NAG Toolbox for MATLAB

d03pj

1 Purpose

d03pj integrates a system of linear or nonlinear parabolic partial differential equations (PDEs), in one space variable with scope for coupled ordinary differential equations (ODEs). The spatial discretization is performed using a Chebyshev C^0 collocation method, and the method of lines is employed to reduce the PDEs to a system of ODEs. The resulting system is solved using a backward differentiation formula (BDF) method or a Theta method (switching between Newton's method and functional iteration).

2 Syntax

```
[ts, u, x, rsave, isave, ind, user, cwsav, lwsav, iwsav, rwsav, ifail] =
d03pj(npde, m, ts, tout, pdedef, bndary, u, xbkpts, npoly, npts, ncode,
odedef, xi, uvinit, rtol, atol, itol, norm_p, laopt, algopt, rsave,
isave, itask, itrace, ind, cwsav, lwsav, iwsav, rwsav, 'nbkpts', nbkpts,
'nxi', nxi, 'neqn', neqn, 'lrsave', lrsave, 'lisave', lisave, 'user',
user)
```

3 Description

d03pj integrates the system of parabolic-elliptic equations and coupled ODEs

$$\sum_{j=1}^{\mathbf{npde}} P_{i,j}\frac{\partial U_j}{\partial t} + Q_i = x^{-m}\frac{\partial}{\partial x}(x^m R_i), \qquad i = 1,2,\ldots,\mathbf{npde}, \qquad a \leq x \leq b, t \geq t_0, \tag{1}$$

$$F_i\left(t, V, \dot{V}, \xi, U^*, U_x^*, R^*, U_t^*, U_{xt}^*\right) = 0, \qquad i = 1,2,\ldots,\mathbf{ncode}, \tag{2}$$

where (1) defines the PDE part and (2) generalizes the coupled ODE part of the problem.

In (1), $P_{i,j}$ and R_i depend on x, t, U, U_x, and V; Q_i depends on x, t, U, U_x, V and **linearly** on $\dot{V}$. The vector U is the set of PDE solution values

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

and the vector U_x is the partial derivative with respect to x. Note that $P_{i,j}$, Q_i and R_i must not depend on $\dfrac{\partial U}{\partial t}$. The vector V is the set of ODE solution values

$$V(t) = \left[V_1(t), \ldots, V_{\mathbf{ncode}}(t)\right]^{\mathrm{T}},$$

and $\dot{V}$ denotes its derivative with respect to time.

In (2), ξ represents a vector of n_ξ spatial coupling points at which the ODEs are coupled to the PDEs. These points may or may not be equal to some of the PDE spatial mesh points. U^*, U_x^*, R^*, U_t^* and U_{xt}^* are the functions U, U_x, R, U_t and U_{xt} evaluated at these coupling points. Each F_i may only depend linearly on time derivatives. Hence the equation (2) may be written more precisely as

$$F = G - A\dot{V} - B\begin{pmatrix} U_t^* \\ U_{xt}^* \end{pmatrix}, \tag{3}$$

where $F = \left[F_1, \ldots, F_{\mathbf{ncode}}\right]^{\mathrm{T}}$, G is a vector of length **ncode**, A is an **ncode** by **ncode** matrix, B is an **ncode** by $\left(n_\xi \times \mathbf{npde}\right)$ matrix and the entries in G, A and B may depend on t, ξ, U^*, U_x^* and V. In practice you need only supply a vector of information to define the ODEs and not the matrices A and B. (See Section 5 for the specification of the (sub)program **odedef**.)

The integration in time is from t_0 to t_{out}, over the space interval $a \leq x \leq b$, where $a = x_1$ and $b = x_{\textbf{nbkpts}}$ are the leftmost and rightmost of a user-defined set of break points $x_1, x_2, \ldots, x_{\textbf{nbkpts}}$. The co-ordinate system in space is defined by the value of m; $m = 0$ for Cartesian co-ordinates, $m = 1$ for cylindrical polar co-ordinates and $m = 2$ for spherical polar co-ordinates.

The PDE system which is defined by the functions $P_{i,j}$, Q_i and R_i must be specified in a user-supplied (sub)program **pdedef**.

The initial values of the functions $U(x,t)$ and $V(t)$ must be given at $t = t_0$. These values are calculated in a , user-supplied (sub)program **uvinit**.

The functions R_i which may be thought of as fluxes, are also used in the definition of the boundary conditions. The boundary conditions must have the form

$$\beta_i(x,t)R_i(x,t,U,U_x,V) = \gamma_i\big(x,t,U,U_x,V,\dot{V}\big), \qquad i = 1,2,\ldots,\textbf{npde}, \tag{4}$$

where $x = a$ or $x = b$. The functions γ_i may only depend **linearly** on $\dot{V}$.

The boundary conditions must be specified in a user-supplied (sub)program **bndary**.

The algebraic-differential equation system which is defined by the functions F_i must be specified in a (sub)program **odedef**. You must also specify the coupling points ξ in the array **xi**. Thus, the problem is subject to the following restrictions:

(i) in (1), $\dot{V}_j(t)$, for $j = 1,2,\ldots,\textbf{ncode}$, may only appear **linearly** in the functions Q_i, for $i = 1,2,\ldots,\textbf{npde}$, with a similar restriction for γ;

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

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

(iv) the evaluation of the functions $P_{i,j}$, Q_i and R_i is done at both the break points and internally selected points for each element in turn, that is $P_{i,j}$, Q_i and R_i are evaluated twice at each break point. Any discontinuities in these functions **must** therefore be at one or more of the mesh points;

(v) at least one of the functions $P_{i,j}$ must be nonzero so that there is a time derivative present in the PDE problem;

(vi) if $m > 0$ and $x_1 = 0.0$, which is the left boundary point, then it must be ensured that the PDE solution is bounded at this point. This can be done either by specifying the solution at $x = 0.0$ or by specifying a zero flux there, that is $\beta_i = 1.0$ and $\gamma_i = 0.0$.

The parabolic equations are approximated by a system of ODEs in time for the values of U_i at the mesh points. This ODE system is obtained by approximating the PDE solution between each pair of break points by a Chebyshev polynomial of degree **npoly**. The interval between each pair of break points is treated by d03pj as an element, and on this element, a polynomial and its space and time derivatives are made to satisfy the system of PDEs at $\textbf{npoly} - 1$ spatial points, which are chosen internally by the code and the break points. The user-defined break points and the internally selected points together define the mesh. The smallest value that **npoly** can take is one, in which case, the solution is approximated by piecewise linear polynomials between consecutive break points and the method is similar to an ordinary finite element method.

In total there are $(\textbf{nbkpts} - 1) \times \textbf{npoly} + 1$ mesh points in the spatial direction, and $\textbf{npde} \times ((\textbf{nbkpts} - 1) \times \textbf{npoly} + 1) + \textbf{ncode}$ ODEs in the time direction; one ODE at each break point for each PDE component, $\textbf{npoly} - 1$ ODEs for each PDE component between each pair of break points, and **ncode** coupled ODEs. The system is then integrated forwards in time using a Backward Differentiation Formula (BDF) method or a Theta method.

4 References

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

Berzins M and Dew P M 1991 Algorithm 690: Chebyshev polynomial software for elliptic-parabolic systems of PDEs *ACM Trans. Math. Software* **17** 178–206

Berzins M, Dew P M and Furzeland R M 1988 Software tools for time-dependent equations in simulation and optimisation of large systems *Proc. IMA Conf. Simulation and Optimization* (ed A J Osiadcz) 35–50 Clarendon Press, Oxford

Berzins M and Furzeland R M 1992 An adaptive theta method for the solution of stiff and nonstiff differential equations *Appl. Numer. Math.* **9** 1–19

Zaturska N B, Drazin P G and Banks W H H 1988 On the flow of a viscous fluid driven along a channel by a suction at porous walls *Fluid Dynamics Research* **4**

5 Parameters

5.1 Compulsory Input Parameters

1: **npde – int32 scalar**

The number of PDEs to be solved.

Constraint: **npde** ≥ 1.

2: **m – int32 scalar**

The co-ordinate system used:

m $= 0$

Indicates Cartesian co-ordinates.

m $= 1$

Indicates cylindrical polar co-ordinates.

m $= 2$

Indicates spherical polar co-ordinates.

Constraint: $0 \leq$ **m** ≤ 2.

3: **ts – double scalar**

The initial value of the independent variable t.

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

4: **tout – double scalar**

The final value of t to which the integration is to be carried out.

5: **pdedef – string containing name of m-file**

pdedef must compute the functions $P_{i,j}$, Q_i and R_i which define the system of PDEs. The functions may depend on x, t, U, U_x and V; Q_i may depend linearly on $\dot{V}$. The functions must be evaluated at a set of points.

```
[p, q, r, ires, user] = pdedef(npde, t, x, nptl, u, ux, ncode, v,
vdot, ires, user)
```

Input Parameters

1: **npde – int32 scalar**

The number of PDEs in the system.

2: **t – double scalar**

The current value of the independent variable t.

3: **x(nptl) – double array**

Contains a set of mesh points at which $P_{i,j}$, Q_i and R_i are to be evaluated. $\mathbf{x}(1)$ and $\mathbf{x}(\mathbf{nptl})$ contain successive user-supplied break points and the elements of the array will satisfy $\mathbf{x}(1) < \mathbf{x}(2) < \cdots < \mathbf{x}(\mathbf{nptl})$.

4: **nptl – int32 scalar**

The number of points at which evaluations are required (the value of $\mathbf{npoly} + 1$).

5: **u(npde,nptl) – double array**

$\mathbf{u}(i,j)$ contains the value of the component $U_i(x,t)$ where $x = \mathbf{x}(j)$, for $i = 1, 2, \ldots, \mathbf{npde}$ and $j = 1, 2, \ldots, \mathbf{nptl}$.

6: **ux(npde,nptl) – double array**

$\mathbf{ux}(i,j)$ contains the value of the component $\frac{\partial U_i(x,t)}{\partial x}$ where $x = \mathbf{x}(j)$, for $i = 1, 2, \ldots, \mathbf{npde}$ and $j = 1, 2, \ldots, \mathbf{nptl}$.

7: **ncode – int32 scalar**

The number of coupled ODEs in the system.

8: **v(*) – double array**

Note: the dimension of the array **v** must be at least **ncode**.

$\mathbf{v}(i)$ contains the value of component $V_i(t)$, for $i = 1, 2, \ldots, \mathbf{ncode}$.

9: **vdot(*) – double array**

Note: the dimension of the array **vdot** must be at least **ncode**.

$\mathbf{vdot}(i)$ contains the value of component $\dot{V}_i(t)$, for $i = 1, 2, \ldots, \mathbf{ncode}$.

Note: $\dot{V}_i(t)$, for $i = 1, 2, \ldots, \mathbf{ncode}$, may only appear linearly in Q_j, for $j = 1, 2, \ldots, \mathbf{npde}$.

10: **ires – int32 scalar**

Set to -1 or 1.

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

$\mathbf{ires} = 2$

Indicates to the integrator that control should be passed back immediately to the calling (sub)program with the error indicator set to $\mathbf{ifail} = 6$.

$\mathbf{ires} = 3$

Indicates to the integrator that the current time step should be abandoned and a smaller time step used instead. You may wish to set $\mathbf{ires} = 3$ when a physically meaningless input or output value has been generated. If you consecutively set $\mathbf{ires} = 3$, then d03pj returns to the calling (sub)program with the error indicator set to $\mathbf{ifail} = 4$.

11: **user – Any MATLAB object**

pdedef is called from d03pj with **user** as supplied to d03pj

Output Parameters

1: **p(npde,npde,nptl) – double array**

$\mathbf{p}(i,j,k)$ must be set to the value of $P_{i,j}(x,t,U,U_x,V)$ where $x = \mathbf{x}(k)$, for $i,j = 1,2,\ldots,\mathbf{npde}$ and $k = 1,2,\ldots,\mathbf{nptl}$.

2: **q(npde,nptl) – double array**

$\mathbf{q}(i,j)$ must be set to the value of $Q_i\left(x,t,U,U_x,V,\dot{V}\right)$ where $x = \mathbf{x}(j)$, for $i = 1,2,\ldots,\mathbf{npde}$ and $j = 1,2,\ldots,\mathbf{nptl}$.

3: **r(npde,nptl) – double array**

$\mathbf{r}(i,j)$ must be set to the value of $R_i(x,t,U,U_x,V)$ where $x = \mathbf{x}(i)$, for $i = 1,2,\ldots,\mathbf{npde}$ and $j = 1,2,\ldots,\mathbf{nptl}$.

4: **ires – int32 scalar**

Set to -1 or 1.

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 (sub)program with the error indicator set to **ifail** $= 6$.

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 d03pj returns to the calling (sub)program with the error indicator set to **ifail** $= 4$.

5: **user – Any MATLAB object**

pdedef is called from d03pj with **user** as supplied to d03pj

6: **bndary – string containing name of m-file**

bndary must compute the functions β_i and γ_i which define the boundary conditions as in equation (4).

```
[beta, gamma, ires, user] = bndary(npde, t, u, ux, ncode, v, vdot,
ibnd, ires, user)
```

Input Parameters

1: **npde – int32 scalar**

The number of PDEs in the system.

2: **t – double scalar**

The current value of the independent variable t.

3: **u(npde) – double array**

$\mathbf{u}(i)$ contains the value of the component $U_i(x,t)$ at the boundary specified by **ibnd**, for $i = 1,2,\ldots,\mathbf{npde}$.

4: **ux(npde) – double array**

ux(i) contains the value of the component $\dfrac{\partial U_i(x,t)}{\partial x}$ at the boundary specified by **ibnd**, for $i = 1, 2, \ldots,$ **npde**.

5: **ncode – int32 scalar**

The number of coupled ODEs in the system.

6: **v(*) – double array**

Note: the dimension of the array **v** must be at least **ncode**.

v(i) contains the value of component $V_i(t)$, for $i = 1, 2, \ldots,$ **ncode**.

7: **vdot(*) – double array**

Note: the dimension of the array **vdot** must be at least **ncode**.

vdot(i) contains the value of component $\dot{V}_i(t)$, for $i = 1, 2, \ldots,$ **ncode**.

Note: $\dot{V}_i(t)$, for $i = 1, 2, \ldots,$ **ncode**, may only appear linearly in Q_j, for $j = 1, 2, \ldots,$ **npde**.

8: **ibnd – int32 scalar**

Specifies which boundary conditions are to be evaluated.

ibnd $= 0$

bndary must set up the coefficients of the left-hand boundary, $x = a$.

ibnd $\neq 0$

bndary must set up the coefficients of the right-hand boundary, $x = b$.

9: **ires – int32 scalar**

Set to -1 or 1.

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 (sub)program with the error indicator set to **ifail** $= 6$.

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 d03pj returns to the calling (sub)program with the error indicator set to **ifail** $= 4$.

10: **user – Any MATLAB object**

bndary is called from d03pj with **user** as supplied to d03pj

Output Parameters

1: **beta(npde) – double array**

beta(i) must be set to the value of $\beta_i(x,t)$ at the boundary specified by **ibnd**, for $i = 1, 2, \ldots,$ **npde**.

2: **gamma(npde) – double array**

gamma(i) must be set to the value of $\gamma_i(x, t, U, U_x, V, \dot{V})$ at the boundary specified by **ibnd**, for $i = 1, 2, \ldots,$ **npde**.

3: **ires – int32 scalar**

Set to -1 or 1.

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 (sub)program with the error indicator set to **ifail** $= 6$.

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 d03pj returns to the calling (sub)program with the error indicator set to **ifail** $= 4$.

4: **user – Any MATLAB object**

bndary is called from d03pj with **user** as supplied to d03pj

7: **u(neqn) – double array**

If **ind** $= 1$ the value of **u** must be unchanged from the previous call.

8: **xbkpts(nbkpts) – double array**

The values of the break points in the space direction. **xbkpts**(1) must specify the left-hand boundary, a, and **xbkpts**(**nbkpts**) must specify the right-hand boundary, b.

Constraint: **xbkpts**(1) $<$ **xbkpts**(2) $< \cdots <$ **xbkpts**(**nbkpts**).

9: **npoly – int32 scalar**

The degree of the Chebyshev polynomial to be used in approximating the PDE solution between each pair of break points.

Constraint: $1 \leq$ **npoly** ≤ 49.

10: **npts – int32 scalar**

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

Constraint: **npts** $= ($**nbkpts** $- 1) \times$ **npoly** $+ 1$.

11: **ncode – int32 scalar**

The number of coupled ODE components.

Constraint: **ncode** ≥ 0.

12: **odedef – string containing name of m-file**

odedef must evaluate the functions F, which define the system of ODEs, as given in (3). If you wish to compute the solution of a system of PDEs only (**ncode** $= 0$), **odedef** must be the string `'d03pck'` for d03pj (or **d53pck** for d03pj). "temp_tag_xref_d03pck d53pck"done**d03pck d53pck**missing entity d03pck d53pck are included in the NAG Fortran Library.

```
[f, ires, user] = odedef(npde, t, ncode, v, vdot, nxi, xi, ucp,
ucpx, rcp, ucpt, ucptx, ires, user)
```

Input Parameters

1: **npde – int32 scalar**

The number of PDEs in the system.

2: **t – double scalar**

The current value of the independent variable t.

3: **ncode – int32 scalar**

The number of coupled ODEs in the system.

4: **v**($*$) **– double array**

Note: the dimension of the array **v** must be at least **ncode**.

v(i) contains the value of component $V_i(t)$, for $i = 1, 2, \ldots,$ **ncode**.

5: **vdot**($*$) **– double array**

Note: the dimension of the array **vdot** must be at least **ncode**.

vdot(i) contains the value of component $\dot{V}_i(t)$, for $i = 1, 2, \ldots,$ **ncode**.

6: **nxi – int32 scalar**

The number of ODE/PDE coupling points.

7: **xi**($*$) **– double array**

Note: the dimension of the array **xi** must be at least **nxi**.

xi(i) contains the ODE/PDE coupling points, ξ_i, for $i = 1, 2, \ldots,$ **nxi**.

8: **ucp**(**npde**,$*$) **– double array**

The first dimension of the array **ucp** must be at least

The second dimension of the array must be at least max(1, **nxi**)

ucp(i,j) contains the value of $U_i(x,t)$ at the coupling point $x = \xi_j$, for $i = 1, 2, \ldots,$ **npde** and $j = 1, 2, \ldots,$ **nxi**.

9: **ucpx**(**npde**,$*$) **– double array**

The first dimension of the array **ucpx** must be at least

The second dimension of the array must be at least max(1, **nxi**)

ucpx(i,j) contains the value of $\dfrac{\partial U_i(x,t)}{\partial x}$ at the coupling point $x = \xi_j$, for $i = 1, 2, \ldots,$ **npde** and $j = 1, 2, \ldots,$ **nxi**.

10: **rcp**(**npde**,$*$) **– double array**

The first dimension of the array **rcp** must be at least

The second dimension of the array must be at least max(1, **nxi**)

rcp(i,j) contains the value of the flux R_i at the coupling point $x = \xi_j$, for $i = 1, 2, \ldots,$ **npde** and $j = 1, 2, \ldots,$ **nxi**.

11: **ucpt(npde,$*$) – double array**

The first dimension of the array **ucpt** must be at least

The second dimension of the array must be at least $\max(1, \mathbf{nxi})$

ucpt(i,j) contains the value of $\dfrac{\partial U_i}{\partial t}$ at the coupling point $x = \xi_j$, for $i = 1, 2, \ldots, \mathbf{npde}$ and $j = 1, 2, \ldots, \mathbf{nxi}$.

12: **ucptx(npde,$*$) – double array**

The first dimension of the array **ucptx** must be at least

The second dimension of the array must be at least $\max(1, \mathbf{nxi})$

ucptx(i,j) contains the value of $\dfrac{\partial^2 U_i}{\partial x \partial t}$ at the coupling point $x = \xi_j$, for $i = 1, 2, \ldots, \mathbf{npde}$ and $j = 1, 2, \ldots, \mathbf{nxi}$.

13: **ires – int32 scalar**

The form of F that must be returned in the array **f**.

ires $= 1$

Equation (5) must be used.

ires $= -1$

Equation (6) must be used.

Should usually remain unchanged. However, you may reset **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 (sub)program with the error indicator set to **ifail** $= 6$.

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 d03pj returns to the calling (sub)program with the error indicator set to **ifail** $= 4$.

14: **user – Any MATLAB object**

odedef is called from d03pj with **user** as supplied to d03pj

Output Parameters

1: **f($*$) – double array**

Note: the dimension of the array **f** must be at least **ncode**.

f(i) must contain the ith component of F, for $i = 1, 2, \ldots, \mathbf{ncode}$, where F is defined as

$$F = G - A\dot{V} - B\begin{pmatrix} U_t^* \\ U_{xt}^* \end{pmatrix}, \tag{5}$$

or

$$F = -A\dot{V} - B\begin{pmatrix} U_t^* \\ U_{xt}^* \end{pmatrix}. \tag{6}$$

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

2: **ires – int32 scalar**

The form of F that must be returned in the array **f**.

ires $= 1$

Equation (5) must be used.

ires $= -1$

Equation (6) must be used.

Should usually remain unchanged. However, you may reset **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 (sub)program with the error indicator set to **ifail** $= 6$.

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 d03pj returns to the calling (sub)program with the error indicator set to **ifail** $= 4$.

3: **user – Any MATLAB object**

odedef is called from d03pj with **user** as supplied to d03pj

13: **xi**$(*)$ **– double array**

Note: the dimension of the array **xi** must be at least $\max(1, \mathbf{nxi})$.

xi(i), for $i = 1, 2, \ldots, \mathbf{nxi}$, must be set to the ODE/PDE coupling points.

Constraint: $\mathbf{xbkpts}(1) \leq \mathbf{xi}(1) < \mathbf{xi}(2) < \cdots < \mathbf{xi}(\mathbf{nxi}) \leq \mathbf{xbkpts}(\mathbf{nbkpts})$.

14: **uvinit – string containing name of m-file**

uvinit must compute the initial values of the PDE and the ODE components $U_i(x_j, t_0)$, for $i = 1, 2, \ldots, \mathbf{npde}$ and $j = 1, 2, \ldots, \mathbf{npts}$, and $V_k(t_0)$, for $k = 1, 2, \ldots, \mathbf{ncode}$.

```
[u, v, user] = uvinit(npde, npts, x, ncode, user)
```

Input Parameters

1: **npde – int32 scalar**

The number of PDEs in the system.

2: **npts – int32 scalar**

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

3: **x(npts) – double array**

x(i), for $i = 1, 2, \ldots, \mathbf{npts}$, contains the current values of the space variable x_i.

4: **ncode – int32 scalar**

The number of coupled ODEs in the system.

5: **user – Any MATLAB object**

uvinit is called from d03pj with **user** as supplied to d03pj

Output Parameters

1: **u(npde,npts) – double array**

u(i,j) contains the value of the component $U_i(x_j, t_0)$, for $i = 1, 2, \ldots,$ **npde** and $j = 1, 2, \ldots,$ **npts**.

2: **v**$(*)$ **– double array**

Note: the dimension of the array **v** must be at least **ncode**.

v(i) contains the value of component $V_i(t_0)$, for $i = 1, 2, \ldots,$ **ncode**.

3: **user – Any MATLAB object**

uvinit is called from d03pj with **user** as supplied to d03pj

15: **rtol**$(*)$ **– double array**

Note: the dimension of the array **rtol** must be at least 1 if **itol** $= 1$ or 2 and at least **neqn** if **itol** $= 3$ or 4.

The relative local error tolerance.

Constraint: **rtol**$(i) \geq 0$ for all relevant i.

16: **atol**$(*)$ **– double array**

Note: the dimension of the array **atol** must be at least 1 if **itol** $= 1$ or 3 and at least **neqn** if **itol** $= 2$ or 4.

The absolute local error tolerance.

Constraint: **atol**$(i) \geq 0$ for all relevant i.

Note: corresponding elements of **rtol** and **atol** cannot both be 0.0.

17: **itol – int32 scalar**

A value to indicate the form of the local error test. **itol** indicates to d03pj whether to interpret either or both of **rtol** or **atol** as a vector or scalar. The error test to be satisfied is $\|e_i/w_i\| < 1.0$, where w_i is defined as follows:

itol	**rtol**	**atol**	w_i
1	scalar	scalar	**rtol**$(1) \times \lvert U_i \rvert +$ **atol**(1)
2	scalar	vector	**rtol**$(1) \times \lvert U_i \rvert +$ **atol**(i)
3	vector	scalar	**rtol**$(i) \times \lvert U_i \rvert +$ **atol**(1)
4	vector	vector	**rtol**$(i) \times \lvert U_i \rvert +$ **atol**(i)

In the above, e_i denotes the estimated local error for the ith component of the coupled PDE/ODE system in time, **u**(i), for $i = 1, 2, \ldots,$ **neqn**.

The choice of norm used is defined by the parameter **norm_p**.

Constraint: $1 \leq$ **itol** ≤ 4.

18: **norm_p – string**

The type of norm to be used.

norm_p = 'M'

Maximum norm.

norm_p = 'A'

Averaged L_2 norm.

If $\mathbf{u}_{\mathrm{norm}}$ denotes the norm of the vector $\mathbf{u}$ of length **neqn**, then for the averaged L_2 norm

$$\mathbf{u}_{\mathrm{norm}} = \sqrt{\frac{1}{\mathbf{neqn}}\sum_{i=1}^{\mathbf{neqn}}(\mathbf{u}(i)/w_i)^2},$$

while for the maximum norm

$$\mathbf{u}_{\mathrm{norm}} = \max_i|\mathbf{u}(i)/w_i|.$$

See the description of **itol** for the formulation of the weight vector w.

Constraint: **norm_p** = 'M' or 'A'.

19: **laopt – string**

The type of matrix algebra required.

laopt = 'F'

Full matrix methods to be used.

laopt = 'B'

Banded matrix methods to be used.

laopt = 'S'

Sparse matrix methods to be used.

Constraint: **laopt** = 'F', 'B' or 'S'.

Note: you are recommended to use the banded option when no coupled ODEs are present (i.e., **ncode** = 0).

20: **algopt(30) – double array**

May be set to control various options available in the integrator. If you wish to employ all the default options, then **algopt**(1) should be set to 0.0. Default values will also be used for any other elements of **algopt** set to zero. The permissible values, default values, and meanings are as follows:

algopt(1)

Selects the ODE integration method to be used. If **algopt**(1) = 1.0, a BDF method is used and if **algopt**(1) = 2.0, a Theta method is used. The default value is **algopt**(1) = 1.0.

If **algopt**(1) = 2.0, then **algopt**(i), for $i = 2, 3, 4$ are not used.

algopt(2)

Specifies the maximum order of the BDF integration formula to be used. **algopt**(2) may be 1.0, 2.0, 3.0, 4.0 or 5.0. The default value is **algopt**(2) = 5.0.

algopt(3)

Specifies what method is to be used to solve the system of nonlinear equations arising on each step of the BDF method. If **algopt**(3) = 1.0 a modified Newton iteration is used and if **algopt**(3) = 2.0 a functional iteration method is used. If functional iteration is selected and the integrator encounters difficulty, then there is an automatic switch to the modified Newton iteration. The default value is **algopt**(3) = 1.0.

algopt(4)

Specifies whether or not the Petzold error test is to be employed. The Petzold error test results in extra overhead but is more suitable when algebraic equations are present, such as $P_{i,j} = 0.0$, for $j = 1, 2, \ldots,$ **npde** for some i or when there is no $\dot{V}_i(t)$ dependence in the coupled ODE system. If **algopt**(4) = 1.0, then the Petzold test is used. If **algopt**(4) = 2.0, then the Petzold test is not used. The default value is **algopt**(4) = 1.0.

If **algopt**(1) = 1.0, then **algopt**(i), for $i = 5, 6, 7$ are not used.

algopt(5)

Specifies the value of Theta to be used in the Theta integration method. $0.51 \leq$ **algopt**$(5) \leq 0.99$. The default value is **algopt**$(5) = 0.55$.

algopt(6)

Specifies what method is to be used to solve the system of nonlinear equations arising on each step of the Theta method. If **algopt**$(6) = 1.0$, a modified Newton iteration is used and if **algopt**$(6) = 2.0$, a functional iteration method is used. The default value is **algopt**$(6) = 1.0$.

algopt(7)

Specifies whether or not the integrator is allowed to switch automatically between modified Newton and functional iteration methods in order to be more efficient. If **algopt**$(7) = 1.0$, then switching is allowed and if **algopt**$(7) = 2.0$, then switching is not allowed. The default value is **algopt**$(7) = 1.0$.

algopt(11)

Specifies a point in the time direction, t_{crit}, beyond which integration must not be attempted. The use of t_{crit} is described under the parameter **itask**. If **algopt**$(1) \neq 0.0$, a value of 0.0 for **algopt**(11), say, should be specified even if **itask** subsequently specifies that t_{crit} will not be used.

algopt(12)

Specifies the minimum absolute step size to be allowed in the time integration. If this option is not required, **algopt**(12) should be set to 0.0.

algopt(13)

Specifies the maximum absolute step size to be allowed in the time integration. If this option is not required, **algopt**(13) should be set to 0.0.

algopt(14)

Specifies the initial step size to be attempted by the integrator. If **algopt**$(14) = 0.0$, then the initial step size is calculated internally.

algopt(15)

Specifies the maximum number of steps to be attempted by the integrator in any one call. If **algopt**$(15) = 0.0$, then no limit is imposed.

algopt(23)

Specifies what method is to be used to solve the nonlinear equations at the initial point to initialize the values of U, U_t, V and $\dot{V}$. If **algopt**$(23) = 1.0$, a modified Newton iteration is used and if **algopt**$(23) = 2.0$, functional iteration is used. The default value is **algopt**$(23) = 1.0$.

algopt(29) and **algopt**(30) are used only for the sparse matrix algebra option, **laopt** = 'S'.

algopt(29)

Governs the choice of pivots during the decomposition of the first Jacobian matrix. It should lie in the range $0.0 <$ **algopt**(29) < 1.0, with smaller values biasing the algorithm towards maintaining sparsity at the expense of numerical stability. If **algopt**(29) lies outside this range then the default value is used. If the functions regard the Jacobian matrix as numerically singular then increasing **algopt**(29) towards 1.0 may help, but at the cost of increased fill-in. The default value is **algopt**(29) $= 0.1$.

algopt(30)

Is used as a relative pivot threshold during subsequent Jacobian decompositions (see **algopt**(29)) below which an internal error is invoked. If **algopt**(30) is greater than 1.0 no check is made on the pivot size, and this may be a necessary option if the Jacobian is found to be numerically singular (see **algopt**(29)). The default value is **algopt**(30) $= 0.0001$.

21: **rsave**(**lrsave**) – **double 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.

22: **isave**(**lisave**) – **int32 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 required for subsequent calls. In particular:

isave(1)

Contains the number of steps taken in time.

isave(2)

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(3)

Contains the number of Jacobian evaluations performed by the time integrator.

isave(4)

Contains the order of the ODE method last used in the time integration.

isave(5)

Contains the number of Newton iterations performed by the time integrator. Each iteration involves residual evaluation of the resulting ODE system followed by a back-substitution using the LU decomposition of the Jacobian matrix.

23: **itask** – **int32 scalar**

Specifies the task to be performed by the ODE integrator.

itask $= 1$

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

itask $= 2$

One step and return.

itask $= 3$

Stop at first internal integration point at or beyond $t =$ **tout**.

itask $= 4$

Normal computation of output values **u** at $t =$ **tout** but without overshooting $t = t_{\mathrm{crit}}$ where t_{crit} is described under the parameter **algopt**.

itask $= 5$

Take one step in the time direction and return, without passing t_{crit}, where t_{crit} is described under the parameter **algopt**.

Constraint: $1 \leq$ **itask** ≤ 5.

24: **itrace – int32 scalar**

The level of trace information required from d03pj and the underlying ODE solver. **itrace** may take the value -1, 0, 1, 2 or 3.

itrace $= -1$

No output is generated.

itrace $= 0$

Only warning messages from the PDE solver are printed on the current error message unit (see x04aa).

itrace > 0

Output from the underlying ODE solver is printed on the current advisory message unit (see x04ab). This output contains details of Jacobian entries, the nonlinear iteration and the time integration during the computation of the ODE system.

If **itrace** < -1, then -1 is assumed and similarly if **itrace** > 3, then 3 is assumed.

The advisory messages are given in greater detail as **itrace** increases. You are advised to set **itrace** $= 0$, unless you are experienced with sub-chapter D02M/N.

25: **ind – int32 scalar**

Must be set to 0 or 1.

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 parameters **tout** and **ifail** should be reset between calls to d03pj.

Constraint: $0 \leq$ **ind** ≤ 1.

26: **cwsav(10) – string array**

27: **lwsav(100) – logical array**

28: **iwsav(505) – int32 array**

29: **rwsav(1100) – double array**

5.2 Optional Input Parameters

1: **nbkpts – int32 scalar**

Default: The dimension of the array **xbkpts**.

the number of break points in the interval $[a, b]$.

Constraint: **nbkpts** ≥ 2.

2: **nxi – int32 scalar**

Default: The dimension of the array **xi**.

The number of ODE/PDE coupling points.

Constraints:

if **ncode** $= 0$, **nxi** $= 0$;
if **ncode** > 0, **nxi** ≥ 0.

3: **neqn – int32 scalar**

Default: The dimension of the array **u**.

the number of ODEs in the time direction.

Constraint: **neqn** $=$ **npde** $\times$ **npts** $+$ **ncode**.

4: **lrsave – int32 scalar**

Default: The dimension of the array **rsave**.

Its size depends on the type of matrix algebra selected. If **laopt** $=$ 'F', **lrsave** $\geq$ **neqn** $\times$ **neqn** $+$ **neqn** $+ NWKRES + LENODE$.

If **laopt** $=$ 'B', **lrsave** $\geq (3 \times MLU + 1) \times$ **neqn** $+ NWKRES + LENODE$.

If **laopt** $=$ 'S', **lrsave** $\geq 4 \times$ **neqn** $+ 11 \times$ **neqn**$/2 + 1 + NWKRES + LENODE$.

Where

$MLU =$ the lower or upper half bandwidths, and
$MLU = 3 \times$ **npde** $- 1$, for PDE problems only, and
$MLU =$ **neqn** $- 1$, for coupled PDE/ODE problems.

$NWKRES = 3 \times (\textbf{npoly} + 1)^2 + (\textbf{npoly} + 1) \times \left[\textbf{npde}^2 + 6 \times \textbf{npde} + \textbf{nbkpts} + 1\right] + 8 \times \textbf{npde} + \textbf{nxi} \times (5 \times \textbf{npde} + 1) + \textbf{ncode} + 3$, when **ncode** > 0, and **nxi** > 0, and
$NWKRES = 3 \times (\textbf{npoly} + 1)^2 + (\textbf{npoly} + 1) \times \left[\textbf{npde}^2 + 6 \times \textbf{npde} + \textbf{nbkpts} + 1\right] + 13 \times \textbf{npde} + \textbf{ncode} + 4$, when **ncode** > 0, and **nxi** $= 0$, and
$NWKRES = 3 \times (\textbf{npoly} + 1)^2 + (\textbf{npoly} + 1) \times \left[\textbf{npde}^2 + 6 \times \textbf{npde} + \textbf{nbkpts} + 1\right] + 13 \times \textbf{npde} + 5$, when **ncode** $= 0$.

$LENODE = (6 + \mathrm{int}(\textbf{algopt}(2))) \times \textbf{neqn} + 50$, when the BDF method is used, and
$LENODE = 9 \times \textbf{neqn} + 50$, when the Theta method is used.

Note: when **laopt** = 'S', the value of **lrsave** may be too small when supplied to the integrator. An estimate of the minimum size of **lrsave** is printed on the current error message unit if **itrace** > 0 and the function returns with **ifail** $= 15$.

5: **lisave – int32 scalar**

Default: The dimension of the array **isave**.

Its size depends on the type of matrix algebra selected:

if **laopt** = 'F', **lisave** ≥ 24;
if **laopt** = 'B', **lisave** $\geq$ **neqn** $+ 24$;
if **laopt** = 'S', **lisave** $\geq 25 \times$ **neqn** $+ 24$.

Note: when using the sparse option, the value of **lisave** may be too small when supplied to the integrator. An estimate of the minimum size of **lisave** is printed on the current error message unit if **itrace** > 0 and the function returns with **ifail** $= 15$.

6: **user – Any MATLAB object**

user is not used by d03pj, but is passed to **pdedef**, **bndary**, **odedef** and **uvinit**. Note that for large objects it may be more efficient to use a global variable which is accessible from the m-files than to use **user**.

5.3 Input Parameters Omitted from the MATLAB Interface

None.

5.4 Output Parameters

1: **ts – double scalar**

The value of t corresponding to the solution values in **u**. Normally **ts** = **tout**.

2: **u(neqn) – double array**

The computed solution $U_i(x_j, t)$, for $i = 1, 2, \ldots,$ **npde** and $j = 1, 2, \ldots,$ **npts**, and $V_k(t)$, for $k = 1, 2, \ldots,$ **ncode**, evaluated at $t =$ **ts**, as follows:

u(**npde** $\times (j-1) + i$) contain $U_i(x_j, t)$, for $i = 1, 2, \ldots,$ **npde** and $j = 1, 2, \ldots,$ **npts**, and
u(**npts** $\times$ **npde** $+ i$) contain $V_i(t)$, for $i = 1, 2, \ldots,$ **ncode**.

3: **x(npts) – double array**

The mesh points chosen by d03pj in the spatial direction. The values of **x** will satisfy **x**(1) $<$ **x**(2) $< \cdots <$ **x**(**npts**).

4: **rsave(lrsave) – double 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.

5: **isave(lisave) – int32 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 required for subsequent calls. In particular:

isave(1)

Contains the number of steps taken in time.

isave(2)

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(3)

Contains the number of Jacobian evaluations performed by the time integrator.

isave(4)

Contains the order of the ODE method last used in the time integration.

isave(5)

Contains the number of Newton iterations performed by the time integrator. Each iteration involves residual evaluation of the resulting ODE system followed by a back-substitution using the LU decomposition of the Jacobian matrix.

6: **ind – int32 scalar**

ind $= 1$.

7: **user – Any MATLAB object**

user is not used by d03pj, but is passed to **pdedef**, **bndary**, **odedef** and **uvinit**. Note that for large objects it may be more efficient to use a global variable which is accessible from the m-files than to use **user**.

8: **cwsav(10) – string array**

9: **lwsav(100) – logical array**

10: **iwsav(505) – int32 array**

11: **rwsav(1100) – double array**

12: **ifail – int32 scalar**

0 unless the function detects an error (see Section 6).

6 Error Indicators and Warnings

Errors or warnings detected by the function:

ifail $= 1$

On entry, **tout** $-$ **ts** is too small,
or **itask** $\neq 1, 2, 3, 4$ or 5,
or **m** $\neq 0$, 1 or 2,
or at least one of the coupling point in array **xi** is outside the interval [**xbkpts**(1), **xbkpts**(**nbkpts**)],
or **npts** $\neq$ (**nbkpts** $- 1) \times$ **npoly** $+ 1$,
or **nbkpts** < 2,
or **npde** ≤ 0,
or **norm_p** $\neq$ 'A' or 'M',
or **itol** $\neq 1, 2, 3$ or 4,
or **npoly** < 1 or **npoly** > 49,
or **ncode** and **nxi** are incorrectly defined,

or **neqn** $\neq$ **npde** $\times$ **npts** + **ncode**,
or **laopt** $\neq$ 'F', 'B' or 'S',
or **ind** $\neq$ 0 or 1,
or break points **xbkpts**(i) are badly ordered,
or **lrsave** is too small,
or **lisave** is too small,
or the ODE integrator has not been correctly defined; check **algopt** parameter.
or either an element of **rtol** or **atol** < 0.0,
or all the elements of **rtol** and **atol** are zero.

ifail = 2

The underlying ODE solver cannot make any further progress, with the values of **atol** and **rtol**, across the integration range from the current point $t =$ **ts**. The components of **u** contain the computed values at the current point $t =$ **ts**.

ifail = 3

In the underlying ODE solver, there were repeated error test failures on an attempted step, before completing the requested task, but the integration was successful as far as $t =$ **ts**. The problem may have a singularity, or the error requirement may be inappropriate.

ifail = 4

In setting up the ODE system, the internal initialization function was unable to initialize the derivative of the ODE system. This could be due to the fact that **ires** was repeatedly set to 3 in at least one of the user-supplied (sub)programs **pdedef**, **bndary** or **odedef**, when the residual in the underlying ODE solver was being evaluated.

ifail = 5

In solving the ODE system, a singular Jacobian has been encountered. You should check your problem formulation.

ifail = 6

When evaluating the residual in solving the ODE system, **ires** was set to 2 in at least one of the user-supplied (sub)programs **pdedef**, **bndary** or **odedef**. Integration was successful as far as $t =$ **ts**.

ifail = 7

The values of **atol** and **rtol** are so small that the function is unable to start the integration in time.

ifail = 8

In one of the user-supplied (sub)programs **pdedef**, **bndary** or **odedef**, **ires** was set to an invalid value.

ifail = 9 (d02nn)

A serious error has occurred in an internal call to the specified function. Check the problem specification and all parameters and array dimensions. Setting **itrace** = 1 may provide more information. If the problem persists, contact NAG.

ifail = 10

The required task has been completed, but it is estimated that a small change in **atol** and **rtol** is unlikely to produce any change in the computed solution. (Only applies when you are not operating in one step mode, that is when **itask** $\neq$ 2 or 5.)

ifail = 11

An error occurred during Jacobian formulation of the ODE system (a more detailed error description may be directed to the current error message unit).

ifail $= 12$

In solving the ODE system, the maximum number of steps specified in **algopt**(15) have been taken.

ifail $= 13$

Some error weights w_i became zero during the time integration (see the description of **itol**). Pure relative error control (**atol**$(i) = 0.0$) was requested on a variable (the ith) which has become zero. The integration was successful as far as $t =$ **ts**.

ifail $= 14$

The flux function R_i was detected as depending on time derivatives, which is not permissible.

ifail $= 15$

When using the sparse option, the value of **lisave** or **lrsave** was not sufficient (more detailed information may be directed to the current error message unit).

7 Accuracy

d03pj 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 parameter **atol** and **rtol**.

8 Further Comments

The parameter specification allows you to include equations with only first-order derivatives in the space direction but there is no guarantee that the method of integration will be satisfactory for such systems. The position and nature of the boundary conditions in particular are critical in defining a stable problem.

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

9 Example

d03pj_bndary.m

```
function                    [beta,gamma,ires,user]                    =
bndary(npde,t,u,ux,ncode,v,vdot,ibnd,ires,user)
    beta = zeros(npde,1);
    gamma = zeros(npde,1);
    beta(1) = 1.0;
    if (ibnd == 0)
       gamma(1) = -v(1)*exp(t);
    else
       gamma(1) = -v(1)*vdot(1);
    end
```

d03pj_odedef.m

```
function [f,ires,user] = ...
        odedef(npde,t,ncode,v,vdot,nxi,xi,ucp,ucpx,rcp,ucpt,ucptx,ires,
user)
  f = zeros(ncode,1);
  if (ires == 1)
     f(1) = vdot(1) - v(1)*ucp(1,1) - ucpx(1,1) - 1.0 - t;
  elseif (ires == -1)
     f(1) = vdot(1);
  end
```

d03pj_pdedef.m

```
function                          [p,q,r,ires,user]                    =
pdedef(npde,t,x,nptl,u,ux,ncode,v,vdot,ires,user)
  p = zeros(npde,npde,nptl);
  q = zeros(npde,nptl);
  r = zeros(npde,nptl);

  for i=1:nptl
     p(1,1,i) = v(1)*v(1);
     r(1,i) = ux(1,i);
     q(1,i) = -x(i)*ux(1,i)*v(1)*vdot(1);
  end
```

d03pj_uvinit.m

```
function [u,v,user] = uvinit(npde,npts,x,ncode,user)
  u = zeros(npde,npts);
  ts = 0.1e-6;

  v(1) = ts;

  for i = 1:npts
    u(1,i) = exp(ts*(1.0-x(i))) - 1.0;
  end
```

```
npde = int32(1);
m = int32(0);
ts = 0.0001;
tout = 0.2;
u = zeros(22,1);
xbkpts = [0; 0.1; 0.2; 0.3; 0.4; 0.5; 0.6; 0.7; 0.8; 0.9; 1];
npoly = int32(2);
npts = int32(21);
ncode = int32(1);
xi = [1];
rtol = [0.0001];
atol = [0.0001];
itol = int32(1);
normtype = 'A';
laopt = 'F';
algopt = zeros(30,1);
rsave = zeros(900, 1);
isave = zeros(24, 1, 'int32');
itask = int32(1);
itrace = int32(0);
ind = int32(0);
cwsav = {''; ''; ''; ''; ''; ''; ''; ''; ''; ''};
lwsav = false(100, 1);
iwsav = zeros(505, 1, 'int32');
rwsav = zeros(1100, 1);
[tsOut, uOut, x, rsaveOut, isaveOut, indOut, user, ...
 cwsavOut, lwsavOut, iwsavOut, rwsavOut, ifail] = ...
     d03pj(npde, m, ts, tout, 'd03pj_pdedef', 'd03pj_bndary', u, xbkpts,
...
     npoly, npts, ncode, 'd03pj_odedef', xi, 'd03pj_uvinit', rtol, atol,
itol, ...
    normtype, laopt, algopt, rsave, isave, itask, itrace, ind, ...
    cwsav, lwsav, iwsav, rwsav)
```

```
tsOut =
    0.2000
uOut =
    0.2217
    0.2096
```

```
     0.1975
     0.1856
     0.1738
     0.1622
     0.1506
     0.1392
     0.1278
     0.1166
     0.1055
     0.0945
     0.0836
     0.0729
     0.0622
     0.0516
     0.0412
     0.0308
     0.0206
     0.0104
     0.0004
     0.1999
x =
          0
     0.0500
     0.1000
     0.1500
     0.2000
     0.2500
     0.3000
     0.3500
     0.4000
     0.4500
     0.5000
     0.5500
     0.6000
     0.6500
     0.7000
     0.7500
     0.8000
     0.8500
     0.9000
     0.9500
     1.0000
rsaveOut =
      array elided
isaveOut =
            15
           259
             9
             2
            50
            50
             0
             1
             0
             0
             0
             0
             0
             0
             0
             0
             0
             0
             0
             0
             0
             0
             0
             0
indOut =
```

```
           1
user =
     0
cwsavOut =
    ''
    ''
    [1x80 char]
    [1x80 char]
    [1x80 char]
    ''
    ''
    ''
    ''
    ''
lwsavOut =
     array elided
iwsavOut =
     array elided
rwsavOut =
     array elided
ifail =
           0
```