. By default, the partition defined by interpolation sites is interpreted as non-uniform.

NOTE
If you pass a sorted array of interpolation sites to the Intel MKL, set the *sitehint* parameter to the DF_SORTED_DATA value. The library uses this information when choosing the search algorithm and ignores any other data hints about the structure of the interpolation sites.

Data Fitting computation routines can use the following hints to speed up the computation:

- DF_UNIFORM_PARTITION describes the structure of breakpoints and the interpolation sites.
- DF_QUASI_UNIFORM_PARTITION describes the structure of breakpoints.

Pass the above hints to the library when appropriate.

The *r* pointer defines the memory location for the sets of interpolation and integration results for all coordinates of function *y*. The sets are stored one by one, in the successive order of the function coordinates from y_1 to y_{ny}.

You can define the following settings for packing the results within each set:

- Computation type: interpolation, integration, or both.
- Computation parameters: derivative orders.
- Storage format for the results. You can specify the format using the *rhint* parameter values described in the table below:

Hint Values for the *rhint* Parameter

Value	Description
`DF_MATRIX_STORAGE_ROWS`	Data is stored in row-major format according to C conventions.
`DF_MATRIX_STORAGE_COLS`	Data is stored in column-major format according to Fortran conventions.
`DF_NO_HINT`	No hint is provided. By default, the results are stored in row-major format.

For spline-based interpolation, you should set the derivatives whose values are required for the computation. You can provide the derivatives by setting the *dorder* array of size *ndorder* as follows:

$$dorder[i] = \begin{cases} 1, \text{if derivative of the } i-th \text{ order is required} \\ 0, \text{otherwise} \end{cases} \qquad i = 0, \ldots, ndorder - 1$$

See below a common structure of the storage formats of the interpolation results within each set *r* for computing derivatives of order i_1, i_2,...,i_m at *nsite* interpolation sites. In this description, *j* is the coordinate of the vector-valued function:

- Row-major format

$r_j(i_1, 0)$	$r_j(i_2, 0)$	...	$r_j(i_m, 0)$
$r_j(i_1, 1)$	$r_j(i_2, 1)$	...	$r_j(i_m, 1)$
...	...	...	...
$r_j(i_1, nsite - 1)$	$r_j(i_2, nsite - 1)$	...	$r_j(i_m, nsite - 1)$

- Column-major format

$r_j(i_1, 0)$	$r_j(i_1, 1)$	...	$r_j(i_1, nsite - 1)$
$r_j(i_2, 0)$	$r_j(i_2, 1)$	...	$r_j(i_2, nsite - 1)$
...	...	...	...
$r_j(i_m, 0)$	$r_j(i_m, 1)$	...	$r_j(i_m, nsite - 1)$

To speed up Data Fitting computations, use the *datahint* parameter that provides additional information about the structure of the partition and interpolation sites. This data represents a floating-point or a double array with the following structure:

Structure of the *datahint* Array

Element Number	Description
0	Task dimension

Element Number	Description
1	Type of additional information
2	Reserved field
3	The total number *q* of elements containing additional information.
4	Element (1)
...	...
q+3	Element (*q*)

Data Fitting computation functions support the following types of additional information for *datahint*[1]:

Types of Additional Information

Type	Element Number	Parameter
`DF_NO_APRIORI_INFO`	0	No parameters are provided. Information about the data structure is absent.
`DF_APRIORI_MOST_LIKELY_CELL`	1	Index of the cell that is likely to contain interpolation sites.

To compute indices of the cells that contain interpolation sites, provide the pointer to the array of size *nsite* for the results. The library supports the following scheme of cell indexing for the given partition$\{x_i\}$, $i=1,\dots,nx$:

cell[j] = i, if *site*[j] $\in[x_i, x_{i+1})$, $i = 0,\dots,$ *nx*,

where

- $x_0 = -\infty$
- $x_{nx+1} = +\infty$
- $j = 0,\dots,$ *nsite*-1

To perform interpolation computations with spline types unsupported in the Data Fitting component, use the extended version of the routine `df?interpolateex1d`. With this routine, you can provide user-defined callback functions for computations within, to the left of, or to the right of interpolaton interval [*a*, *b*]. The callback functions compute indices of the cells that contain the specified interpolation sites or can serve as an approximation for computing the exact indices of such cells.

If you do not pass any function for computations at the sites outside the interval [*a*, *b*], the routine uses the default settings.

See Also

df?interpcallback
df?searchcellscallback

df?integrate1d/df?integrateex1d

Computes a spline-based integral.

Syntax

status = dfsintegrate1d(*task*, *method*, *nlim*, *llim*, *llimhint*, *rlim*, *rlimhint*, *ldatahint*, *rdatahint*, *r*, *rhint*)

status = dfdintegrate1d(*task*, *method*, *nlim*, *llim*, *llimhint*, *rlim*, *rlimhint*, *ldatahint*, *rdatahint*, *r*, *rhint*) *status* = dfsintegrateex1d(*task*, *method*, *nlim*, *llim*, *llimhint*, *rlim*, *rlimhint*, *ldatahint*, *rdatahint*, *r*, *rhint*, *le_cb*, *le_params*, *re_cb*, *re_params*, *i_cb*, *i_params*, *search_cb*, *search_params*)

status = dfdintegrateex1d(*task*, *method*, *nlim*, *llim*, *llimhint*, *rlim*, *rlimhint*, *ldatahint*, *rdatahint*, *r*, *rhint*, *le_cb*, *le_params*, *re_cb*, *re_params*, *i_cb*, *i_params*, *search_cb*, *search_params*)

status = dfsIntegrate1D(*task*, *method*, *nlim*, *llim*, *llimhint*, *rlim*, *rlimhint*, *ldatahint*, *rdatahint*, *r*, *rhint*)

status = dfdIntegrate1D(*task*, *method*, *nlim*, *llim*, *llimhint*, *rlim*, *rlimhint*, *ldatahint*, *rdatahint*, *r*, *rhint*)

status = dfsIntegrateEx1D(*task*, *method*, *nlim*, *llim*, *llimhint*, *rlim*, *rlimhint*, *ldatahint*, *rdatahint*, *r*, *rhint*, *le_cb*, *le_params*, *re_cb*, *re_params*, *i_cb*, *i_params*, *search_cb*, *search_params*)

status = dfdIntegrateEx1D(*task*, *method*, *nlim*, *llim*, *llimhint*, *rlim*, *rlimhint*, *ldatahint*, *rdatahint*, *r*, *rhint*, *le_cb*, *le_params*, *re_cb*, *re_params*, *i_cb*, *i_params*, *search_cb*, *search_params*)

Include Files

- Fortran: mkl_df.f90
- C: mkl.h

Input Parameters

Name	Type	Description
task	**Fortran:** TYPE(DF_TASK) **C:** DFTaskPtr	Descriptor of the task.
method	**Fortran:** INTEGER **C:** const MKL_INT	Integration method. The supported value is DF_METHOD_PP.
nlim	**Fortran:** INTEGER **C:** const MKL_INT	Number of pairs of integraion limits.
llim	**Fortran:** REAL(KIND=4) DIMENSION(*) for dfsintegrate1d/ dfsintegrateex1d REAL(KIND=8) DIMENSION(*) for dfdintegrate1d/ dfdintegrateex1d **C:** const float* for dfsIntegrate1D/ dfsIntegrateEx1D const double* for dfdIntegrate1D/ dfdIntegrateEx1D	Array of size *nlim* that defines the left-side integration limits.

Name	**Type**	**Description**
llimhint	**Fortran:** INTEGER **C:** const MKL_INT	A flag describing the structure of the left-side integration limits *llim*. For the list of possible values of *llimhint*, see table "Hint Values for Integration Limits". If you set the flag to the DF_NO_HINT value, the library assumes that the left-side integration limits define a non-uniform partition.
rlim	**Fortran:** REAL(KIND=4) DIMENSION(*) for dfsintegrate1d/ dfsintegrateex1d REAL(KIND=8) DIMENSION(*) for dfdintegrate1d/ dfdintegrateex1d **C:** const float* for dfsIntegrate1D/ dfsIntegrateEx1D const double* for dfdIntegrate1D/ dfdIntegrateEx1D	Array of size *nlim* that defines the right-side integration limits.
rlimhint	**Fortran:** INTEGER **C:** const MKL_INT	A flag describing the structure of the right-side integration limits *rlim*. For the list of possible values of *rlimhint*, see table "Hint Values for Integration Limits". If you set the flag to the DF_NO_HINT value, the library assumes that the right-side integration limits define a non-uniform partition.
ldatahint	**Fortran:** REAL(KIND=4) DIMENSION(*) for dfsintegrate1d/ dfsintegrateex1d REAL(KIND=8) DIMENSION(*) for dfdintegrate1d/ dfdintegrateex1d **C:** const float* for dfsIntegrate1D/ dfsIntegrateEx1D const double* for dfdIntegrate1D/ dfdIntegrateEx1D	Array that contains additional information about the structure of partition *x* and left-side integration limits. For details on the *ldatahint* array, see table "Structure of the *datahint* Array" in the description of the df?Intepolate1D function.
rdatahint	**Fortran:** REAL(KIND=4) DIMENSION(*) for dfsintegrate1d/ dfsintegrateex1d	Array that contains additional information about the structure of partition *x* and right-side integration limits. For details on the *rdatahint* array, see table "Structure of the *datahint* Array" in the description of the df?Intepolate1D function.

Name	Type	Description

	`REAL(KIND=8) DIMENSION(*)` for `dfdintegrate1d`/ `dfdintegrateex1d`
	
	C: `const float*` for `dfsIntegrate1D`/ `dfsIntegrateEx1D`
	
	`const double*` for `dfdIntegrate1D`/ `dfdIntegrateEx1D`	

r	**Fortran:** `REAL(KIND=4) DIMENSION(*)` for `dfsintegrate1d`/ `dfsintegrateex1d`
	
	`REAL(KIND=8) DIMENSION(*)` for `dfdintegrate1d`/ `dfdintegrateex1d`
	
	C: `float*` for `dfsIntegrate1D`/ `dfsIntegrateEx1D`
	
	`double*` for `dfdIntegrate1D`/ `dfdIntegrateEx1D`
		Array of `integration` results. The size of the array should be sufficient to hold *nlim***ny* values, where *ny* is the dimension of the vector-valued function. The integration results are packed according to the settings in *rhint*.

rhint	**Fortran:** `INTEGER` **C:** `const MKL_INT`	A flag describing the structure of the results. For the list of possible values of *rhint*, see table "Hint Values for Integration Results". If you set the flag to the `DF_NO_HINT` value, the library stores the results in row-major format.
le_cb	**Fortran:** `INTEGER` **C:** `constdfsIntegrCallBack` for `dfsIntegrateEx1D` `constdfdIntegrCallBack` for `dfdIntegrateEx1D`	User-defined callback function for integration on interval [*llim*[*i*], `min`(*rlim*[*i*], *a*)) for *llim*[*i*] < *a* . Set to `NULL` if you are not supplying a callback function.
le_params	**Fortran:** `INTEGER DIMENSION(*)` **C:** `const void*`	Pointer to additional user-defined parameters passed by the library to the *le_cb* function. Set to `NULL` if there are no additional parameters or if you are not supplying a callback function.
re_cb	**Fortran:** `INTEGER` **C:** `constdfsInterpCallBack` for `dfsIntegrateEx1D`	User-defined callback function for integration on interval [`max`(*llim*[*i*], *b*), *rlim*[*i*]) for *rlim*[*i*] ≥*b*. Set to `NULL` if you are not supplying a callback function.

Name	Type	Description
	`constdfdInterpCallBack` for `dfdIntegrateEx1D`	
re_params	**Fortran:** `INTEGER DIMENSION(*)` **C:** `const void*`	Pointer to additional user-defined parameters passed by the library to the *re_cb* function. Set to `NULL` if there are no additional parameters or if you are not supplying a callback function.
i_cb	**Fortran:** `INTEGER` **C:** `constdfsIntegrCallBack` for `dfsIntegrateEx1D` `constdfdIntegrCallBack` for `dfdIntegrateEx1D`	User-defined callback function for integration on interval [`max`(*a*, *llim*[*i*],), `min`(*rlim*[*i*], *b*)). Set to `NULL` if you are not supplying a callback function.
i_params	**Fortran:** `INTEGER DIMENSION(*)` **C:** `const void*`	Pointer to additional user-defined parameters passed by the library to the *i_cb* function. Set to `NULL` if there are no additional parameters or if you are not supplying a callback function.
search_cb	**Fortran:** `INTEGER` **C:** `constdfsSearchCellsCallBack` for `dfsIntegrateEx1D` `constdfdSearchCellsCallBack` for `dfdIntegrateEx1D`	User-defined callback function for computing indices of cells that can contain interpolation sites. Set to `NULL` if you are not supplying a callback function.
search_params	**Fortran:** `INTEGER DIMENSION(*)` **C:** `const void*`	Pointer to additional user-defined parameters passed by the library to the *search_cb* function. Set to `NULL` if there are no additional parameters or if you are not supplying a callback function.

Output Parameters

Name	Type	Description
status	**Fortran:** `INTEGER` **C:** `int`	Status of the routine: • `DF_STATUS_OK` if the routine execution completed successfully. • Non-zero error code if the routine execution failed. See "Task Status and Error Reporting" for error code definitions.

Description

The `df?integrate1d`/`df?integrateex1d` routine computes spline-based integral on user-defined intervals

$$I(i,j) = \int_{llim[i]}^{rlim[i]} f_j(x)dx,\ i = 0,\ldots,nlim-1,\ j = 0,\ldots,ny-1$$

If *rlim*[*i*] < *llim*[*i*], the routine returns

$$I(i,j) = -\int_{rlim[i]}^{llim[i]} f_j(x)dx$$

The routine supports the following hint values for integration results:

Hint Values for Integration Results

Value	Description
`DF_MATRIX_STORAGE_ROWS`	Data is stored in row-major format according to C conventions.
`DF_MATRIX_STORAGE_COLS`	Data is stored in column-major format according to Fortran conventions.
`DF_NO_HINT`	No hint is provided. By default, the coordinates of vector-valued function *y* are provided and stored in row-major format.

A common structure of the storage formats for the integration results is as follows:

- Row-major format

I(0, 0)	...	*I*(0, *nlim* - 1])
...	...	...
I (*ny* - 1, 0)	...	*I*(*ny* - 1, *nlim* - 1])

- Column-major format

I(0, 0)	...	*I* (*ny* - 1, 0)
...	...	...
I(0, *nlim* - 1])	...	*I*(*ny* - 1, *nlim* - 1])

Using the *llimhint* and *rlimhint* parameters, you can provide the following hint values for integration limits:

Hint Values for Integration Limits

Value	Description
`DF_SORTED_DATA`	Integration limits are sorted in the ascending order and define a non-uniform partition.
`DF_NON_UNIFORM_PARTITION`	Partition defined by integration limits is non-uniform.
`DF_UNIFORM_PARTITION`	Partition defined by integration limits is uniform.
`DF_NO_HINT`	No hint is provided. By default, partition defined by integration limits is interpreted as non-uniform.

To compute integration with splines unsupported in the Data Fitting component, use the extended version of the routine `df?integrateex1d`. With this routine, you can provide user-defined callback functions that compute:

- integrals within, to the left of, or to the right of the interpolation interval [*a*, *b*]
- indices of cells that contain the provided integration limits or can serve as an approximation for computing the exact indices of such cells

If you do not pass callback functions, the routine uses the default settings.

See Also

df?interpolate1d/df?interpolateex1d
df?integrcallback
df?searchcellscallback

df?searchcells1d/df?searchcellsex1d

Searches sub-intervals containing interpolation sites.

Syntax

status = dfssearchcells1d(*task*, *method*, *nsite*, *site*, *sitehint*, *datahint*, *cell*)

status = dfdsearchcells1d(*task*, *method*, *nsite*, *site*, *sitehint*, *datahint*, *cell*)

status = dfssearchcellsex1d(*task*, *method*, *nsite*, *site*, *sitehint*, *datahint*, *cell*, *search_cb*, *search_params*)

status = dfdsearchcellsex1d(*task*, *method*, *nsite*, *site*, *sitehint*, *datahint*, *cell*, *search_cb*, *search_params*)

status = dfsSearchCells1D(*task*, *method*, *nsite*, *site*, *sitehint*, *datahint*, *cell*)

status = dfdSearchCells1D(*task*, *method*, *nsite*, *site*, *sitehint*, *datahint*, *cell*)

status = dfsSearchCellsEx1D(*task*, *method*, *nsite*, *site*, *sitehint*, *datahint*, *cell*, *search_cb*, *search_params*)

status = dfdSearchCellsEx1D(*task*, *method*, *nsite*, *site*, *sitehint*, *datahint*, *cell*, *search_cb*, *search_params*)

Include Files

- Fortran: `mkl_df.f90`
- C: `mkl.h`

Input Parameters

Name	Type	Description
task	**Fortran:** `TYPE(DF_TASK)` **C:** `DFTaskPtr`	Descriptor of the task.
method	**Fortran:** `INTEGER` **C:** `const MKL_INT`	Search method. The supported value is `DF_METHOD_STD`.
nsite	**Fortran:** `INTEGER` **C:** `const MKL_INT*`	Number of interpolation sites.
site	**Fortran:** `REAL(KIND=4) DIMENSION(*)` for `dfssearchcells1d`/ `dfssearchcellsex1d`	Array of interpolation sites of size *nsite*. The structure of the array is defined by the *sitehint* parameter: • If the sites form a non-uniform partition, the array should contain *nsite* values.

Name	Type	Description

	`REAL(KIND=8) DIMENSION(*)` for `dfdsearchcells1d`/ `dfdsearchcellsex1d`
	
	C: `const float*` for `dfsSearchCells1D`/ `dfsSearchCellsEx1D`
	
	`const double*` for `dfdSearchCells1D`/ `dfdSearchCellsEx1D`
		• If the sites form a uniform partition, the array should contain two entries that represent the left-most and the right-most interpolation sites. The first entry of the array contains the left-most interpolation point. The second entry of the array contains the right-most interpolation point.

sitehint	**Fortran:** `INTEGER` **C:** `const MKL_INT`	A flag describing the structure of the interpolation sites. For the list of possible values of *sitehint*, see table "Hint Values for Interpolation Sites". If you set the flag to `DF_NO_HINT`, the library interprets the site-defined partition as non-uniform.

datahint	**Fortran:** `REAL(KIND=4) DIMENSION(*)` for `dfssearchcells1d`/ `dfssearchcellsex1d`
	
	`REAL(KIND=8) DIMENSION(*)` for `dfdsearchcells1d`/ `dfdsearchcellsex1d`
	
	C: `const float*` for `dfsSearchCells1D`/ `dfsSearchCellsEx1D`
	
	`const double*` for `dfdSearchCells1D`/ `dfdSearchCellsEx1D`
		Array that contains additional information about the structure of partition *x* and interpolation sites. This data helps to speed up the computation. If you provide a `NULL` pointer, the routine uses the default settings for computations. For details on the *datahint* array, see table "Structure of the *datahint* Array".

cell	**Fortran:** `INTEGER DIMENSION(*)` **C:** `MKL_INT*`	Array of cell indices in partition *x* that contain the interpolation sites.
search_cb	**Fortran:** `INTEGER` **C:** `constdfsSearchCellsCallBack` for `dfsSearchCellsEx1D` `constdfdSearchCellsCallBack` for `dfdSearchCellsEx1D`	User-defined callback function for computing indices of cells that can contain interpolation sites. Set to `NULL` if you are not supplying a callback function.
search_params	**Fortran:** `INTEGER DIMENSION(*)` **C:** `const void*`	Pointer to additional user-defined parameters passed by the library to the *search_cb* function. Set to `NULL` if there are no additional parameters or if you are not supplying a callback function.

Output Parameters

Name	Type	Description
status	**Fortran:** `INTEGER` **C:** `int`	Status of the routine: • `DF_STATUS_OK` if the routine execution completed successfully. • Non-zero error code if the routine execution failed. See "Task Status and Error Reporting" for error code definitions.

Description

The `df?searchcells1d`/`df?searchcellsex1d` routines return array *`cell`* of indices of sub-intervals (cells) in partition *`x`* that contain interpolation sites available in array *`site`*. For details on the cell indexing scheme, see the description of the `df?interpolate1d`/`df?interpolateex1d` computation routines.

Use the *`datahint`* parameter to provide additional information about the structure of the partition and/or interpolation sites. The definition of the *`datahint`* parameter is availalbe in the description of the `df?interpolate1d`/`df?interpolateex1d` computation routines.

For description of the user-defined callback for computation of cell indices, see `df?searchcellscallback`.

See Also

df?interpolate1d/df?interpolateex1d
df?searchcellscallback

df?interpcallback

A callback function for user-defined interpolation to be passed into `df?interpolateex1d`*.*

Syntax

```
status = dfsinterpcallback(n, cell, site, r, params)
status = dfdinterpcallback(n, cell, site, r, params)
status = dfsInterpCallBack(n, cell, site, r, params)
status = dfdInterpCallBack(n, cell, site, r, params)
```

Include Files

- Fortran: `mkl_df.f90`
- C: `mkl.h`

Input Parameters

Name	Type	Description
n	**Fortran:** `INTEGER(KIND=8)` **C:** `long long*`	Number of interpolation sites.
cell	**Fortran:** `INTEGER(KIND=8) DIMENSION(*)` **C:** `long long*`	Array of size *`n`* containing indices of the cells to which the interpolation sites in array *`site`* belong.

Name	Type	Description

site	**Fortran:** `REAL(KIND=4) DIMENSION(*)` for `dfsinterpcallback`
	`REAL(KIND=8) DIMENSION(*)` for `dfdinterpcallback`
	C: `float*` for `dfsInterpCallBack`
	`double*` for `dfdInterpCallBack`
		Array of interpolation sites of size *n*.

r	**Fortran:** `REAL(KIND=4) DIMENSION(*)` for `dfsinterpcallback`
	`REAL(KIND=8) DIMENSION(*)` for `dfdinterpcallback`
	C: `float*` for `dfsInterpCallBack`
	`double*` for `dfdInterpCallBack`
		Array of the computed interpolation results packed in row-major format.

params	**Fortran:** `INTEGER DIMENSION(*)` **C:** `void*`	Pointer to user-defined parameters of the callback function.

Output Parameters

Name	Type	Description
status	**Fortran:** `INTEGER` **C:** `int`	The status returned by the callback function: • Zero indicates successful completion of the callback operation. • A negative value indicates an error. • A positive value indicates a warning. See "Task Status and Error Reporting" for error code definitions.

Description

When passed into the `df?interpolateex1d` routine, this function performs user-defined interpolation operation.

See Also

df?interpolate1d/df?interpolateex1d
df?searchcellscallback

df?integrcallback

A callback function that you can pass into `df?integrateex1d` *to define integration computations.*

Syntax

status = dfsintegrcallback(*n*, *lcell*, *llim*, *rcell*, *rlim*, *r*, *params*)

status = dfdintegrcallback(*n*, *lcell*, *llim*, *rcell*, *rlim*, *r*, *params*)

status = dfsIntegrCallBack(*n*, *lcell*, *llim*, *rcell*, *rlim*, *r*, *params*)

status = dfdIntegrCallBack(*n*, *lcell*, *llim*, *rcell*, *rlim*, *r*, *params*)

Include Files

- Fortran: `mkl_df.f90`
- C: `mkl.h`

Input Parameters

Name	Type	Description
n	**Fortran:** `INTEGER(KIND=8)` **C:** `long long*`	Number of pairs of integration limits.
lcell	**Fortran:** `INTEGER(KIND=8) DIMENSION(*)` **C:** `long long*`	Array of size *n* with indices of the cells that contain the left-side integration limits in array *llim*.

llim	**Fortran:** `REAL(KIND=4) DIMENSION(*)` for `dfsintegrcallback`
	
	`REAL(KIND=8) DIMENSION(*)` for `dfdintegrcallback`
	
	C: `float*` for `dfsIntegrCallBack`
	
	`double*` for `dfdIntegrCallBack`
		Array of size *n* that holds the left-side integration limits.

rcell	**Fortran:** `INTEGER(KIND=8) DIMENSION(*)` **C:** `long long*`	Array of size *n* with indices of the cells that contain the right-side integration limits in array *rlim*.
rlim	**Fortran:** `REAL(KIND=4) DIMENSION(*)` for `dfsintegrcallback` `REAL(KIND=8) DIMENSION(*)` for `dfdintegrcallback` **C:** `float*` for `dfsIntegrCallBack`	Array of size *n* that holds the right-side integration limits.

Name	Type	Description
	`double*` for `dfdIntegrCallBack`	

r	**Fortran:** `REAL(KIND=4) DIMENSION(*)` for `dfsintegrcallback`
	`REAL(KIND=8) DIMENSION(*)` for `dfdintegrcallback`
	C: `float*` for `dfsIntegrCallBack`
	`double*` for `dfdIntegrCallBack`
		Array of integration results. For packing the results in row-major format, follow the instructions described in `df?interpolate1d/df?interpolateex1d`.

params	**Fortran:** `INTEGER DIMENSION(*)` **C:** `void*`	Pointer to user-defined parameters of the callback function.

Output Parameters

Name	Type	Description
status	**Fortran:** `INTEGER` **C:** `int`	The status returned by the callback function: • Zero indicates successful completion of the callback operation. • A negative value indicates an error. • A positive value indicates a warning. See "Task Status and Error Reporting" for error code definitions.

Description

When passed into the `df?integrateex1d` routine, this function defines integration computations. If at least one of the integration limits is outside the interpolation interval [*a*, *b*], the library decomposes the integration into sub-intervals that belong to the extrapolation range to the left of *a*, the extrapolation range to the right of *b*, and the interpolation interval [*a*, *b*], as follows:

- If the left integration limit is to the left of the interpolation interval (*llim*< *a*), the `df?integrateex1d` routine passes *llim* as the left integration limit and `min(`*rlim*, *a*`)` as the right integration limit to the user-defined callback function.
- If the right integration limit is to the right of the interpolation interval (*rlim*> *b*), the `df?integrateex1d` routine passes `max(`*llim*, *b*`)` as the left integration limit and *rlim* as the right integration limit to the user-defined callback function.
- If the left and the right integration limits belong to the interpolation interval, the `df?integrateex1d` routine passes them to the user-defined callback function unchanged.

The value of the integral is the sum of integral values obtained on the sub-intervals.

See Also

df?integrate1d/df?integrateex1d
df?integrcallback

df?searchcellscallback

A callback function for user-defined search to be passed into `df?interpolateex1d` *or* `df?searchcellsex1d`*.*

Syntax

```
status = dfssearchcellscallback(n, site, cell, flag, params)
status = dfdsearchcellscallback(n, site, cell, flag, params)
status = dfsSearchCellsCallBack(n, site, cell, flag, params)
status = dfdSearchCellsCallBack(n, site, cell, flag, params)
```

Include Files

- Fortran: `mkl_df.f90`
- C: `mkl.h`

Input Parameters

Name	Type	Description
n	**Fortran:** `INTEGER(KIND=8)` **C:** `long long*`	Number of interpolation sites.

site	**Fortran:** `REAL(KIND=4) DIMENSION(*)` for `dfssearchcellscallback`
	`REAL(KIND=8) DIMENSION(*)` for `dfdsearchcellscallback`
	C: `float*` for `dfsSearchCellsCallBack`
	`double*` for `dfdSearchCellsCallBack`
		Array of interpolation sites of size *n*.

cell	**Fortran:** `INTEGER(KIND=8) DIMENSION(*)` **C:** `long long*`	Array of size *n* that returns indices of the cells computed by the callback function.
flag	**Fortran:** `INTEGER(KIND=4) DIMENSION(*)` **C:** `int*`	Array of size *n*, with values set as follows: • If the cell with index *cell*[*i*] contains *site*[*i*], set *flag*[*i*] to 1. • Otherwise, set *flag*[*i*] to zero. In this case, the library interprets the index as an approximation and computes the index of the cell containing *site*[*i*] by using the provided index as a starting point for the search.
params	**Fortran:** `INTEGER DIMENSION(*)` **C:** `void*`	Pointer to user-defined parameters of the callback function.

Name	**Type**	**Description**

Output Parameters

Name	**Type**	**Description**
status	**Fortran:** `INTEGER` **C:** `int`	The status returned by the callback function: • Zero indicates successful completion of the callback operation. • A negative value indicates an error. • The `DF_STATUS_EXACT_RESULT` status indicates that cell indices returned by the callback function are exact. In this case, you do not need to initialize entries of the *flag* array. • A positive value indicates a warning. See "Task Status and Error Reporting" for error code definitions.

Description

When passed into the `df?interpolateex1d` or `df?searchcellsex1d` routine, this function performs a user-defined search.

See Also

df?interpolate1d/df?interpolateex1d
df?interpcallback

Task Destructors

Task destructors are routines used to delete task descriptors and deallocate the corresponding memory resources. The Data Fitting task destructor `dfdeletetask` destroys a Data Fitting task and frees the memory.

dfdeletetask

Destroys a Data Fitting task object and frees the memory.

Syntax

```
status = dfdeletetask(task)

status = dfDeleteTask(&task)
```

Include Files

- Fortran: `mkl_df.f90`
- C: `mkl.h`

Input Parameters

Name	Type	Description
`task`	**Fortran:** `TYPE(DF_TASK)` **C:** `DFTaskPtr`	Descriptor of the task to destroy.

Output Parameters

Name	Type	Description
`status`	**Fortran:** `INTEGER` **C:** `int`	Status of the routine: • `DF_STATUS_OK` if the task is deleted successfully. • Non-zero error code if the operation failed. See "Task Status and Error Reporting" for error code definitions.

Description

Given a pointer to a task descriptor, this routine deletes the Data Fitting task descriptor and frees the memory allocated for the structure. If the task is deleted successfully, the routine sets the task pointer to `NULL`. Otherwise, the routine returns an error code.