NAG Library Function Document

nag_pde_parab_1d_keller (d03pec)

1 Purpose

nag_pde_parab_1d_keller (d03pec) integrates a system of linear or nonlinear, first-order, time-dependent partial differential equations (PDEs) in one space variable. The spatial discretization is performed using the Keller box scheme and the method of lines is employed to reduce the PDEs to a system of ordinary differential equations (ODEs). The resulting system is solved using a Backward Differentiation Formula (BDF) method.

2 Specification

```
#include <nag.h>
#include <nagd03.h>
```

```
void nag_pde_parab_1d_keller (Integer npde, double *ts, double tout,
     void (*pdedef)(Integer npde, double t, double x, const double u[],
          const double ut[], const double ux[], double res[], Integer *ires,
          Nag_Comm *comm),
     void (*bndary)(Integer npde, double t, Integer ibnd, Integer nobc,
          const double u[], const double ut[], double res[], Integer *ires,
          Nag_Comm *comm),
     double u[], Integer npts, const double x[], Integer nleft, double acc,
     double rsave[], Integer lrsave, Integer isave[], Integer lisave,
     Integer itask, Integer itrace, const char *outfile, Integer *ind,
     Nag_Comm *comm, Nag_D03_Save *saved, NagError *fail)
```

3 Description

nag_pde_parab_1d_keller (d03pec) integrates the system of first-order PDEs

$$G_i(x,t,U,U_x,U_t)=0, \quad i=1,2,\dots,\mathbf{npde}. \tag{1}$$

In particular the functions G_i must have the general form

$$G_i=\sum_{j=1}^{\mathbf{npde}} P_{i,j}\frac{\partial U_j}{\partial t}+Q_i, \quad i=1,2,\dots,\mathbf{npde}, \quad a\le x\le b, t\ge t_0, \tag{2}$$

where $P_{i,j}$ and Q_i depend on x, t, U, U_x and the vector U is the set of solution values

$$U(x,t)=\left[U_1(x,t),\dots,U_{\mathbf{npde}}(x,t)\right]^{\mathrm{T}}, \tag{3}$$

and the vector U_x is its partial derivative with respect to x. Note that $P_{i,j}$ and Q_i must not depend on $\frac{\partial U}{\partial t}$.

The integration in time is from t_0 to t_{out}, over the space interval $a\le x\le b$, where $a=x_1$ and $b=x_{\mathbf{npts}}$ are the leftmost and rightmost points of a user-defined mesh $x_1,x_2,\dots,x_{\mathbf{npts}}$. The mesh should be chosen in accordance with the expected behaviour of the solution.

The PDE system which is defined by the functions G_i must be specified in **pdedef**.

The initial values of the functions $U(x,t)$ must be given at $t=t_0$. For a first-order system of PDEs, only one boundary condition is required for each PDE component U_i. The **npde** boundary conditions are separated into n_a at the left-hand boundary $x=a$, and n_b at the right-hand boundary $x=b$, such that $n_a+n_b=\mathbf{npde}$. The position of the boundary condition for each component should be chosen with care; the general rule is that if the characteristic direction of U_i at the left-hand boundary (say) points into the interior of the solution domain, then the boundary condition for U_i should be specified at the left-hand boundary. Incorrect positioning of boundary conditions generally results in initialization or integration difficulties in the underlying time integration functions.

The boundary conditions have the form:

$$G_i^L(x,t,U,U_t) = 0 \quad \text{at } x = a, \quad i = 1,2,\ldots,n_a \tag{4}$$

at the left-hand boundary, and

$$G_i^R(x,t,U,U_t) = 0 \quad \text{at } x = b, \quad i = 1,2,\ldots,n_b \tag{5}$$

at the right-hand boundary.

Note that the functions G_i^L and G_i^R must not depend on U_x, since spatial derivatives are not determined explicitly in the Keller box scheme (see Keller (1970)). If the problem involves derivative (Neumann) boundary conditions then it is generally possible to restate such boundary conditions in terms of permissible variables. Also note that G_i^L and G_i^R must be linear with respect to time derivatives, so that the boundary conditions have the general form

$$\sum_{j=1}^{\mathbf{npde}} E_{i,j}^L \frac{\partial U_j}{\partial t} + S_i^L = 0, \quad i = 1,2,\ldots,n_a \tag{6}$$

at the left-hand boundary, and

$$\sum_{j=1}^{\mathbf{npde}} E_{i,j}^R \frac{\partial U_j}{\partial t} + S_i^R = 0, \quad i = 1,2,\ldots,n_b \tag{7}$$

at the right-hand boundary, where $E_{i,j}^L$, $E_{i,j}^R$, S_i^L, and S_i^R depend on x, t and U only.

The boundary conditions must be specified in **bndary**.

The problem is subject to the following restrictions:

(i) $t_0 < t_{\mathrm{out}}$, so that integration is in the forward direction;

(ii) $P_{i,j}$ and Q_i must not depend on any time derivatives;

(iii) The evaluation of the function G_i is done at the mid-points of the mesh intervals by calling the **pdedef** for each mid-point in turn. Any discontinuities in the function **must** therefore be at one or more of the mesh points $x_1, x_2, \ldots, x_{\mathbf{npts}}$;

(iv) At least one of the functions $P_{i,j}$ must be nonzero so that there is a time derivative present in the problem.

In this method of lines approach the Keller box scheme (see Keller (1970)) is applied to each PDE in the space variable only, resulting in a system of ODEs in time for the values of U_i at each mesh point. In total there are **npde** $\times$ **npts** ODEs in the time direction. This system is then integrated forwards in time using a BDF method.

4 References

Berzins M (1990) Developments in the NAG Library software for parabolic equations *Scientific Software Systems* (eds J C Mason and M G Cox) 59–72 Chapman and Hall

Berzins M, Dew P M and Furzeland R M (1989) Developing software for time-dependent problems using the method of lines and differential-algebraic integrators *Appl. Numer. Math.* **5** 375–397

Keller H B (1970) A new difference scheme for parabolic problems *Numerical Solutions of Partial Differential Equations* (ed J Bramble) **2** 327–350 Academic Press

Pennington S V and Berzins M (1994) New NAG Library software for first-order partial differential equations *ACM Trans. Math. Softw.* **20** 63–99

5 Arguments

1: **npde** – Integer *Input*

On entry: the number of PDEs in the system to be solved.

Constraint: **npde** ≥ 1.

2: **ts** – double * *Input/Output*

On entry: the initial value of the independent variable t.

Constraint: **ts** $<$ **tout**.

On exit: the value of t corresponding to the solution values in **u**. Normally **ts** $=$ **tout**.

3: **tout** – double *Input*

On entry: the final value of t to which the integration is to be carried out.

4: **pdedef** – function, supplied by the user *External Function*

pdedef must compute the functions G_i which define the system of PDEs. **pdedef** is called approximately midway between each pair of mesh points in turn by nag_pde_parab_1d_keller (d03pec).

The specification of **pdedef** is:

```
void pdedef (Integer npde, double t, double x, const double u[],
     const double ut[], const double ux[], double res[],
     Integer *ires, Nag_Comm *comm)
```

1: **npde** – Integer *Input*

On entry: the number of PDEs in the system.

2: **t** – double *Input*

On entry: the current value of the independent variable t.

3: **x** – double *Input*

On entry: the current value of the space variable x.

4: **u**[**npde**] – const double *Input*

On entry: **u**$[i-1]$ contains the value of the component $U_i(x,t)$, for $i = 1, 2, \ldots,$ **npde**.

5: **ut**[**npde**] – const double *Input*

On entry: **ut**$[i-1]$ contains the value of the component $\frac{\partial U_i(x,t)}{\partial t}$, for $i = 1, 2, \ldots,$ **npde**.

6: **ux**[**npde**] – const double *Input*

On entry: **ux**$[i-1]$ contains the value of the component $\frac{\partial U_i(x,t)}{\partial x}$, for $i = 1, 2, \ldots,$ **npde**.

7: **res**[**npde**] – double *Output*

On exit: **res**$[i-1]$ must contain the ith component of G, for $i = 1, 2, \ldots,$ **npde**, where G is defined as

$$G_i = \sum_{j=1}^{\mathbf{npde}} P_{i,j} \frac{\partial U_j}{\partial t}, \tag{8}$$

i.e., only terms depending explicitly on time derivatives, or

$$G_i = \sum_{j=1}^{\mathbf{npde}} P_{i,j} \frac{\partial U_j}{\partial t} + Q_i, \tag{9}$$

i.e., all terms in equation (2).

The definition of G is determined by the input value of **ires**.

8: **ires** – Integer * *Input/Output*

On entry: the form of G_i that must be returned in the array **res**.

ires $= -1$
Equation (8) must be used.

ires $= 1$
Equation (9) must be used.

On exit: should usually remain unchanged. However, you may set **ires** to force the integration function to take certain actions, as described below:

ires $= 2$
Indicates to the integrator that control should be passed back immediately to the calling function with the error indicator set to **fail.code** = NE_USER_STOP.

ires $= 3$
Indicates to the integrator that the current time step should be abandoned and a smaller time step used instead. You may wish to set **ires** $= 3$ when a physically meaningless input or output value has been generated. If you consecutively set **ires** $= 3$, then nag_pde_parab_1d_keller (d03pec) returns to the calling function with the error indicator set to **fail.code** = NE_FAILED_DERIV.

9: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to **pdedef**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_pde_parab_1d_keller (d03pec) you may allocate memory and initialize these pointers with various quantities for use by **pdedef** when called from nag_pde_parab_1d_keller (d03pec) (see Section 3.2.1.1 in the Essential Introduction).

5: **bndary** – function, supplied by the user *External Function*

bndary must compute the functions G_i^L and G_i^R which define the boundary conditions as in equations (4) and (5).

The specification of **bndary** is:

```
void bndary (Integer npde, double t, Integer ibnd, Integer nobc,
     const double u[], const double ut[], double res[], Integer *ires,
     Nag_Comm *comm)
```

1: **npde** – Integer *Input*

On entry: the number of PDEs in the system.

2: **t** – double *Input*

On entry: the current value of the independent variable t.

3: **ibnd** – Integer *Input*

On entry: determines the position of the boundary conditions.

ibnd $= 0$
: **bndary** must compute the left-hand boundary condition at $x = a$.

ibnd $\neq 0$
: Indicates that **bndary** must compute the right-hand boundary condition at $x = b$.

4: **nobc** – Integer *Input*

On entry: specifies the number of boundary conditions at the boundary specified by **ibnd**.

5: **u**[**npde**] – const double *Input*

On entry: **u**$[i-1]$ contains the value of the component $U_i(x,t)$ at the boundary specified by **ibnd**, for $i = 1, 2, \ldots,$ **npde**.

6: **ut**[**npde**] – const double *Input*

On entry: **ut**$[i-1]$ contains the value of the component $\frac{\partial U_i(x,t)}{\partial t}$ at the boundary specified by **ibnd**, for $i = 1, 2, \ldots,$ **npde**.

7: **res**[**nobc**] – double *Output*

On exit: **res**$[i-1]$ must contain the ith component of G^L or G^R, depending on the value of **ibnd**, for $i = 1, 2, \ldots,$ **nobc**, where G^L is defined as

$$G_i^L = \sum_{j=1}^{\mathbf{npde}} E_{i,j}^L \frac{\partial U_j}{\partial t}, \tag{10}$$

i.e., only terms depending explicitly on time derivatives, or

$$G_i^L = \sum_{j=1}^{\mathbf{npde}} E_{i,j}^L \frac{\partial U_j}{\partial t} + S_i^L, \tag{11}$$

i.e., all terms in equation (6), and similarly for G_i^R.
The definitions of G^L and G^R are determined by the input value of **ires**.

8: **ires** – Integer * *Input/Output*

On entry: the form G_i^L (or G_i^R) that must be returned in the array **res**.

ires $= -1$
: Equation (10) must be used.

ires $= 1$
: Equation (11) must be used.

On exit: should usually remain unchanged. However, you may set **ires** to force the integration function to take certain actions, as described below:

ires $= 2$
: Indicates to the integrator that control should be passed back immediately to the calling function with the error indicator set to **fail.code** = NE_USER_STOP.

ires $= 3$
Indicates to the integrator that the current time step should be abandoned and a smaller time step used instead. You may wish to set **ires** $= 3$ when a physically meaningless input or output value has been generated. If you consecutively set **ires** $= 3$, then nag_pde_parab_1d_keller (d03pec) returns to the calling function with the error indicator set to **fail.code** = NE_FAILED_DERIV.

9: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to **bndary**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_pde_parab_1d_keller (d03pec) you may allocate memory and initialize these pointers with various quantities for use by **bndary** when called from nag_pde_parab_1d_keller (d03pec) (see Section 3.2.1.1 in the Essential Introduction).

6: **u**[**npde** × **npts**] – double *Input/Output*

On entry: the initial values of $U(x,t)$ at $t =$ **ts** and the mesh points **x**[$j-1$], for $j = 1, 2, \ldots,$ **npts**.

On exit: **u**[**npde** $\times (j-1) + i - 1$] will contain the computed solution at $t =$ **ts**.

7: **npts** – Integer *Input*

On entry: the number of mesh points in the interval $[a, b]$.

Constraint: **npts** ≥ 3.

8: **x**[**npts**] – const double *Input*

On entry: the mesh points in the spatial direction. **x**[0] must specify the left-hand boundary, a, and **x**[**npts** $- 1$] must specify the right-hand boundary, b.

Constraint: **x**[0] < **x**[1] < ⋯ < **x**[**npts** $- 1$].

9: **nleft** – Integer *Input*

On entry: the number n_a of boundary conditions at the left-hand mesh point **x**[0].

Constraint: $0 \leq$ **nleft** $\leq$ **npde**.

10: **acc** – double *Input*

On entry: a positive quantity for controlling the local error estimate in the time integration. If $E(i,j)$ is the estimated error for U_i at the jth mesh point, the error test is:

$$|E(i,j)| = \mathbf{acc} \times (1.0 + |\mathbf{u}[\mathbf{npde} \times (j-1) + i - 1]|).$$

Constraint: **acc** > 0.0.

11: **rsave**[**lrsave**] – double *Communication Array*

If **ind** $= 0$, **rsave** need not be set on entry.

If **ind** $= 1$, **rsave** must be unchanged from the previous call to the function because it contains required information about the iteration.

12: **lrsave** – Integer *Input*

On entry: the dimension of the array **rsave**.

Constraint: **lrsave** $\geq (4 \times$ **npde** $+$ **nleft** $+ 14) \times$ **npde** $\times$ **npts** $+ (3 \times$ **npde** $+ 21) \times$ **npde** $+$
$7 \times$ **npts** $+ 54$.

13: **isave**[**lisave**] – Integer *Communication Array*

If **ind** $= 0$, **isave** need not be set on entry.

If **ind** $= 1$, **isave** must be unchanged from the previous call to the function because it contains required information about the iteration. In particular:

isave[0]
: Contains the number of steps taken in time.

isave[1]
: Contains the number of residual evaluations of the resulting ODE system used. One such evaluation involves computing the PDE functions at all the mesh points, as well as one evaluation of the functions in the boundary conditions.

isave[2]
: Contains the number of Jacobian evaluations performed by the time integrator.

isave[3]
: Contains the order of the last backward differentiation formula method used.

isave[4]
: Contains the number of Newton iterations performed by the time integrator. Each iteration involves an ODE residual evaluation followed by a back-substitution using the *LU* decomposition of the Jacobian matrix.

14: **lisave** – Integer *Input*

On entry: the dimension of the array **isave**.

Constraint: **lisave** $\geq$ **npde** $\times$ **npts** $+ 24$.

15: **itask** – Integer *Input*

On entry: specifies the task to be performed by the ODE integrator.

itask $= 1$
: Normal computation of output values **u** at $t =$ **tout**.

itask $= 2$
: Take one step and return.

itask $= 3$
: Stop at the first internal integration point at or beyond $t =$ **tout**.

Constraint: **itask** $= 1$, 2 or 3.

16: **itrace** – Integer *Input*

On entry: the level of trace information required from nag_pde_parab_1d_keller (d03pec) and the underlying ODE solver as follows:

itrace ≤ -1
: No output is generated.

itrace $= 0$
: Only warning messages from the PDE solver are printed .

itrace $= 1$
Output from the underlying ODE solver is printed . This output contains details of Jacobian entries, the nonlinear iteration and the time integration during the computation of the ODE system.

itrace $= 2$
Output from the underlying ODE solver is similar to that produced when **itrace** $= 1$, except that the advisory messages are given in greater detail.

itrace ≥ 3
Output from the underlying ODE solver is similar to that produced when **itrace** $= 2$, except that the advisory messages are given in greater detail.

You are advised to set **itrace** $= 0$.

17: **outfile** – const char * *Input*

On entry: the name of a file to which diagnostic output will be directed. If **outfile** is **NULL** the diagnostic output will be directed to standard output.

18: **ind** – Integer * *Input/Output*

On entry: indicates whether this is a continuation call or a new integration.

ind $= 0$
Starts or restarts the integration in time.

ind $= 1$
Continues the integration after an earlier exit from the function. In this case, only the arguments **tout** and **fail** should be reset between calls to nag_pde_parab_1d_keller (d03pec).

Constraint: **ind** $= 0$ or 1.

On exit: **ind** $= 1$.

19: **comm** – Nag_Comm *

The NAG communication argument (see Section 3.2.1.1 in the Essential Introduction).

20: **saved** – Nag_D03_Save * *Communication Structure*

saved must remain unchanged following a previous call to a Chapter d03 function and prior to any subsequent call to a Chapter d03 function.

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

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

6 Error Indicators and Warnings

NE_ACC_IN_DOUBT

Integration completed, but a small change in **acc** is unlikely to result in a changed solution.
acc $= \langle value \rangle$.

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 $\langle value \rangle$ had an illegal value.

NE_FAILED_DERIV

In setting up the ODE system an internal auxiliary was unable to initialize the derivative. This could be due to your setting **ires** $= 3$ in **pdedef** or **bndary**.

NE_FAILED_START

acc was too small to start integration: **acc** $= \langle value \rangle$.

NE_FAILED_STEP

Error during Jacobian formulation for ODE system. Increase **itrace** for further details.

Repeated errors in an attempted step of underlying ODE solver. Integration was successful as far as **ts**: **ts** $= \langle value \rangle$.

Underlying ODE solver cannot make further progress from the point **ts** with the supplied value of **acc**. **ts** $= \langle value \rangle$, **acc** $= \langle value \rangle$.

NE_INT

ires set to an invalid value in call to **pdedef** or **bndary**.

On entry, **ind** $= \langle value \rangle$.
Constraint: **ind** $= 0$ or 1.

On entry, **itask** $= \langle value \rangle$.
Constraint: **itask** $= 1$, 2 or 3.

On entry, **nleft** $= \langle value \rangle$.
Constraint: **nleft** ≥ 0.

On entry, **npde** $= \langle value \rangle$.
Constraint: **npde** ≥ 1.

On entry, **npts** $= \langle value \rangle$.
Constraint: **npts** ≥ 3.

NE_INT_2

On entry, **lisave** is too small: **lisave** $= \langle value \rangle$. Minimum possible dimension: $\langle value \rangle$.

On entry, **lrsave** is too small: **lrsave** $= \langle value \rangle$. Minimum possible dimension: $\langle value \rangle$.

On entry, **nleft** $= \langle value \rangle$, **npde** $= \langle value \rangle$.
Constraint: **nleft** $\leq$ **npde**.

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.

Serious error in internal call to an auxiliary. Increase **itrace** for further details.

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.

NE_NOT_CLOSE_FILE

Cannot close file $\langle value \rangle$.

NE_NOT_STRICTLY_INCREASING

On entry, mesh points **x** appear to be badly ordered: $I = \langle value \rangle$, $\mathbf{x}[I-1] = \langle value \rangle$, $J = \langle value \rangle$ and $\mathbf{x}[J-1] = \langle value \rangle$.

NE_NOT_WRITE_FILE

Cannot open file $\langle value \rangle$ for writing.

NE_REAL

On entry, **acc** $= \langle value \rangle$.
Constraint: **acc** > 0.0.

NE_REAL_2

On entry, **tout** $= \langle value \rangle$ and **ts** $= \langle value \rangle$.
Constraint: **tout** $>$ **ts**.

On entry, **tout** $-$ **ts** is too small: **tout** $= \langle value \rangle$ and **ts** $= \langle value \rangle$.

NE_SING_JAC

Singular Jacobian of ODE system. Check problem formulation.

NE_USER_STOP

In evaluating residual of ODE system, **ires** $= 2$ has been set in **pdedef** or **bndary**. Integration is successful as far as **ts**: **ts** $= \langle value \rangle$.

7 Accuracy

nag_pde_parab_1d_keller (d03pec) controls the accuracy of the integration in the time direction but not the accuracy of the approximation in space. The spatial accuracy depends on both the number of mesh points and on their distribution in space. In the time integration only the local error over a single step is controlled and so the accuracy over a number of steps cannot be guaranteed. You should therefore test the effect of varying the accuracy argument, **acc**.

8 Parallelism and Performance

nag_pde_parab_1d_keller (d03pec) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_pde_parab_1d_keller (d03pec) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

The Keller box scheme can be used to solve higher-order problems which have been reduced to first-order by the introduction of new variables (see the example problem in nag_pde_parab_1d_keller_ode (d03pkc)). In general, a second-order problem can be solved with slightly greater accuracy using the Keller box scheme instead of a finite difference scheme (nag_pde_parab_1d_fd (d03pcc) or nag_pde_parab_1d_fd_ode (d03phc) for example), but at the expense of increased CPU time due to the larger number of function evaluations required.

It should be noted that the Keller box scheme, in common with other central-difference schemes, may be unsuitable for some hyperbolic first-order problems such as the apparently simple linear advection equation $U_t + aU_x = 0$, where a is a constant, resulting in spurious oscillations due to the lack of dissipation. This type of problem requires a discretization scheme with upwind weighting (nag_pde_parab_1d_cd (d03pfc) for example), or the addition of a second-order artificial dissipation term.

The time taken depends on the complexity of the system and on the accuracy requested.

10 Example

This example is the simple first-order system

$$\frac{\partial U_1}{\partial t} + \frac{\partial U_1}{\partial x} + \frac{\partial U_2}{\partial x} = 0,$$

$$\frac{\partial U_2}{\partial t} + 4\frac{\partial U_1}{\partial x} + \frac{\partial U_2}{\partial x} = 0,$$

for $t \in [0, 1]$ and $x \in [0, 1]$.

The initial conditions are

$$U_1(x, 0) = \exp(x), \quad U_2(x, 0) = \sin(x),$$

and the Dirichlet boundary conditions for U_1 at $x = 0$ and U_2 at $x = 1$ are given by the exact solution:

$$U_1(x, t) = \tfrac{1}{2}\{\exp(x + t) + \exp(x - 3t)\} + \tfrac{1}{4}\{\sin(x - 3t) - \sin(x + t)\},$$

$$U_2(x, t) = \exp(x - 3t) - \exp(x + t) + \tfrac{1}{2}\{\sin(x + t) + \sin(x - 3t)\}.$$

10.1 Program Text

```
/* nag_pde_parab_1d_keller (d03pec) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagd03.h>
#include <nagx01.h>

#ifdef __cplusplus
extern "C" {
#endif
static void NAG_CALL pdedef(Integer, double, double, const double[],
                            const double[], const double[], double[],
                            Integer *, Nag_Comm *);
static void NAG_CALL bndary(Integer, double, Integer, Integer, const double[],
                            const double[], double[], Integer *, Nag_Comm *);
static void NAG_CALL exact(double, Integer, Integer, double *, double *);
static void NAG_CALL uinit(Integer, Integer, double *, double *);
#ifdef __cplusplus
}
#endif

#define U(I, J)  u[npde*((J) -1)+(I) -1]
#define EU(I, J) eu[npde*((J) -1)+(I) -1]

int main(void)
{
  const Integer npde = 2, npts = 41, nleft = 1, neqn = npde*npts;
  const Integer lisave = neqn+24, nwkres = npde*(npts+21+3*npde)+7*npts+4;
  const Integer lrsave = 11*neqn+(4*npde+nleft+2)*neqn+50+nwkres;
  static double ruser[2] = {-1.0, -1.0};
  Integer       exit_status = 0, i, ind, it, itask, itrace;
```

```
  double        acc, tout, ts;
  double        *eu = 0, *rsave = 0, *u = 0, *x = 0;
  Integer       *isave = 0;
  NagError      fail;
  Nag_Comm      comm;
  Nag_D03_Save  saved;

  INIT_FAIL(fail);

  printf(
          "nag_pde_parab_1d_keller (d03pec) Example Program Results\n\n");

  /* For communication with user-supplied functions: */
  comm.user = ruser;

  /* Allocate memory */

  if (!(eu = NAG_ALLOC(npde*npts, double)) ||
      !(rsave = NAG_ALLOC(lrsave, double)) ||
      !(u = NAG_ALLOC(npde*npts, double)) ||
      !(x = NAG_ALLOC(npts, double)) ||
      !(isave = NAG_ALLOC(lisave, Integer)))
    {
      printf("Allocation failure\n");
      exit_status = 1;
      goto END;
    }

  itrace = 0;
  acc = 1e-6;

  printf("  Accuracy requirement =%12.3e", acc);
  printf(" Number of points = %3"NAG_IFMT"\n\n", npts);

  /* Set spatial-mesh points */

  for (i = 0; i < npts; ++i) x[i] = i/(npts-1.0);

  printf(" x        ");
  printf("%10.4f%10.4f%10.4f%10.4f%10.4f\n\n",
          x[4], x[12], x[20], x[28], x[36]);

  ind = 0;
  itask = 1;

  uinit(npde, npts, x, u);

  /* Loop over output value of t */

  ts = 0.0;
  tout = 0.0;
  for (it = 0; it < 5; ++it)
    {
      tout = 0.2*(it+1);
      /* nag_pde_parab_1d_keller (d03pec).
       * General system of first-order PDEs, method of lines,
       * Keller box discretisation, one space variable
       */
      nag_pde_parab_1d_keller(npde, &ts, tout, pdedef, bndary, u, npts, x,
                              nleft, acc, rsave, lrsave, isave, lisave, itask,
                              itrace, 0, &ind, &comm, &saved, &fail);

      if (fail.code != NE_NOERROR)
        {
          printf("Error from nag_pde_parab_1d_keller (d03pec).\n%s\n",
                  fail.message);
          exit_status = 1;
          goto END;
        }

      /* Check against the exact solution */
```

```
      exact(tout, npde, npts, x, eu);

      printf(" t = %5.2f\n", ts);
      printf(" Approx u1");
      printf("%10.4f%10.4f%10.4f%10.4f%10.4f\n",
              U(1, 5), U(1, 13), U(1, 21), U(1, 29), U(1, 37));

      printf(" Exact  u1");
      printf("%10.4f%10.4f%10.4f%10.4f%10.4f\n",
              EU(1, 5), EU(1, 13), EU(1, 21), EU(1, 29), EU(1, 37));

      printf(" Approx u2");
      printf("%10.4f%10.4f%10.4f%10.4f%10.4f\n",
              U(2, 5), U(2, 13), U(2, 21), U(2, 29), U(2, 37));

      printf(" Exact  u2");
      printf("%10.4f%10.4f%10.4f%10.4f%10.4f\n\n",
              EU(2, 5), EU(2, 13), EU(2, 21), EU(2, 29), EU(2, 37));
    }
  printf(" Number of integration steps in time = %6"NAG_IFMT"\n", isave[0]);
  printf(" Number of function evaluations = %6"NAG_IFMT"\n", isave[1]);
  printf(" Number of Jacobian evaluations =%6"NAG_IFMT"\n", isave[2]);
  printf(" Number of iterations = %6"NAG_IFMT"\n\n", isave[4]);

 END:
  NAG_FREE(eu);
  NAG_FREE(rsave);
  NAG_FREE(u);
  NAG_FREE(x);
  NAG_FREE(isave);

  return exit_status;
}



static void NAG_CALL pdedef(Integer npde, double t, double x, const double u[],
                            const double udot[], const double dudx[], double
                            res[], Integer *ires, Nag_Comm *comm)
{
  if (comm->user[0] == -1.0)
    {
      printf("(User-supplied callback pdedef, first invocation.)\n");
      comm->user[0] = 0.0;
    }
  if (*ires == -1)
    {
      res[0] = udot[0];
      res[1] = udot[1];
    }
  else
    {
      res[0] = udot[0] + dudx[0] + dudx[1];
      res[1] = udot[1] + 4.0*dudx[0] + dudx[1];
    }
  return;
}


static void NAG_CALL bndary(Integer npde, double t, Integer ibnd, Integer nobc,
                            const double u[], const double udot[],
                            double res[], Integer *ires, Nag_Comm *comm)
{
  if (comm->user[1] == -1.0)
    {
      printf("(User-supplied callback bndary, first invocation.)\n");
      comm->user[1] = 0.0;
    }
  if (ibnd == 0)
    {
```

```
      if (*ires == -1)
        {
          res[0] = 0.0;
        }
      else
        {
          res[0] = u[0] - 0.5*(exp(t) + exp(-3.0*t))
                   - 0.25*(sin(-3.0*t) - sin(t));
        }
    }
  else
    {
      if (*ires == -1)
        {
          res[0] = 0.0;
        }
      else
        {
          res[0] = u[1] - exp(1.0 - 3.0*t) + exp(t + 1.0)
                   - 0.5*(sin(1.0 - 3.0*t) + sin(
                            t + 1.0));
        }
    }
  return;
}

static void NAG_CALL uinit(Integer npde, Integer npts, double *x, double *u)
{
  /* Routine for PDE initial values */

  Integer i;

  for (i = 1; i <= npts; ++i)
    {
      U(1, i) = exp(x[i-1]);
      U(2, i) = sin(x[i-1]);
    }
  return;
}

static void NAG_CALL exact(double t, Integer npde, Integer npts, double *x,
                           double *u)
{
  /* Exact solution (for comparison purposes) */

  Integer i;

  for (i = 1; i <= npts; ++i)
    {
      U(1, i) = 0.5*(exp(x[i-1] + t) + exp(x[i-1] - 3.0*t)) +
                0.25*(sin(x[i-1] - 3.0*t) - sin(x[i-1] + t));
      U(2, i) = exp(x[i-1] - 3.0*t) - exp(x[i-1] + t) +
                0.5*(sin(x[i-1] - 3.0*t) + sin(x[i-1] + t));
    }
  return;
}
```

10.2 Program Data

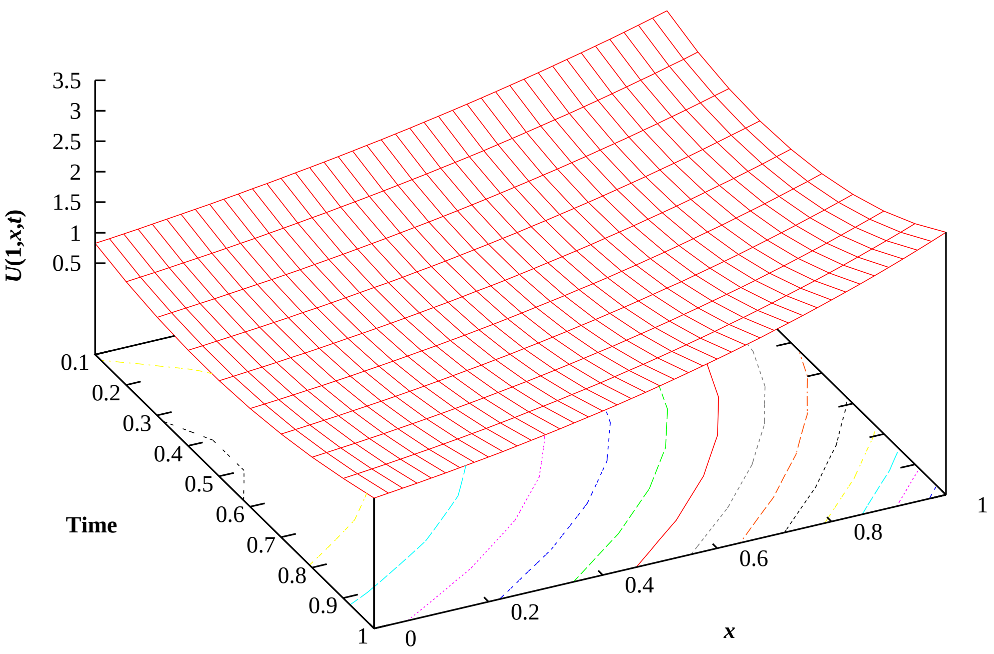
None.

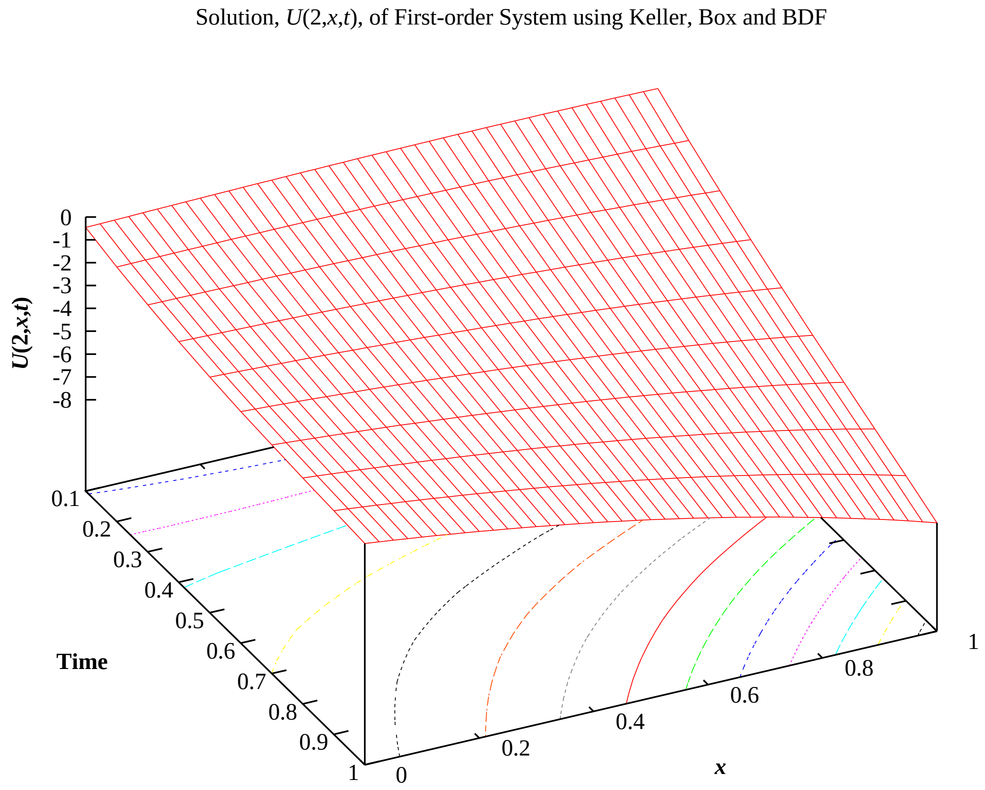
10.3 Program Results

```
nag_pde_parab_1d_keller (d03pec) Example Program Results

  Accuracy requirement =   1.000e-06 Number of points =  41

 x            0.1000    0.3000    0.5000    0.7000    0.9000
```

```
(User-supplied callback bndary, first invocation.)
(User-supplied callback pdedef, first invocation.)
 t =  0.20
 Approx u1    0.7845    1.0010    1.2733    1.6115    2.0281
 Exact  u1    0.7845    1.0010    1.2733    1.6115    2.0281
 Approx u2   -0.8352   -0.8159   -0.8367   -0.9128   -1.0609
 Exact  u2   -0.8353   -0.8160   -0.8367   -0.9129   -1.0609

 t =  0.40
 Approx u1    0.6481    0.8533    1.1212    1.4627    1.8903
 Exact  u1    0.6481    0.8533    1.1212    1.4627    1.8903
 Approx u2   -1.5216   -1.6767   -1.8934   -2.1917   -2.5944
 Exact  u2   -1.5217   -1.6767   -1.8935   -2.1917   -2.5945

 t =  0.60
 Approx u1    0.6892    0.8961    1.1747    1.5374    1.9989
 Exact  u1    0.6892    0.8962    1.1747    1.5374    1.9989
 Approx u2   -2.0047   -2.3434   -2.7677   -3.3002   -3.9680
 Exact  u2   -2.0048   -2.3436   -2.7678   -3.3003   -3.9680

 t =  0.80
 Approx u1    0.8977    1.1247    1.4320    1.8349    2.3514
 Exact  u1    0.8977    1.1247    1.4320    1.8349    2.3512
 Approx u2   -2.3403   -2.8675   -3.5110   -4.2960   -5.2536
 Exact  u2   -2.3405   -2.8677   -3.5111   -4.2961   -5.2537

 t =  1.00
 Approx u1    1.2470    1.5206    1.8828    2.3528    2.9519
 Exact  u1    1.2470    1.5205    1.8829    2.3528    2.9518
 Approx u2   -2.6229   -3.3338   -4.1998   -5.2505   -6.5218
 Exact  u2   -2.6232   -3.3340   -4.2001   -5.2507   -6.5219

 Number of integration steps in time =    149
 Number of function evaluations =    399
 Number of Jacobian evaluations =    13
 Number of iterations =    323
```

Example Program
Solution, $U(1,x,t)$, of First-order System using Keller, Box and BDF




Solution, U(2,x,t), of First-order System using Keller, Box and BDF
U(2,x,t)
0
-1
-2
-3
-4
-5
-6
-7
-8
Time
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
x
0
0.2
0.4
0.6
0.8
1