NAG Library Function Document

nag_check_derivs (c05zdc)

1 Purpose

nag_check_derivs (c05zdc) checks the user-supplied gradients of a set of nonlinear functions in several variables, for consistency with the functions themselves. The function must be called twice.

2 Specification

```
#include <nag.h>
#include <nagc05.h>
void nag_check_derivs (Integer mode, Integer m, Integer n, const double x[],
     const double fvec[], const double fjac[], double xp[],
     const double fvecp[], double err[], NagError *fail)
```

3 Description

nag_check_derivs (c05zdc) is based on the MINPACK routine CHKDER (see Moré *et al.* (1980)). It checks the ith gradient for consistency with the ith function by computing a forward-difference approximation along a suitably chosen direction and comparing this approximation with the user-supplied gradient along the same direction. The principal characteristic of nag_check_derivs (c05zdc) is its invariance under changes in scale of the variables or functions.

4 References

Moré J J, Garbow B S and Hillstrom K E (1980) User guide for MINPACK-1 *Technical Report ANL-80-74* Argonne National Laboratory

5 Arguments

1: **mode** – Integer *Input*

On entry: the value 1 on the first call and the value 2 on the second call of nag_check_derivs (c05zdc).

Constraint: **mode** $= 1$ or 2.

2: **m** – Integer *Input*

On entry: m, the number of functions.

Constraint: **m** ≥ 1.

3: **n** – Integer *Input*

On entry: n, the number of variables. For use with nag_zero_nonlin_eqns_deriv_easy (c05rbc), nag_zero_nonlin_eqns_deriv_expert (c05rcc) and nag_zero_nonlin_eqns_deriv_rcomm (c05rdc), **m** $=$ **n**.

Constraint: **n** ≥ 1.

4: **x**[**n**] – const double *Input*

On entry: the components of a point x, at which the consistency check is to be made. (See Section 7.)

5: **fvec**[**m**] – const double *Input*

On entry: if **mode** $= 2$, **fvec** must contain the value of the functions evaluated at x. If **mode** $= 1$, **fvec** is not referenced.

6: **fjac**[**m** $\times$ **n**] – const double *Input*

Note: the (i, j)th element of the matrix is stored in **fjac**$[(j-1) \times \mathbf{m} + i - 1]$.

On entry: if **mode** $= 2$, **fjac** must contain the value of $\dfrac{\partial f_i}{\partial x_j}$ at the point x, for $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$. If **mode** $= 1$, **fjac** is not referenced.

7: **xp**[**n**] – double *Output*

On exit: if **mode** $= 1$, **xp** is set to a point neighbouring **x**. If **mode** $= 2$, **xp** is undefined.

8: **fvecp**[**m**] – const double *Input*

On entry: if **mode** $= 2$, **fvecp** must contain the value of the functions evaluated at **xp** (as output by a preceding call to nag_check_derivs (c05zdc) with **mode** $= 1$). If **mode** $= 1$, **fvecp** is not referenced.

9: **err**[**m**] – double *Output*

On exit: if **mode** $= 2$, **err** contains measures of correctness of the respective gradients. If **mode** $= 1$, **err** is undefined. If there is no loss of significance (see Section 7), then if **err**$[i-1]$ is 1.0 the ith user-supplied gradient $\dfrac{\partial f_i}{\partial x_j}$, for $j = 1, 2, \ldots, n$ is correct, whilst if **err**$[i-1]$ is 0.0 the ith gradient is incorrect. For values of **err**$[i-1]$ between 0.0 and 1.0 the categorisation is less certain. In general, a value of **err**$[i-1] > 0.5$ indicates that the ith gradient is probably correct.

10: **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.
See Section 3.2.1.2 in the Essential Introduction for further information.

NE_BAD_PARAM

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

NE_INT

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

On entry, **mode** $=$ ⟨*value*⟩.
Constraint: **mode** $= 1$ or 2.

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

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.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in the Essential Introduction for further information.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in the Essential Introduction for further information.

7 Accuracy

nag_check_derivs (c05zdc) does not perform reliably if cancellation or rounding errors cause a severe loss of significance in the evaluation of a function. Therefore, none of the components of x should be unusually small (in particular, zero) or any other value which may cause loss of significance. The relative differences between corresponding elements of **fvecp** and **fvec** should be at least two orders of magnitude greater than the ***machine precision*** returned by nag_machine_precision (X02AJC).

8 Parallelism and Performance

Not applicable.

9 Further Comments

The time required by nag_check_derivs (c05zdc) increases with **m** and **n**.

10 Example

This example checks the Jacobian matrix for a problem with 15 functions of 3 variables (sometimes referred to as the Bard problem).

10.1 Program Text

```
/* nag_check_derivs (c05zdc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nagc05.h>

#ifdef __cplusplus
extern "C" {
#endif
static void NAG_CALL f(Integer m, Integer n, double x[], double fvec[],
                       double fjac[], Integer iflag);
#ifdef __cplusplus
}
#endif

int main(void)
{
  Integer  exit_status = 0, j, m, n, mode, iflag, err_detected;
  NagError fail;
  double   *fjac = 0, *fvec = 0, *x = 0, *xp = 0, *fvecp = 0, *err = 0;
  INIT_FAIL(fail);

  printf("nag_check_derivs (c05zdc) Example Program Results\n");
  n = 3;
  m = n;

  if (n > 0)
```

```
  {
    if (!(fjac = NAG_ALLOC(m*n, double)) ||
        !(fvec = NAG_ALLOC(m, double)) ||
        !(fvecp = NAG_ALLOC(m, double)) ||
        !(err = NAG_ALLOC(m, double)) ||
        !(x = NAG_ALLOC(n, double)) ||
        !(xp = NAG_ALLOC(n, double)))
      {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
      }
  }
else
  {
    printf("Invalid n.\n");
    exit_status = 1;
    goto END;
  }

/* Set up an arbitrary point at which to check the 1st derivatives */
x[0] = 9.2e-01;
x[1] = 1.3e-01;
x[2] = 5.4e-01;

/* nag_check_derivs (c05zdc).
 * Derivative checker for user-supplied Jacobian
 */

mode = 1;
nag_check_derivs(mode, m, n, x, fvec, fjac, xp, fvecp, err, &fail);

if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_check_derivs (c05zdc).\n%s\n",
            fail.message);
    exit_status = 1;
    goto END;
  }

/* Evaluate at the original point x and the update point xp */
/* Get fvec, the functions at x */
iflag = 1;
f(m, n, x, fvec, fjac, iflag);

/* Get fvecp, the functions at xp */
iflag = 1;
f(m, n, xp, fvecp, fjac, iflag);

/* Get fjac, the Jacobian at x */
iflag = 2;
f(m, n, x, fvec, fjac, iflag);

mode = 2;
nag_check_derivs(mode, m, n, x, fvec, fjac, xp, fvecp, err, &fail);

if (fail.code != NE_NOERROR)
  {
    printf("Error from nag_check_derivs (c05zdc).\n%s\n",
            fail.message);
    exit_status = 1;
    goto END;
  }

printf("\nAt point ");
for (j = 0; j < n; ++j)
  printf("%13.5e", x[j]);
printf(",\n");

err_detected = 0;
```

```
  for (j = 0; j < n; ++j)
    {

      if (err[j] <= 0.5)
        {
          printf("suspicious gradient number %"NAG_IFMT
                 " with error measure %13.5e\n", j, err[j]);
          err_detected = 1;
        }

    }

  if (!err_detected)
    {
      printf("gradients appear correct\n");
    }

 END:
  NAG_FREE(fjac);
  NAG_FREE(fvec);
  NAG_FREE(fvecp);
  NAG_FREE(err);
  NAG_FREE(x);
  NAG_FREE(xp);
  return exit_status;
}

static void NAG_CALL f(Integer m, Integer n, double x[], double fvec[],
                       double fjac[], Integer iflag)
{
  Integer j, k;

  if (iflag == 1)
    {
      /* Calculate the function values */
      for (k = 0; k < m; k++)
        {
          fvec[k] = (3.0-x[k]*2.0) * x[k] + 1.0;
          if (k > 0) fvec[k] -= x[k-1];
          if (k < m-1) fvec[k] -= x[k+1] * 2.0;
        }
    }
  else if (iflag == 2)
    {
      /* Calculate the corresponding first derivatives */
      for (k = 0; k < m; k++)
        {
          for (j = 0; j < n; j++)
            fjac[j*m + k] = 0.0;
          fjac[k*m + k] = 3.0 - x[k] * 4.0;
          if (k > 0)
            fjac[(k-1)*m + k] = -1.0;
          if (k < m-1)
            fjac[(k+1)*m + k] = -2.0;
        }
    }
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_check_derivs (c05zdc) Example Program Results

At point   9.20000e-01  1.30000e-01  5.40000e-01,
gradients appear correct
```