NAG Library Function Document

nag_2d_spline_deriv_rect (e02dhc)

1 Purpose

nag_2d_spline_deriv_rect (e02dhc) computes the partial derivative (of order ν_x, ν_y), of a bicubic spline approximation to a set of data values, from its B-spline representation, at points on a rectangular grid in the x-y plane. This function may be used to calculate derivatives of a bicubic spline given in the form produced by nag_2d_spline_interpolant (e01dac), nag_2d_spline_fit_panel (e02dac), nag_2d_spline_fit_grid (e02dcc) and nag_2d_spline_fit_scat (e02ddc).

2 Specification

```
#include <nag.h>
#include <nage02.h>
void nag_2d_spline_deriv_rect (Integer mx, Integer my, const double x[],
     const double y[], Integer nux, Integer nuy, double z[],
     Nag_2dSpline *spline, NagError *fail)
```

3 Description

nag_2d_spline_deriv_rect (e02dhc) determines the partial derivative $\frac{\partial^{\nu_x+\nu_y}}{\partial x^{\nu_x}\partial y^{\nu_y}}$ of a smooth bicubic spline approximation $s(x,y)$ at the set of data points $\left(x_q, y_r\right)$.

The spline is given in the B-spline representation

$$s(x,y) = \sum_{i=1}^{n_x-4}\sum_{j=1}^{n_y-4} c_{ij} M_i(x) N_j(y), \tag{1}$$

where $M_i(x)$ and $N_j(y)$ denote normalized cubic B-splines, the former defined on the knots λ_i to λ_{i+4} and the latter on the knots μ_j to μ_{j+4}, with n_x and n_y the total numbers of knots of the computed spline with respect to the x and y variables respectively. For further details, see Hayes and Halliday (1974) for bicubic splines and de Boor (1972) for normalized B-splines. This function is suitable for B-spline representations returned by nag_2d_spline_interpolant (e01dac), nag_2d_spline_fit_panel (e02dac), nag_2d_spline_fit_grid (e02dcc) and nag_2d_spline_fit_scat (e02ddc).

The partial derivatives can be up to order 2 in each direction; thus the highest mixed derivative available is $\frac{\partial^4}{\partial x^2 \partial y^2}$.

The points in the grid are defined by coordinates x_q, for $q = 1, 2, \ldots, m_x$, along the x axis, and coordinates y_r, for $r = 1, 2, \ldots, m_y$, along the y axis.

4 References

de Boor C (1972) On calculating with B-splines *J. Approx. Theory* **6** 50–62

Dierckx P (1981) An improved algorithm for curve fitting with spline functions *Report TW54* Department of Computer Science, Katholieke Univerciteit Leuven

Dierckx P (1982) A fast algorithm for smoothing data on a rectangular grid while using spline functions *SIAM J. Numer. Anal.* **19** 1286–1304

Hayes J G and Halliday J (1974) The least squares fitting of cubic spline surfaces to general data sets *J. Inst. Math. Appl.* **14** 89–103

Reinsch C H (1967) Smoothing by spline functions *Numer. Math.* **10** 177–183

5 Arguments

1: **mx** – Integer *Input*

On entry: m_x, the number of grid points along the x axis.

Constraint: **mx** ≥ 1.

2: **my** – Integer *Input*

On entry: m_y, the number of grid points along the y axis.

Constraint: **my** ≥ 1.

3: **x**[**mx**] – const double *Input*

On entry: **x**$[q-1]$ must be set to x_q, the x coordinate of the qth grid point along the x axis, for $q = 1, 2, \ldots, m_x$, on which values of the partial derivative are sought.

Constraint: $x_1 < x_2 < \cdots < x_{m_x}$.

4: **y**[**my**] – const double *Input*

On entry: **y**$[r-1]$ must be set to y_r, the y coordinate of the rth grid point along the y axis, for $r = 1, 2, \ldots, m_y$ on which values of the partial derivative are sought.

Constraint: $y_1 < y_2 < \cdots < y_{m_y}$.

5: **nux** – Integer *Input*

On entry: specifies the order, ν_x of the partial derivative in the x-direction.

Constraint: $0 \leq$ **nux** ≤ 2.

6: **nuy** – Integer *Input*

On entry: specifies the order, ν_y of the partial derivative in the y-direction.

Constraint: $0 \leq$ **nuy** ≤ 2.

7: **z**[**mx** $\times$ **my**] – double *Output*

On exit: **z**$[m_y \times (q-1) + r - 1]$ contains the derivative $\frac{\partial^{\nu_x+\nu_y}}{\partial x^{\nu_x}\partial y^{\nu_y}} s\left(x_q, y_r\right)$, for $q = 1, 2, \ldots, m_x$ and $r = 1, 2, \ldots, m_y$.

8: **spline** – Nag_2dSpline * *Input*

Pointer to structure of type Nag_2dSpline describing the bicubic spline approximation to be differentiated.

In normal usage, the call to nag_2d_spline_deriv_rect (e02dhc) follows a call to nag_2d_spline_interpolant (e01dac), nag_2d_spline_fit_panel (e02dac), nag_2d_spline_fit_grid (e02dcc) or nag_2d_spline_fit_scat (e02ddc), in which case, members of the structure **spline** will have been set up correctly for input to nag_2d_spline_deriv_rect (e02dhc).

9: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_INT

On entry, **mx** = ⟨*value*⟩.
Constraint: **mx** ≥ 1.

On entry, **my** = ⟨*value*⟩.
Constraint: **my** ≥ 1.

On entry, **nux** = ⟨*value*⟩.
Constraint: $0 \leq$ **nux** ≤ 2.

On entry, **nuy** = ⟨*value*⟩.
Constraint: $0 \leq$ **nuy** ≤ 2.

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

NE_NOT_STRICTLY_INCREASING

On entry, for $i =$ ⟨*value*⟩, $\mathbf{x}[i-2] =$ ⟨*value*⟩ and $\mathbf{x}[i-1] =$ ⟨*value*⟩.
Constraint: $\mathbf{x}[i-2] \leq \mathbf{x}[i-1]$, for $i = 2, 3, \ldots,$ **mx**.

On entry, for $i =$ ⟨*value*⟩, $\mathbf{y}[i-2] =$ ⟨*value*⟩ and $\mathbf{y}[i-1] =$ ⟨*value*⟩.
Constraint: $\mathbf{y}[i-2] \leq \mathbf{y}[i-1]$, for $i = 2, 3, \ldots,$ **my**.

7 Accuracy

On successful exit, the partial derivatives on the given mesh are accurate to ***machine precision*** with respect to the supplied bicubic spline. Please refer to Section 7 in nag_2d_spline_interpolant (e01dac), nag_2d_spline_fit_panel (e02dac), nag_2d_spline_fit_grid (e02dcc) and nag_2d_spline_fit_scat (e02ddc) of the function document for the respective function which calculated the spline approximant for details on the accuracy of that approximation.

8 Parallelism and Performance

Not applicable.

9 Further Comments

None.

10 Example

This example reads in values of m_x, m_y, x_q, for $q = 1, 2, \ldots, m_x$, and y_r, for $r = 1, 2, \ldots, m_y$, followed by values of the ordinates $f_{q,r}$ defined at the grid points (x_q, y_r). It then calls nag_2d_spline_fit_grid (e02dcc) to compute a bicubic spline approximation for one specified value of S. Finally it evaluates the spline and its first x derivative at a small sample of points on a rectangular grid by calling nag_2d_spline_deriv_rect (e02dhc).

10.1 Program Text

```
/* nag_2d_spline_deriv_rect (e02dhc) Example Program.
 *
 * Copyright 2011, Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */
```

```
#include <math.h>
#include <string.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nage02.h>
#include <nagx04.h>

#ifdef __cplusplus
extern "C" {
#endif

static void NAG_CALL print_spline(Integer *ngx, double *gridx, Integer *ngy,
                                  double *gridy, double *z, double *zder,
                                  Integer *exit_status);

#ifdef __cplusplus
}
#endif

#define F(I, J) f[my*(I)+(J)]

int main(void)
{
  /* Scalars */
  Integer      exit_status = 0;
  Integer      i, j, mx, my, ngx, ngy, nux, nuy, nxest, nyest;
  double       delta, fp, s, xhi, xlo, yhi, ylo;
  /* Arrays */
  double       *f = 0, *gridx = 0, *gridy = 0, *x = 0, *y = 0, *z = 0,
  *zder = 0;
  /* NAG types */
  Nag_2dSpline spline;
  Nag_Comm     warmstartinf;
  Nag_Start    startc;
  NagError     fail;

  INIT_FAIL(fail);

  printf("nag_2d_spline_deriv_rect (e02dhc) Example Program Results\n");
  fflush(stdout);

  /* Skip heading in data file*/
  scanf("%*[^\n] ");

  /* Input the number of X, Y co-ordinates MX, MY.*/
  scanf("%ld %ld%*[^\n]", &mx, &my);
  nxest = mx + 4;
  nyest = my + 4;
  spline.nx = 4;
  spline.ny = 4;

  /* Alocations for spline fitting */
  if (!(x = NAG_ALLOC((mx), double)) ||
      !(y = NAG_ALLOC((my), double)) ||
      !(f = NAG_ALLOC((mx * my), double))
      )
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
  for (i = 0; i < mx; i++) scanf("%lf", &x[i]);
  scanf("%*[^\n]");
  for (i = 0; i < my; i++) scanf("%lf", &y[i]);
  scanf("%*[^\n]");

  /* Input the MX*MY function values F at grid points and smoothing factor.*/
  for (i = 0; i < mx; i++)
    for (j = 0; j < my; j++)
      scanf("%lf", &F(i, j));
```

```
  scanf("%*[^\n]");
  scanf("%lf%*[^\n]", &s);

  /* nag_2d_spline_fit_grid (e02dcc).
   * Least squares bicubic spline fit with automatic knot placement,
   * two variables (rectangular grid)
   */
  startc = Nag_Cold;
  nag_2d_spline_fit_grid(startc, mx, x, my, y, f, s, nxest, nyest, &fp,
                         &warmstartinf, &spline, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_2d_spline_fit_grid (e02dcc)\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  printf("\nSpline fit used smoothing factor s = %13.4e.\n", s);
  printf("Number of knots in each direction  = %5ld,%5ld.\n\n",
         spline.nx, spline.ny);
  printf("Sum of squared residuals           = %13.4e.\n", fp);
  fflush(stdout);

  /* Spline and its derivative to be evaluated on rectangular grid with
   * ngx*ngy points on the domain [xlo,xhi] by [ylo,yhi].
   */
  scanf("%ld%lf%lf%*[^\n]", &ngx, &xlo, &xhi);
  scanf("%ld%lf%lf%*[^\n]", &ngy, &ylo, &yhi);

  /* Allocations for spline evaluation.*/
  if (!(gridx = NAG_ALLOC((ngx), double)) ||
      !(gridy = NAG_ALLOC((ngy), double)) ||
      !(z = NAG_ALLOC((ngx * ngy), double)) ||
      !(zder = NAG_ALLOC((ngx * ngy), double))
      )
    {
      printf("Allocation failure\n");
      exit_status = -2;
      goto END;
    }

  delta = (xhi - xlo)/(double) (ngx - 1);
  gridx[0] = xlo;
  for (i = 1; i < ngx - 1; i++) gridx[i] = gridx[i-1] + delta;
  gridx[ngx-1] = xhi;

  delta = (yhi - ylo)/(double) (ngy - 1);
  gridy[0] = ylo;
  for (i = 1; i < ngy - 1; i++) gridy[i] = gridy[i-1] + delta;
  gridy[ngy-1] = yhi;

  /* Evaluate spline (nux=nuy=0) using
   * nag_2d_spline_deriv_rect (e02dhc).
   * Evaluation of spline surface at mesh of points with derivatives
   */
  nux = 0;
  nuy = 0;
  nag_2d_spline_deriv_rect(ngx, ngy, gridx, gridy, nux, nuy, z, &spline, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_2d_spline_deriv_rect (e02dhc))\n%s\n",
             fail.message);
      exit_status = 2;
      goto END;
    }

  /* Evaluate spline partial derivative of order (nux,nuy)*/
  scanf("%ld%ld%*[^\n]", &nux, &nuy);
  printf("\nDerivative of spline has order nux, nuy =%5ld, %5ld.\n",
         nux, nuy);
  fflush(stdout);
```

```
  /* nag_2d_spline_deriv_rect (e02dhc).
   * Evaluation of spline surface at mesh of points with derivatives
   */
  nag_2d_spline_deriv_rect(ngx, ngy, gridx, gridy, nux, nuy, zder, &spline,
                           &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_2d_spline_deriv_rect (e02dhc)\n%s\n",
             fail.message);
      exit_status = 3;
      goto END;
    }
  fflush(stdout);

  /* Print tabulated spline and derivative evaluations.*/
  print_spline(&ngx, gridx, &ngy, gridy, z, zder, &exit_status);

 END:
  NAG_FREE(f);
  NAG_FREE(gridx);
  NAG_FREE(gridy);
  NAG_FREE(x);
  NAG_FREE(y);
  NAG_FREE(z);
  NAG_FREE(zder);
  NAG_FREE(spline.lamda);
  NAG_FREE(spline.mu);
  NAG_FREE(spline.c);
  NAG_FREE(warmstartinf.nag_w);
  NAG_FREE(warmstartinf.nag_iw);
  return exit_status;
}

static void NAG_CALL print_spline(Integer *ngx, double *gridx, Integer *ngy,
                                  double *gridy, double *z, double *zder,
                                  Integer *exit_status)
{
  /* Print spline function and spline derivative evaluation*/
  Integer        indent = 0, ncols = 80;
  char           formc[] = "%8.3f";
  Integer        i;
  char           title[49];
  char           *outfile = 0;
  char           **clabsc = 0, **rlabsc = 0;
  Nag_OrderType  order;
  Nag_MatrixType matrixc;
  Nag_DiagType   diagc;
  Nag_LabelType  chlabelc;
  NagError       fail;

  INIT_FAIL(fail);

  /* Allocate for row and column label*/
  if (
    !(clabsc = NAG_ALLOC(*ngx, char *)) ||
    !(rlabsc = NAG_ALLOC(*ngy, char *))
    )
    {
      printf("Allocation failure\n");
      *exit_status = -3;
      goto END;
    }

  /* Allocate memory to clabsc and rlabsc elements and generate
   * column and row labels to print the results with.
   */
  for (i = 0; i < *ngx; i++)
    {
      clabsc[i] = NAG_ALLOC(11, char);
      sprintf(clabsc[i], "%5.2f%5s", gridx[i], "");
    }
```

```
  for (i = 0; i < *ngy; i++)
    {
      rlabsc[i] = NAG_ALLOC(11, char);
      sprintf(rlabsc[i], "%5.2f%5s", gridy[i], "");
    }



  order = Nag_ColMajor;
  matrixc = Nag_GeneralMatrix;
  diagc = Nag_NonUnitDiag;
  chlabelc = Nag_CharacterLabels;

  /* Print the spline evaluations, z. */
  strcpy(title, "Spline evaluated on X-Y grid (X across, Y down):");
  printf("\n");
  fflush(stdout);

  /* nag_gen_real_mat_print_comp (x04cbc).
   * Print real general matrix (comprehensive)
   */
  nag_gen_real_mat_print_comp(order, matrixc, diagc, *ngy, *ngx, z, *ngy,
                              formc,
                              title, chlabelc, (const char **) rlabsc,
                              chlabelc,
                              (const char **) clabsc, ncols,
                              indent, outfile,
                              &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_gen_real_mat_print_comp (x04cbc)\n%s\n",
             fail.message);
      *exit_status = 4;
      goto END;
    }

  /* Print the spline derivative evaluations, zder. */
  strcpy(title, "Spline derivative evaluated on X-Y grid:");
  printf("\n");
  fflush(stdout);

  nag_gen_real_mat_print_comp(order, matrixc, diagc, *ngy, *ngx, zder, *ngy,
                              formc, title, chlabelc, (const char **) rlabsc,
                              chlabelc, (const char **) clabsc, ncols, indent,
                              outfile, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_gen_real_mat_print_comp (x04cbc)\n%s\n",
             fail.message);
      *exit_status = 5;
      goto END;
    }

 END:
  for (i = 0; i < *ngy; i++) NAG_FREE(rlabsc[i]);
  NAG_FREE(rlabsc);
  for (i = 0; i < *ngx; i++) NAG_FREE(clabsc[i]);
  NAG_FREE(clabsc);
}
```

10.2 Program Data

```
nag_2d_spline_deriv_rect (e02dhc) Example Program Data
 11    9                                                    : mx, my
 0.0         0.5         1.0         1.5         2.0
 2.5         3.0         3.5         4.0         4.5
 5.0                                                        : x
 0.0         0.5         1.0         1.5         2.0
 2.5         3.0         3.5         4.0                  : y
 1.0000      0.88758     0.54030     0.070737   -0.41515
-0.80114    -0.97999    -0.93446    -0.65664
 1.5000      1.3564      0.82045     0.10611    -0.62422
-1.2317     -1.4850     -1.3047     -0.98547
 2.0600      1.7552      1.0806      0.15147    -0.83229
-1.6023     -1.9700     -1.8729     -1.4073
 2.5700      2.1240      1.3508      0.17684    -1.0404
-2.0029     -2.4750     -2.3511     -1.6741
 3.0000      2.6427      1.6309      0.21221    -1.2484
-2.2034     -2.9700     -2.8094     -1.9809
 3.5000      3.1715      1.8611      0.24458    -1.4565
-2.8640     -3.2650     -3.2776     -2.2878
 4.0400      3.5103      2.0612      0.28595   -1.6946
-3.2046     -3.9600     -3.7958     -2.6146
 4.5000      3.9391      2.4314      0.31632    -1.8627
-3.6351     -4.4550     -4.2141     -2.9314
 5.0400      4.3879      2.7515      0.35369    -2.0707
-4.0057     -4.9700     -4.6823     -3.2382
 5.5050      4.8367      2.9717      0.38505    -2.2888
-4.4033     -5.4450     -5.1405     -3.5950
 6.0000      5.2755      3.2418      0.42442    -2.4769
-4.8169     -5.9300     -5.6387     -3.9319                : f(x,y)
 0.1                                                        : s
 6           0.0         5.0                              : ngx, xlo, xhi
 5           0.0         4.0                              : ngy, ylo, yhi
 1           0                                            : nux, nuy
```

10.3 Program Results

```
nag_2d_spline_deriv_rect (e02dhc) Example Program Results

Spline fit used smoothing factor s =    1.0000e-01.
Number of knots in each direction  =    10,   13.

Sum of squared residuals           =    1.0004e-01.

Derivative of spline has order nux, nuy =    1,     0.

 Spline evaluated on X-Y grid (X across, Y down):
          0.00     1.00     2.00     3.00     4.00     5.00
 0.00    0.992    2.043    3.029    4.014    5.021    5.997
 1.00    0.541    1.088    1.607    2.142    2.705    3.239
 2.00   -0.417   -0.829   -1.241   -1.665   -2.083   -2.485
 3.00   -0.978   -1.975   -2.914   -3.913   -4.965   -5.924
 4.00   -0.648   -1.363   -1.991   -2.606   -3.251   -3.933

 Spline derivative evaluated on X-Y grid:
          0.00     1.00     2.00     3.00     4.00     5.00
 0.00    1.093    1.013    0.970    1.004    1.001    0.939
 1.00    0.565    0.531    0.515    0.558    0.559    0.499
 2.00   -0.429   -0.404   -0.421   -0.423   -0.412   -0.389
 3.00   -1.060   -0.951   -0.949   -1.048   -1.031   -0.861
 4.00   -0.779   -0.661   -0.608   -0.628   -0.663   -0.701
```